University of Twente

Computer Science

FollowMe!

Distributed Movement Coordination in Wireless Sensor and Actuator Networks

Master Thesis

May 15, 2008

Author
Stephan Bosch

Supervisors
Hans Scholten
Raluca Marin-Perianu
Mihai Marin-Perianu
Paul Havinga

Abstract

In this thesis, we explore the potential of WSAN technology in dynamic applications requiring a sense-and-react control loop by considering the problem of distributed movement coordination of vehicles on wheels. The final goal is to obtain a self-organizing group (or swarm) of mobile nodes that maintain a formation by periodically exchanging sensed movement information. Such a coordinated team of mobile wireless sensor and actuator nodes can bring numerous benefits for various applications in the field of cooperative surveillance, mapping unknown areas, disaster management, automated highway and space exploration.

The work we present in this thesis is considers a pair of vehicles that synchronize their movements, thus trying to maintain a specific formation. We use toy cars as prototype vehicles for this research. One of the two cars acts as the *leader*, being remotely controlled by the user, and the other one represents the *follower*, which implements a control loop for mimicking the leader's movements. Each vehicle has a low-power wireless sensor node attached, featuring a 3D accelerometer and a magnetic compass. Velocity and heading are computed on both vehicles in real time using inertial navigation techniques. The leader periodically transmits its measurements to the follower, which implements a lightweight fuzzy logic controller for imitating the leader's movement pattern. This solution is not restricted to vehicles on wheels, but could support any moving entities capable of determining their velocity and heading. We call our initial implementation *FollowMe*. We report in detail on all development phases, covering design, simulation, implementation and testing. We perform the tests on our university's running track and hockey field.

The simulations and tests show that the system exhibits the desired leader-follower behavior, using just the compact, low-cost wireless sensors and actuators. Making use of a fuzzy controller facilitates the implementation on resource constrained sensor nodes and handles robustly the noisy sensor data as well as the rough mechanical capabilities of the vehicles. The vehicle simulations we devised show adequately realistic results and provide a means to test, evaluate, tune and debug the controller. However, some issues are still of concern, such as the effect of inclination changes on the sensor readings, the accumulating errors seen in the velocity estimate, the sensitivity of the compass sensor to external influence and the stability and the overall performance of the follower controller. Also, the simulations are open for improvement towards more realistic physical models of the vehicles.

Samenvatting

In deze scriptie verkennen wij de potentie van WSAN technologie in dynamische toepassingen die een reactieve besturingslus vereisen. Dit doen wij door het bestuderen van gedistribueerde bewegingscoordinatie van voertuigen op wielen. Het uiteindelijke doel is om een zichzelf organiserende groep (of zwerm) van mobiele robots te verkrijgen die in een formatie kunnen blijven bewegen door periodiek bewegingsinformatie uit te wisselen. Een dergelijk gecoördineerd team van mobiele WSAN nodes kan vele voordelen hebben voor diverse toepassingen, zoals coöperatieve beveiliging, onbekende gebieden in kaart brengen, rampenbestrijding, de automatische snelweg en de verkennende ruimtevaart.

Het onderzoek dat wij in deze scriptie presenteren heeft betrekking op een tweetal voertuigen die elkaars bewegingen synchroniseren en dus geen specifieke formatie onderhouden. Wij gebruiken speelgoedauto's als prototype voertuigen. Een van de twee auto's doet dienst als de *leader* (leider), die op afstand bestuurd wordt door de gebruiker, en de ander doet dienst als the *follower* (volgeling), die een besturingslus implementeert om de bewegingen van de leader te imiteren. Aan elk voertuig is een energie-efficiënte draadloze sensor node bevestigd voorzien van een driedimensionale versnellingsmeter en een compas sensor. De snelheid en koers worden op beide voertuigen realtime berekend met behulp van traagheidsnavigatietechnieken. De leader stuurt periodiek zijn metingen naar de follower die een, qua benodigde rekenkracht, lichtgewicht fuzzy logic besturingseenheid gebruikt om het bewegingspatroon van de leader te imiteren. Deze oplossing is niet beperkt tot alleen voertuigen op wielen, maar zou gebruikt kunnen worden voor alle denkbare bewegende objecten die hun snelheid en koers kunnen bepalen. Wij noemen onze initiële implementatie *FollowMe*. Wij beschrijven alle ontwerpfasen in detail en daarbij behandelen wij het ontwerp, simulaties, de implementatie en tests.

De simulaties en tests tonen aan dat het systeem het gewenste leader-follower gedrag vertoont, alleen gebruik makend van kompacte, goedkope, draadloze sensors en actuators. Fuzzy logic maakt de weg vrij voor de implementatie van de besturingslus op zeer beperkte hardware en verdraagt op robuuste wijze de relatief hoge ruisniveaus van de sensors en de ruwe mechanische eigenschappen van de voertuigen. De voertuigsimulaties die wij bedacht hebben vertonen adequaat realistische resultaten en voorzien in de mogelijkheid om de besturingseenheid te testen, te evalueren, af te stellen en van software problemen te ontdoen. Echter, enkele problemen zijn nog steeds bron van zorg, zoals het effect van inclinatieveranderingen op sensor metingen, de accumulerende fouten in de geschatte snelheid, de gevoeligheid van het compas voor externe invloeden en de stabiliteit en de algemene prestaties van de bestuuringseenheid van de follower. Tevens zijn de simulaties nog te verbeteren om realistischere fysieke modellen van de voertuigen te verkrijgen.

Contents

Chapter 1

Introduction

The concept of Wireless Sensor and Actuator Networks (WSANs) [1] is to use a wireless network of small and resource-limited sensor and actuator nodes which can perform distributed sensing and actuation tasks. The preceding and more restricted Wireless Sensor Networks (WSNs) [2] employed networked sensor nodes wich could only gather information on the physical world, process the collected data and transfer the results back to one or more destinations in the network called *sinks*. WSANs extend WSN research with the addition of actuators that use the sensed information about the environment to take decisions and then perform appropriate actions on that environment. This yields interesting research topics regarding the coordination between the sensors and the actuators. The emergent market of WSANs targets a large number of applications, ranging from transport and logistics to industrial processes [16], and even planetary sensing or space exploration [17]. Even if limited in terms of computational power, memory, energy and communication bandwidth, the sensor and actuator nodes collaborate in the network, in order to accomplish complex collaborative control tasks at scale.

In this work we explore the potential of WSAN technology in dynamic applications requiring a *sense-and-react* control loop. More specifically, we consider the problem of distributed movement coordination of vehicles on wheels, equipped with wireless sensors and actuators. The final goal is to have a self-organizing group (or swarm) of nodes that maintain the formation by exchanging periodically their sensed movement information. Application domains include cooperative surveillance, clearing mines [11], mapping unknown areas [7], disaster control, and the automated highway [20].

For simplicity, we assume that there exists a *leader* whose trajectory has to be copied by the *followers*. Both leader and followers sense their movements using inertial sensors, so GPS technology is not used. Compared to the related work in the robotics community, we specifically utilize only small, low-cost, low-power, low-resolution inertial sensors: accelerometers and magnetic compasses. Using WSN technology, the leader communicates its measurements to the followers, which try to copy the leader's movements using their actuators. This is where the sense-and-react control loop is used: a follower measures its own movements and tries to match these to the leader's using its actuators.

In this work we demonstrate the leader-follower concept by designing and implementing a simple version of the control loop with only one follower node. We call our implementation *FollowMe*. The design of FollowMe is evaluated using simulation and actual tests with two remote-controlled toy cars. For the tests, one toy car is modified to give a WSN node the means to control its actuators: it will act as the follower. The other remote-controlled toy car is left unmodified and the leader node is mounted on it.

Due to the resource limitations of the sensor nodes, simplicity is key in this work; it is not feasible to build an elaborate sense and control loop for the followers. For its robustness and simplicity, we envision using fuzzy logic as a feasible means to implement the control loop on the follower nodes.

In summary, we devise a miniaturized, low-cost navigation system using low-power wireless sensor nodes equipped with accelerometers and magnetic compasses. In this work we explore fuzzy logic as a lightweight and robust control solution for coordinating the group movement in a leader-follower fashion. This thesis describes all development phases, covering simulation, controller tuning, inertial sensor evaluation, calibration, scheduling, debugging and benchmarking. At the end, we evaluate the performance of our prototype

system through field experiments and we discuss the most important results.

In the remainder of this chapter the problems, the challenges, the solution and the expected results of this work are described in detail along with a survey of related work. The final section describes the overall structure of the remainder of this thesis.

1.1 Problem

The scope of this research is limited to the use of only two vehicles: the leader and one follower. The leader vehicle is not autonomous and is to be remotely controlled. Both vehicles are equipped with sensor nodes that measure their movements using inertial sensors. The leader periodically communicates its measurements to the follower. The follower node must follow the leader's movements autonomously using the communicated measurements, while maintaining a linear train-like formation at a predetermined distance from the leader.

This project has the following goals:

- Obtain sufficiently accurate inertial measurements at both nodes, such that the follower node can follow and maintain a linear formation with the leader robot, i.e. directly behind the leader in convoy-like manner.
- Wirelessly communicate the measurements from leader to follower in real-time.
- Design, simulate, implement and test a movement control loop for a follower node to make it successfully follow the leader's movements in a formation. It must be able to deal with the accumulation of errors that is very common in inertial movement measurements.
- Evaluate the overall performance of the system using real field tests.

For this project, the following assumptions are made:

- The nodes will never encounter any obstacles in their path.
- No sloping surfaces are ever encountered by the nodes. This means that the inclination of the node-controlled vehicles is assumed to remain relatively constant and level.
- The surface of the driving range is sufficiently smooth, meaning that the node-controlled vehicles will never shake and rattle heavily when driving.
- No GPS or other navigation system is assumed to be available other than a magnetic compass.
- The system is not subjected to major external disturbances.

The problem is simplified further as follows:

- Only two nodes are to be used in the experiment.
- Consequently, only single-hop communication is considered.
- All nodes in the network are physically identical, including the equipped sensors.
- The mobile nodes in the network will use acceleration and compass sensors to measure acceleration and heading respectively. Inter-node distance needs to be measured as well, but currently no means is available to do so. Therefore, the followers do not actually track the leader's path, but rather mimic its movements.

1.2 Challenges

A solution to the problems presented in the previous section faces the following challenges:

- Scarce resources: What makes this project unique, when compared to common robotics research, is the use of WSN technology. This primarily means that the processing power at a single node is fairly limited, both in terms of computational and memory resources. Also, the radio communication

bandwidth is very low: at most a few hundred kbit/s. The energy constraints commonly associated with WSNs are a less profound issue, since the vehicle's actuators consume much more power than the processor and the radio. Therefore, a bigger battery should allow for more design freedom.

- Accuracy and frequency of movement measurements: An important prerequisite for our design to work is the availability of a means to accurately measure the movements of the leader and follower node. To be able to follow the leader, the follower needs to obtain information on the traveled distance, the current speed or acceleration of the leader along with the direction of its movement. To actually follow the leader, the follower needs to match its own movements to those measured by the leader. The accuracy and frequency of these movement measurements at leader and follower are of great importance: if the measurement precision is too low or if the current view of reality is not updated, i.e. measured, often enough, the follower is bound to lose track of the leader.
- Trade-off between the frequency and accuracy of measurements and the required communication bandwidth: The leader and follower communicate wirelessly to exchange measurements. This communication channel must provide enough bandwidth to transfer the measurements at the required accuracy and frequency. Clearly, there is a trade-off between the frequency and accuracy of measurements and the required communication bandwidth. A suitable communication protocol needs to be designed.
- Rapid response time: The follower needs the ability to control its own movements quickly in response to incoming measurements from the leader. If the follower's controller produces its control commands with a large delay, the follower will not be able to almost instantaneously mimic the leader's movements, causing it to lose track. Practical tests need to be performed to show what values are acceptable for this delay and what the resulting velocity and heading deviation can be.

These challenges are much interwoven. We need to find sensors that provide sufficient accuracy and bandwidth, but also keep the use of resources like CPU and power consumption within limits. Very fast and accurate sensors are of little use if processing the resulting measurements cannot be scheduled on the node's CPU. However, using sensors with low accuracy or bandwidth will have a detrimental effect on the effectiveness of the follower. This means that the follower will lose track of the leader earlier. The trade-off between the frequency and accuracy of sensor measurements and the required communication bandwidth also extends to resource usage. If messages are sent from leader to follower with high frequency, the node's CPU will spend more time transmitting or receiving messages. This makes scheduling other tasks like data acquisition and control more difficult.

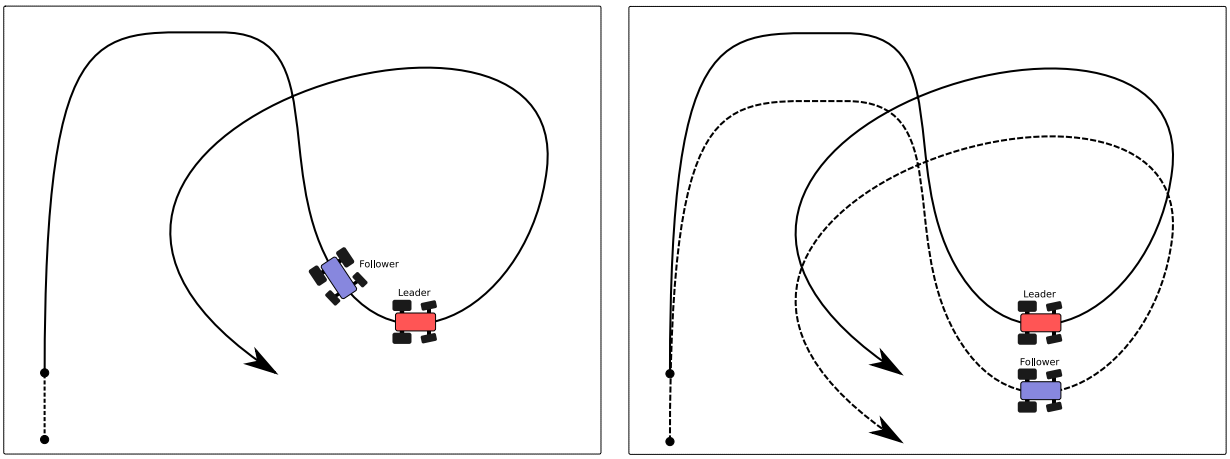
Designing and implementing a suitable controller for the follower is not a trivial task on the limited node hardware. An elaborate controller design will likely fail to achieve an appropriate response time and its execution will have a profound impact on the resource usage on the node. Furthermore, the controller must be able to operate with the relatively inaccurate and noisy measurements from the inexpensive, low-power inertial sensors.

The delay between the leader's actions and the follower's responses will cause tardy reactions. Also, the wireless channel is fairly unreliable, so movement data might get lost in transmission. Furthermore, the sensor measurements are bound to be inaccurate. These and other factors will ultimately result in the follower node completely losing track of the leader. This means that the distance between leader and follower will gradually increase to an unacceptable level. A mechanism to detect and resolve this situation is necessary.

1.3 Solution Overview

The ultimate goal of this project is to let one actuator node follow the other at constant distance along the same path in a convoy-like manner. This is depicted in Figure 1.1(a). In previous research (e.g. [15] and [36]), the follower nodes follow the leader by detecting it visually or only measuring the distance to it. For this project, the idea is to let the leader actively communicate local inertial movement measurements to the follower node.

The convoy leader-follower behavior presented in Figure 1.1(a) will not be achieved when both nodes synchronously maintain the same heading and velocity. The follower is required to delay its movements to make it follow the same path: if the follower were to change its heading at the same time as the leader for



(a) Convoy (b) Synchronous

Figure 1.1: Leader-follower movement coordination.

instance, it would turn too early and leave the leader's path. The length of this delay would depend on the velocities of leader and follower and the distance between them. Therefore, just instantly mimicking the leader's movements is not sufficient for the follower to achieve the real leader-follower behavior. It is, however, a good starting point for this research. Achieving synchronous movement is a prerequisite for the feasibility of the future convoy implementation. Figure 1.1(b) graphically shows how perfect synchronous movement control relates to the convoy behavior.

We choose to use an acceleration sensor to infer the velocity of the nodes. This is performed by integrating the acceleration measurements into a velocity estimate. With the advent of Micro-Electro-Mechanical Systems (MEMS) [34] technology, very small and power-efficient digital accelerometers have become available, making the use of an accelerometer feasible for sensor nodes. Using integration has a significant limitation: the obtained velocity estimate will accumulate all sensor errors until it exceeds what can be considered a valid estimate. We solve this by assuming that the vehicles stop regularly, providing an opportunity to reset the velocity to a zero value.

Alternatively, we could measure the speed directly, for example by measuring the number of wheel revolutions per unit of time using a so-called odometer. This provides a more absolute velocity measurement that stays valid no matter how long the vehicles are driving. However, this requires making physical changes to both toy cars, as most of these don't have the means to measure wheel revolutions[1]. Using an accelerometer makes our solution suitable for any type of wheeled vehicle, but integrating acceleration data results in the accumulation of error. In future work, we could combine our velocity integration with direct velocity measurement technology to improve performance.

We use electronic compass sensors to measure the heading of the nodes. Electronic compasses are highly sensitive magnetic sensors capable of measuring the magnitude of weak magnetic field of the earth with high accuracy. Such compass sensors can be accurate, small and used at a low level of power consumption. This makes such sensors very useful four our application. Unfortunately, the high sensitivity of the sensors makes them highly susceptible to distortion from stray magnetic fields and nearby ferrous metal objects.

We could alternatively use gyroscopic sensors to keep track of the heading of the vehicles. However, just like inferring velocity from acceleration data, inferring an angular position from the gyroscope output involves integration, because gyroscopes measure angular velocity and not an absolute angle. Thus, this integration will accumulate error and therefore the estimated heading will correspond less to the actual heading as time progresses. In contrast, compass sensors can provide a more absolute reference. In future work we could combine the use of compass and gyroscope to obtain a more reliable common result, using sensor data fusion for example [19, 25, 30].

To achieve the actual leader-follower behavior in which the nodes follow the same path and to limit the distance between the nodes, it would be useful to have a means to measure that distance. The movements

[1] Coincidentally, our follower car does have an odometer which we do not use in this work

of the nodes are measured using acceleration and compass sensors. Unfortunately, these sensors do not provide any information on the distance between the nodes. For maintaining a leader-follower formation, this information is of great importance. Therefore, some other sensor mechanism to measure inter-node distance will be necessary in future work.

Matching the leader's speed and following the leader's heading can be viewed as two separate design goals. The idea for this project is to keep the design simple and therefore these two concerns are implemented as two separate simple controllers: one for matching the leader's velocity and another for steering towards the leader's direction of movement. In Chapter 6, however, we explain that these two controllers can influence each other's performance in practice and we provide a simple solution.

Various alternatives exist for implementing the *sense-and-react* type of controller we need. Our previous research [33] has shown that fuzzy logic can provide a simple, robust and flexible control solution that is feasible for WSN nodes. In this work we show that, in addition to distributed decision making, fuzzy control can be applied successfully to a WSN for distributed movement coordination. We elaborate on the application of fuzzy control to our design in Chapter 6.

The sensors that measure the movements of both nodes need to provide a minimum level of accuracy and frequency for the follower to have adequate performance. For this initial work, we are aiming for a follower that manages to mimic the leader's movements at a distance of no more than a few meters for at least 30 seconds. An accelerometer is used to infer the velocity of the nodes by means of integration; therefore both its measurement resolution and its sample frequency influence the accuracy of the velocity measurement. The requirements for the acceleration measurements to obtain sufficiently accurate velocity integration are outlined in Chapter 4.

For the compass sensor, the sample frequency is less crucial. As we show in Chapter 2, this sensor provides very stable heading measurements if it is used properly. It is sufficient that the sensor can be sampled at the rate the controller is executed. However, the compass sensor must provide enough heading accuracy to allow the follower to match the leader's heading during the time required. At a consistent relative heading error of up to one degree, the follower will deviate from the leader by less than 2 m at a linearly traveled distance of 100 m. Therefore, we are confident that single degree accuracy is sufficient. Note that the consistency between leader and follower of the movement measurements is more important than their absolute accuracy. If both vehicles simultaneously incur the same errors, the follower's controller acts on correct data.

In Chapter 2, we describe the toy car vehicles we use for the work. The vehicle we use for the leader node is very agile and therefore the follower needs a relatively low response time to mimic its movements. That response time is mainly dictated by the frequency at which the controller is executed. On the one hand, if this frequency is too high the sensor node cannot provide the required processing power and, on the other hand, for efficiency it is best to keep this frequency as low as possible. We briefly analyze the control frequency requirement in Section 6.2.

In summary, for this research only a leader and one follower node is to be implemented. We use an accelerometer and a compass sensor on each node to measure velocity and heading respectively. We aim for a follower that achieves synchronized movement while maintaining a distance of no more than a few meters for at least 30 seconds. The follower's controller will be modeled as two separate fuzzy logic controllers that try to match the speed and direction of the leader node respectively. Although this does not achieve the desired follower behavior, it is a good point to start this research. When a means to measure the distance between the nodes becomes available, it is possible to limit the distance between both nodes and to implement a follower that actually tries to trace the leader's path.

1.4 Expected Results

As we outline above, this project is currently limited to synchronizing the movements of a leader and just one follower. The choice for using WSN technology and small, cheap, low-power and relatively unreliable inertial sensors imposes limits on what we can achieve. Also, using integration to obtain a velocity estimate from acceleration measurements is error prone and will affect the performance of the follower. The assumption that the vehicles stop regularly to reset the velocity estimate will bound this effect, but the observed errors can still be very significant.

If, for example, the leader accumulates more positive error in its velocity estimate than the follower, the follower will accelerate to a velocity beyond that of the leader if possible, thus breaking movement synchrony. This can also happen when the follower accumulates more negative error than the leader, making its perception of its own velocity too low. Of course, velocity integration errors can also cause the follower to fall behind, or even drive in the opposite direction if the errors are more significant than and opposite to the actual velocity.

The compass is not a problem with respect to accumulating errors, meaning that errors are transient and the vehicles will be able to find a common heading even when they are initially facing in a different direction. Consequently, if the compass sensor is working adequately, at least the heading should be synchronous at all times. This holds of course only as long as the follower is still driving. If the follower for instance stops or drives much slower than the leader, it will not be able to steer at all or quick enough to match the leader's changing heading.

Compass sensors are, however, very sensitive to metal and magnetic influence. Since both vehicles can never be at the same exact location, these influences will differ if present. And, both accelerometers and compass sensors are affected by changes in their inclination. If this is not compensated, the results are invalid and most probably different for the individual cars. That is why we choose to assume that our driving field is level and no metal or magnetic objects are present (other than the cars themselves). This should eliminate these problems for the most part and provide reliable heading and velocity estimates.

Our mobile nodes are assumed to be identical in Section 1.1, but as we describe in Chapter 2, the vehicles are not quite the same for practical reasons. This means that we probably incur different sensor errors and other sensor-related problems on the respective toy cars. This can have a detrimental effect on how synchronous the movement of the cars is. Additionally, the capabilities of the cars differ, opening the possibility that the follower can physically not always synchronize to what the leader is doing.

However, if we manage to keep the heading and velocity errors for both vehicles within the same boundaries, we should be able to achieve synchronous movement for at least a short period of time. This requires our test field to be approximately level and free from metal and magnetic objects. And when the leader car is not driven beyond the capability of the follower, the 30 seconds we aim for is deemed to be a realistic goal for this work.

Concretely, we expect the following results to emerge from this work:

- We expect the control loop to be feasible on the resource constrained sensor nodes as long as the control and data processing algorithms are kept simple and computationally fast. The solution we present in this work that uses fuzzy control and simple navigation algorithms to infer heading and velocity should satisfy these requirements.
- We expect the follower to synchronize with sufficient accuracy. Depending on the achieved accuracy, frequency and reliability of the sensors and the controller's response time, the follower will be successful in synchronizing to the leader's movements during the required period. The main concern are the reliability of the sensor measurements and the frequency of the acceleration measurements for the integration. We expect the reliability of the compass to be adequate as long as it is properly calibrated and the encounter of metal objects is avoided. Both accelerometer and compass sensor are sensitive to inclination changes, but if the driving surface is smooth and level as assumed in Section 1.1, the measurements will be sufficiently reliable. Although the required accelerometer sampling frequency is bound to be relatively high, the processing algorithms do not need to run at that same frequency. Since acquiring a single sample is a very small task, sufficiently high frequencies will be attainable.
- We expect to gain much practical experience on using inertial sensors in combination with wireless sensor network technology. We should be able to classify sources of noise and unreliable sensor readings and the provide solutions to mitigate these effects.

1.5 Related Work

The field of robotics accounts for extensive related work on swarms (or flocks) of robots [5]. The common objective is to achieve formation control and make multiple robots move collectively. Various technologies

are used to determine inter-robot distances and orientations for the purpose of maintaining a formation: modulated infrared light [42], acoustic signals among submersibles [28], omnidirectional cameras and physical attachments [35]. Such robots typically feature powerful computing boards with clock frequencies ranging from 40 to 400 MHz. In comparison, our solution targets a more loosely-coupled coordination among resource-constrained devices, using only inertial sensors and low-power wireless communication.

In recent work, Allred *et al.* [3] describe SensorFlock, which is composed of small bird-sized nodes called micro-air vehicles (MAVs). The MAVs are expected to communicate wirelessly and follow a certain trajectory mission plan. The flight control system fuses information from the GPS and gyroscope sensors on board. SensorFlock focuses on keeping the nodes autonomously in the air and provides an in-depth study of the RF characteristics and networking connectivity. In comparison, we focus on inertial sensing and explore fuzzy logic as a lightweight yet robust control method for coordinating the movements of mobile nodes in the field.

Wang *et al.* [52] overview several mobile WSAN platforms based on MICA motes and running TinyOS, such as: MASmote, MICAmote, CotsBots and Robomote. The MAS-net project also considers the usage of MASmotes for formation control [51], including a leader-follower strategy. The MASmotes rely on a pseudo-GPS location system (based on cameras) and odometry to measure node displacement. In contrast, our solution is fully localized and does not require any infrastructure.

Fuzzy control for autonomous vehicles has also been considered by previous research. Kodagoda *et al.* [27] present an autonomous golf car controlled by a 450 MHz PC. Simulations of similar systems are provided in [22] and [13]. Compared to these approaches, our system is much more resource constrained, exploits wireless communication and assumes a simpler physical vehicle model.

1.6 Structure of this Document

Throughout this document we employ a bottom-up approach for the description of our system's design. First, in Chapter 2, we provide an overview of the hardware involved for our research. We present the vehicles, the wireless sensor nodes and the sensors we use for this work. In the subsequent four chapters, the software of the sensor nodes is described starting with the low-level driver software, data processing and ranging up to the communication protocol and the design of the follower controller.

The software of the leader and follower is similar in many respects. The only differences are the vehicle controller included in the follower and the fact that the follower needs to receive messages. Both nodes need to continuously measure their movements, process the measured data to remove any noise or other external influences and finally calculate crisp movement status like the current velocity and heading. On the leader, these steps are directly followed by a broadcast of its movement status over the radio channel. The follower performs an additional step by invoking its controller with the incoming data from its sensors and the last message from the leader. The follower's software controller produces control signals that are applied on its actuators. Subsequently, the follower broadcasts its own movement status. Messages broadcasted from both nodes are picked up and stored by the gateway, which is a third node in the network connected to an external computer.

In summary, the main tasks that are implemented in software are to:

1. Sample the sensors at the required rates. We explain this in Section 3.1.
2. Process the incoming sensor data into a usable form. This task involves simple filtering and the inference of the velocity and the heading of the nodes from the sensor data. In Chapter 4, we outline these techniques.
3. Communicate heading and velocity of the leader to the follower and from both nodes to the gateway. We outline the communication design in Chapter 5.
4. Control the follower using local and remote movement status. This is described further in Chapter 6. The actual use of the vehicle's actuators is described in Section 3.2.

Before taking theory to practice, the performance of follower's controller is first evaluated using simulation. Apart from evaluation, the simulations are also useful for finding the cause of problems that rise during actual tests in the field: all necessary control data is logged and in simulation the follower's controller can

be simulated using the logged data to reproduce the problem. We discuss the various simulations and their applications and results in Chapter 7.

We describe the actual implementation of the system in Chapter 8. In that chapter we provide a detailed description of the hardware and software. We describe the practical problems that we encountered including the solutions we devised. We evaluate the resulting system using tests with the two vehicles outside. A description of these tests is presented in Chapter 9 along with a presentation of their respective results.

Finally, in Chapter 10, we asses whether our goals for this work are achieved, we evaluate our design and provide a survey of open issues and recommendations for future work.

Chapter 2

Hardware

In this chapter, we provide a detailed description of the hardware portion of the solution. A global overview of the hardware is shown in Figure 2.1. The system consists of two mobile vehicles: the leader and the follower. The leader communicates its local movement measurements to the follower through a wireless radio channel. This is where the wireless sensor network technology comes in. Both vehicles carry a wireless sensor node. On the leader the node is strictly an observer that measures the leader's movements. On the follower, the sensor node can also actively control the vehicle. The nodes on both vehicles measure the movements of their vehicle, but only the node on the follower has the means to control its vehicle. The leader node actively and continuously sends its measurements over the wireless sensor network to the follower. The follower compares the incoming data to its own local measurements and controls its vehicle in an effort to follow the leader's movements.

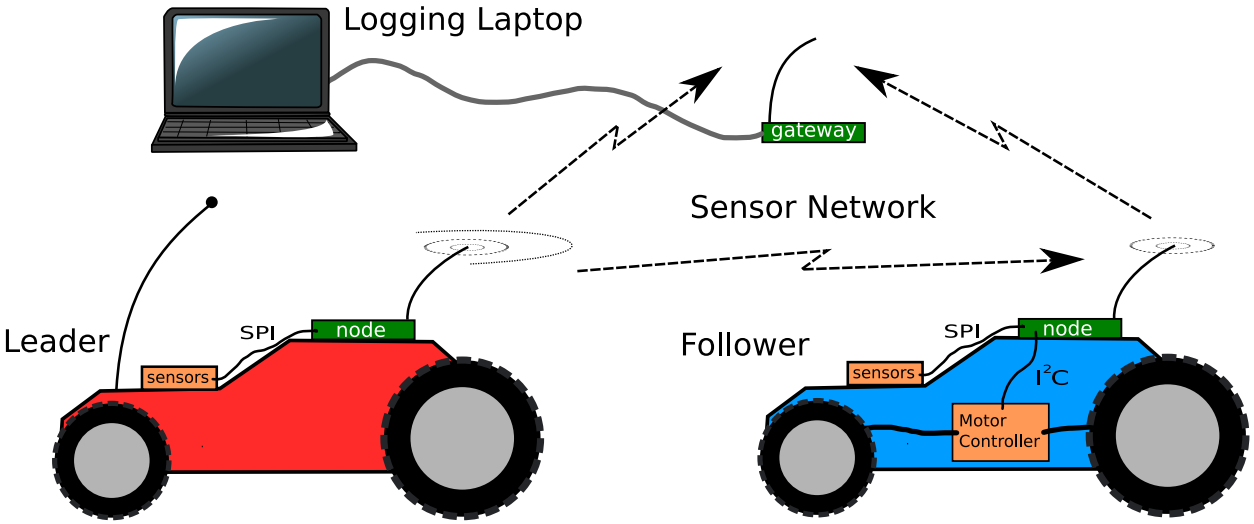


Figure 2.1: An overview of the hardware

There exists also a third node in the wireless sensor network that is used to monitor and log the transmissions of both vehicles. This is the so-called gateway and it is connected to an external (laptop) computer to store and evaluate the incoming data. Not only the leader transmits its local measurements over the network; the follower does so as well to give the gateway node the ability to log exactly what is going on in the network of the two vehicles. The resulting data logs are used in the simulations and evaluations described in Chapters 7 and 9 respectively.

First, in Section 2.1, we present the wireless sensor nodes we use for this work. Subsequently, in Section 2.2, we provide details on the toy cars that we use as vehicles. The rest of this chapter is dedicated to describing and evaluating the used acceleration and compass sensors.

2.1 Wireless Sensor Nodes

This research uses the wireless sensor nodes developed by Ambient Systems [4]. These are called μNodes and use the MSP430F1611 micro-controller from Texas Instruments [48]. This controller has 48 Kb flash ROM and 10 Kb static RAM. Additionally, the node is equipped with an EEPROM for permanently storing custom data. Figure 2.2 provides a picture of one of these μNodes.

The node is equipped with a 868 MHz radio transceiver, which, ideally, can achieve transmission over a distance of a few hundred meters at a data rate of 100 Kbps at a maximum power level of +10 dBm. The hardware dictates a message size of 32 bytes, but the CPU of the node does not need to worry about encoding the bits on the channel and a CRC check is performed by the radio already.

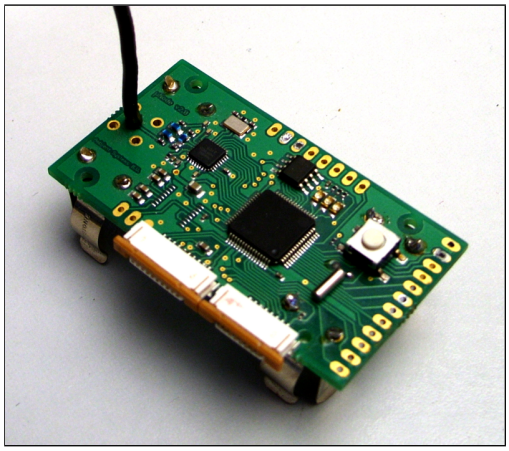


Figure 2.2: A μNode by Ambient Systems.

A node provides external access to 16 of it's processor's general purpose I/O ports. In addition, on special nodes a serial port is available for communication with a normal computer. Such nodes are called gateway nodes. All nodes work with a supply voltage and signal levels of about 3 V with a maximum of 3.6 V. Sensors connected to the node must match these specifications.

At our department at the University of Twente, the Ambient μNodes are the default platform for research on wireless sensor networks. This makes them easily available for this research. Furthermore, as this hardware is part of much research at our department, we have a vast amount of experience on using it.

Ambient Systems has developed an operating system for the μNodes called AmbientRT [23]. It uses a real-time scheduling algorithm, making it well suited for time-constrained embedded applications like the application we describe here. Being a wireless sensor network operating system, AmbientRT can also provide network functionality. It either provides low-level access to the radio hardware or it provides access to a well-defined radio stack.

Unfortunately, AmbientRT is not open source, making it difficult to make changes to the operating system if the need arises. However, this is unlikely. Also, the network stack is not freely available, forcing us to define our own simple communication protocol (refer to Chapter 5).

2.2 Vehicles

Both the leader and the follower node are attached to large toy cars for our experiments. For practical reasons, the vehicles are not identical as shown in Figure 2.3. Both vehicles have front-wheel steering and rear-wheel drive. As shown in the figure, the leader is a model of a red Ferrari. The follower is different and stripped from its hull to mount the hardware. Both cars are equipped with a large battery that provides energy for the car's motors and controller. The leader has a 19.2 V battery whereas the follower operates at 9.6 V. Relative to the leader, the follower car is has a much lower maximum velocity and it is less agile in

terms of steering.

Figure 2.3: A picture of the leader (left) and follower (right) vehicles at the test location.

The leader and follower have the same set of sensors mounted on the vehicle, not necessarily on exactly the same position (refer to Chapter 8 for details). The leader is left largely unmodified, apart from the mounted node and sensors. In contrast, the follower vehicle is equipped with a custom motor controller in addition to the node and sensors. This controller is directly responsible for executing the requested motor actions. It is a substitute for the remote control receiver that originally came with the toy car. The follower node communicates with the motor controller through an I^2C 2-wire bus as shown in Figure 2.1.

The follower's motor controller is described in more detail in Section 3.2 and, including circuit schematics, in Appendix A. We provide a short explanation here. Like the original controller, the motor controller controls the actuation of the drive and steering motors using a pulse-width modulated (PWM) output signal. The drive motor is the actuator that makes the vehicle drive either forward or backward. The controller can also short-circuit this motor to implement an inductive brake.

The steering motor directs the front wheels of the follower car in the requested direction for steering. On the toy car we use as follower this motor is not a servo, but rather a normal motor that pushes the wheels to the left or to the right. When the steering motor is inactive, the wheels are centered using a spring. The intensity at which the motor is driven determines how far the spring is deflected from its center position and thus how far the wheels are turned.

Simple steering tests that involve quick heading changes show that the vehicles can steer at a rate faster than 90°/s (see also Section 9.2.1). Especially the leader is very agile, with the ability to steer even faster than that, depending on its current velocity.

2.3 Sensors

In Chapter 1 we choose to use a compass and an acceleration sensor to measure the movements of the leader and follower nodes. These two types of sensors are the bare minimum to achieve the synchronous leader-follower behavior described in Section 1.1. In this section we show the actual compass and acceleration sensors that we use and we provide a brief explanation of how these work internally. Practical initial tests with these sensors are outlined in Section 2.4

Inclination Problems

As described in more detail in the subsequent sections, both the electronic compass and the accelerometer are sensitive to changes in their inclination. A non-level inclination can be calibrated out when the sensors are stationary, but when the sensors move along with the vehicles, changes in the inclination cannot be compensated for.

The use of a gyroscopic sensor [49] could significantly improve the performance of the designed system in this respect. Using such a sensor, changes in the pitch, roll and yaw angles of the sensors could be measured, giving the system the ability to keep track of changes in the sensors' inclination and compensate. Unfortunately, we are currently unable to find an affordable three-axis gyroscopic sensor with a power supply voltage of about 3 volts and reasonable power consumption. Also, continuous inclination compensation involves relatively complex calculations that are as of yet unsure to be feasible on the used sensor nodes.

The accelerometer itself can also be used to asses the inclination angles of the sensors [50]. However, this requires the sensor to be stationary. Otherwise, vibration noise and the sensor's true acceleration add to the result and distort the inclination measurement. This could be mended using a steep low-pass filter, but this will cause a significant processing delay, making the inclination measurements lag behind the actual inclination.

Consequently, the design of the system will not make use of a gyroscope or any other means to measure the inclination of the sensors dynamically. Therefore, we assume that the sensors on the vehicles only incur minor changes in inclination and that the vehicles are stationary at regular intervals, allowing the sensors to be re-calibrated.

Distance Measurement

In Chapter 1, the future need to measure distance between the nodes is coined. Although many alternatives exist to measure the distance, most of these not practical for our design. Note that very high accuracy measurements are not a requirement; it is sufficient to obtain an estimate on the distance between the nodes. For practical reasons, it is very important that the employed means to measure that distance does not require that the nodes have specific orientations with respect to each other. This for example makes the options that involve laser or infra-red range finders less practical.

Currently, the most promising solution involves measuring radio channel parameters like the received signal strength (RSSI) to obtain an estimate on the distance. Note, however, that the RSSI method is commonly assessed to be unreliable due to the high variability of the signal strength measurements at constant distance [31]. The causes for this variability include the presence of obstacles, multi-path signal propagation and frequency-selective fading. According to [31], variability in radio hardware and the orientation of the antennae are also important factors. That work also discusses the effects of three-dimensional node placement, indicating that RSSI further approaches uselessness in those 3D situations. Our work is limited to nodes following each other on the same surface, however, and therefore, assuming all node antennas point upwards, those 3D effects should be less significant.

Our current design omits using distance measurements for follower control. However, it is sensible to keep this future extension in mind. Our sensor nodes do not include the means to gather RSSI data on radio transmissions, so either a new type of node is necessary or an additional radio specifically tailored for distance measurements. The follower sensor board that we show in Section 8.1 provides room for an additional SPI radio module.

2.3.1 Sensor Bus

As we show in the subsequent sections, both the acceleration and the compass sensor use the SPI[1] bus to connect to the sensor node. SPI is a synchronous serial bus standard, which, in its most basic form, uses a

[1]Probably due to its simplicity, SPI is a *de facto* standard, meaning that it is not agreed by any international committee or even described in a single specification document. Various aspects of the hardware and protocol are optional and different among the various implementations. Refer to the data sheets of the sensors we use for more information [41, 46].

four-wire connection: a clock signal, a serial input signal, a serial output signal and a slave enable signal. If more than one device is to be accessed on the same bus, a separate slave enable signal is required for each device attached to the bus. The enable signals are used by the bus master, in this case the sensor node, to activate the device it wants to communicate with. In this section we give a brief overview of how we connect the sensors to a node using a shared SPI bus. In Appendix A.1, this connection is described in detail with a circuit diagram.

To signal events to the master, many SPI slave devices have an additional output signal which is commonly connected to one of the master's interrupt inputs. Both the acceleration and the compass sensor provide this functionality to signal that their measurement is ready. In Section 3.1 we describe, however, that these signals are connected to a normal input of the sensor node, thus no interrupt is generated. An overview of how the sensor devices are connected to the μNode is presented in Figure 2.4.

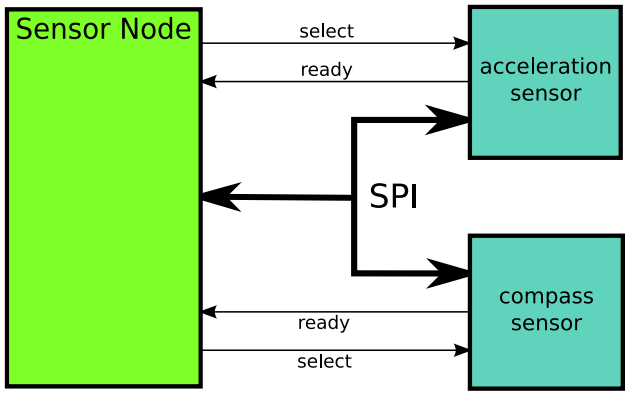


Figure 2.4: The SPI bus connecting the acceleration and compass sensors to the sensor node.

Although the μNode's controller has two hardware SPI interfaces available, none of these are made available to the application by the operating system. That is why the SPI bus is implemented in software on the sensor node. A software implementation is slower and less efficient than an implementation using a hardware interface. However, for our purposes, the required bandwidth is not high and thus this is no problem.

2.3.2 Acceleration Sensor

We use an acceleration sensor, commonly known as an accelerometer, to obtain an estimate on the current velocity of a node. This is achieved by integrating the acceleration measurements, as we describe in Section 4.1.

Accelerometer Theory

An accelerometer is an inertial sensor as it uses no absolute external reference to measure acceleration. Acceleration is measured as the deflection of a proof mass or as the necessary force to keep the proof mass in place. If the sensor is subjected to acceleration, forces are exerted on the mass. The magnitude of these forces is inferred from the deflection of the proof mass or from the forces applied to counter this deflection. Because the proof mass is of known value, the measured force is directly proportional to the acceleration of the sensor.

Unfortunately, not only the current acceleration of the sensor is represented at the output of an accelerometer. The gravitational acceleration also has a persistent influence on the sensor. According to [44], an accelerometer actually measures the so-called specific force, defined as the time rate of change of velocity relative to local gravitational space. Concretely this means that the specific force is a combination of the gravitational acceleration and the acceleration of other forces exerted on the object (the ones we need to measure). The specific force is sometimes used synonymous to acceleration, but that does not take the effect of gravity into account.

For example, when the accelerometer lies stationary on the floor, its true acceleration is zero, but it will sense a specific force value equal in magnitude to the local gravitational acceleration vector. According to Newton's second and third laws, the floor exerts an upward force on the accelerometer equal to the sensor's mass times the local gravitational acceleration. When stationary, this gravitational acceleration is what the accelerometer measures as specific force. Now suppose the sensor is in free-fall in a vacuum. This means that absolutely no external forces are exerted on the sensor and the sensor will read zero. When its not falling inside a vacuum, it will incur aerodynamic friction and thus it will not measure zero but rather the acceleration caused by the air drag.

If the gravitational acceleration vector is known, it can be subtracted from the measured specific force to obtain the actual acceleration vector. However, this gravitational vector cannot be inferred from the sensor measurement when other acceleration components are part of the measurement, i.e. when the sensor is moving. There is not enough information available to do so. If the gravitational vector remains constant in magnitude and direction, i.e. the inclination of the sensor remains constant, the gravitational vector can be measured at instances the sensor is stationary and used in subsequent measurements. This situation is not very realistic however. At the beginning of Section 2.3, we propose the application of a gyroscopic sensor to tackle this problem, as commonly done in inertial navigation systems[49]. For this work however, we assume that the inclination of our accelerometers does not significantly change when attached to the nodes by choosing an appropriate testing ground.

Note that this explanation implicitly considers an accelerometer with the ability to measure acceleration with an arbitrary direction. An elemental accelerometer unit can only measure the specific force in a single direction. A sensor that can measure acceleration in any direction used three such elemental sensors fixed in orthogonal orientations. Thus, the actual specific force is measured as its individual three-dimensional vector components. Such triadic sensors are available in a single package.

For our research we need to use an accelerometer with at least two measurement axes. Because a node can steer in any direction, a single axis accelerometer is not enough. A third axis would be necessary if we would need to calculate the magnitude and direction of the gravity vector in stationary conditions. In stead, we take sensor measurements at instances that the nodes are stationary and subtract these values from subsequent measurements. This calibrates the sensor for the current inclination. We could assume that the inclination remains absolutely constant all the time, but in reality it will gradually change: perfectly level surfaces simply do not exist in practice. Consequently, the calibration needs to be done each time the nodes are stationary to obtain the best possible results with our calibration strategy. We describe this strategy in greater detail in Section 4.1.3.

The LIS3LV02DQ Three-Axis Accelerometer

We choose to use the LIS3LV02DQ three-axis accelerometer from ST Microelectronics [46]. Because this sensor uses Micro-Electromechanical Systems (MEMS) [34] technology, it is very small compared to the traditional mechanical versions which are relatively large, power-inefficient and expensive. This sensor can be interfaced digitally using an SPI or I^2C bus. It uses a supply voltage of 3 volts, which matches the supply voltage of the sensor nodes. Many other MEMS accelerometers have a supply voltage of at least 5 volts and an analog or PWM[2] interface. That makes interfacing those more difficult.

This accelerometer comes in a very small package that makes building a prototype design unpractical. Fortunately, readily made sensor boards are available that have a mounted accelerometer with the necessary peripheral electronics. Such breakout boards are available from Sparkfun Electronics for about $ 45 each [45]. Figure 2.5 shows such a breakout board with the LIS3LV02DQ accelerometer[3].

The LIS3LV02DQ can measure accelerations of up to 2 or 6 g on each axis in either direction, depending on the configured mode. The measured specific force is represented as a 12 bit value. It can sample at rates of up to 2560 Hz. With a supply voltage of 3.3 volt, the sensor typically draws 650 μA of current.

[2]Refer to Appendix A.2.1 for an explanation of what PWM is. Although the appendix describes its use for power control, it can also be used as a digital interface.

[3]Picture courtesy of Sparkfun Electronics (http://www.sparkfun.com).

Figure 2.5: The LIS3LV02DQ three-axis accelerometer on the Sparkfun breakout board.

Accelerometer Operation

Although the chosen sensor can be sampled at very high frequencies, we only need a relatively low sample frequency to achieve adequate velocity integration. In Chapter 4, we describe what sample frequency is adequate.

Internally, the sensor can only be configured for a specific set of sample frequencies: 40 Hz, 160 Hz, 640 Hz and 2560 Hz. When connected to a master device, like our sensor node, it is not the master who dictates the sample frequency: the sensor contains its own oscillator and it continuously writes the measurements to its output registers. The master must make sure it reads the incoming measurements before these are overwritten by the next measurements. One can decide to read the sensor at a lower frequency, discarding some of the samples. However, this will cause jitter in the sample period if the sensor sample frequency is not an integer multiple of the frequency at which the master reads the data.

In Section 2.4.2, we evaluate the performance of the LIS3LV02DQ using a series of simple practical tests. These simple tests provide a basis for the determination of a proper sensor configuration in terms of sample frequency and acceleration measurement range. These choices are motivated in Chapter 4.

2.3.3 Compass Sensor

For measuring the current heading of the nodes, we choose to use a magnetic compass sensor. Luckily, this does not involve a conventional model with a needle, but rather a solid-state electronic compass. This means that the compass is resistant to vibration, that the sensor can directly be interfaced to the sensor node and most importantly that it can be relatively small and power-efficient.

Basic Compass Theory

The measurement principle of an electronic compass is no different from the traditional needle compasses: it uses the earth's magnetic field to find a bearing towards the magnetic north pole. This is possible because the earth's magnetic field has a vector component parallel to the earth's surface that always points to the magnetic north [8]. While the component parallel to the surface always points north, the actual magnetic field vector points into the ground on the northern hemisphere, is parallel to the surface on the equator and points into the air on the southern hemisphere. The angle between the earth's magnetic field vector and the earth's surface at a specific location is called the *inclination* angle. For compass navigation purposes, we are only interested in the component parallel to the earth.

Note that the 'north' is explicitly referenced to as the 'magnetic north pole'. That is done because the magnetic north pole is not on the same location as the actual north pole, i.e. at the north side of the earth's rotation axis. Consequently, there is an difference between the bearing to the north pole that the compass measures and the actual bearing to the north pole. This angular difference is called the *declination* angle. The declination depends on the location on earth. To obtain the actual bearing to north, a look-up table is commonly used [8]. Luckily, we are not interested in the bearing to the actual north pole. We use the

compass bearing as a firm reference to determine whether both leader and follower are pointing in the same direction. If they are, they should measure the same compass heading, regardless of their common position on earth.

In summary, to find the current compass bearing of our nodes, it is necessary to measure the vector component of the earth's magnetic field that is parallel to the earth's surface. A compass sensor uses magnetic sensing elements called magnetometers to measure the magnitude of a magnetic field. Much like the accelerometer we describe in the previous section, these elemental magnetometers can only measure this magnetic field magnitude along a single direction. Therefore, solid-state electronic compasses all consist of two or three of these magnetometer sensors mounted perpendicularly, one for each measurable axis referenced to the sensor's package. Each magnetometer will measure a component of the total magnetic field vector. If such a compass sensor is held level, i.e. with the x and y axis magnetometers parallel to the earth surface, the x and y axis combined will measure the magnetic field vector parallel to the earth surface.

To find the angle of the sensor relative to the bearing towards the magnetic north pole, it would be sufficient to calculate the arctangent of these two axis measurements. Note that the actual magnitude of the magnetic field has no influence on the calculated heading. However, there is no way to guarantee that the compass sensor is always held perfectly level. If the sensor is not held level, the component of the magnetic field perpendicular to the earth's surface will influence the measurement.

An important disadvantage of compass sensors is their susceptibility to magnetic and ferrous interference. The earth's magnetic field is fairly weak, meaning that other relatively weak magnetic fields can significantly interfere with the sensor. Also, nearby ferrous metal (iron, nickel or cobalt) objects will distort the magnetic field, causing the sensor to measure erroneous field magnitudes on its axes. If the ferrous interference is constant, i.e. the ferrous objects are part of the vehicle on which the sensor is mounted, the errors can be compensated for with relative ease. For dynamic effects, like driving past metal or magnetic objects there is no straightforward way to compensate.

In Section 2.4.1, we provide an evaluation of the compass sensor's behavior when subjected to interference and to what extent the inclination of the sensor influences the measured magnetic field vector. In Section 4.2, we describe compass heading calculation and interference in more detail and we provide a simple solution.

The Micromag 3 Magnetometer

Although multiple electronic compass sensor models exist, we could only find one that operated at the required supply voltage of 3 Volts: the Micromag 3 manufactured by the PNI Corporation [41]. This is a relatively cheap three-axis magnetometer module (about $ 60 each) and it draws about 500 μA at 3 Volt supply voltage. The sensor module has a digital SPI interface, making the use of this sensor for this application fairly straightforward as we do not have to deal with analog interfacing problems. Figure 2.6 shows the Micromag3 module[4].

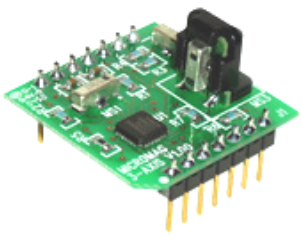


Figure 2.6: The PNI Micromag3 three-axis magnetic field sensing module.

The PNI Micromag 3 sensor module employs the Magneto-Inductive (MI) magnetic field sensing technique briefly described in [10] and in more detail in [39] and [40]. This technique uses a coil with a single winding

[4]Picture courtesy of the PNI Corporation (http://www.pnicorp.com).

to sense the external magnetic field component parallel to the coil's core. The effective inductance of the coil varies with the applied external magnetic field. The coil is part of an oscillator circuit in which it determines the oscillation frequency. This frequency is measured in the sensor module's controller by determining the time needed for a specific number of signal periods to occur. This interval is measured using the sensor's internal high frequency 16 bit *measurement counter*. The number of required signal periods is configurable and called the *period count*[5].

To compensate for temperature effects, to make the sensor largely linear over its configured measurement range and to produce a convenient zero-centered sensor output value, the coils are successively biased in two directions. This yields two time measurement results and the difference of these is presented as the output value for a sensor axis[6]. If the magnetic field is zero, the oscillator oscillates at a middle frequency and the forwards and backwards biased measurements are identical. This yields a zero result. The oscillator output frequency will change when a magnetic field is applied, depending on the sensor's coil bias direction and the direction of the magnetic field. For example, in forward bias the frequency will be larger than the middle value and in reverse bias the frequency will be smaller than the middle value for a specific magnetic field vector. The earlier described subtraction now becomes offset, yielding a non-zero result corresponding to the direction and magnitude of that magnetic field.

When the period count is chosen too high for a strong magnetic field parallel to the measured axis, the measurement counter will overflow. That is why the period counter determines the maximum field strength that can be measured for an axis in either direction. The resolution of the magnetic field measurement also depends on the value of the period count. If the period count is increased, a larger part of the measurement counter's 16 bit range is used for measurement of the same frequency range and thus the resolution of the measurement increases.

Additionally, the period count influences the measurement delay, because with a specific output frequency from the oscillator it will take more time to complete a measurement when a higher period count is required. Similarly, the field strength also influences the measurement delay: if the output frequency of the oscillator resulting from an applied magnetic field is lower, more time is needed to achieve the desired period count. According to the data sheet [41], the measurement delay will reach its worst-case value when there is no magnetic field present. This worst-case measurement delay value determines the maximum sampling frequency.

Looking at the described effects of choosing a period count, there is a trade-off between the maximum sample rate and the accuracy of the individual measurements. And, unlike most three-axis accelerometers, only one axis can be measured at a time for this compass sensor. The cause of this is that the sensor module is equipped with a single measurement counter only [40]. Consequently, a change in the worst-case measurement delay has a three-fold influence on the maximum achievable sample rate. In Section 2.4.1, we show the resolution effect of the period count value with a few practical tests. In Section 4.2 we evaluate the trade-off between the resolution and sample rate and choose a period count configuration suitable for our application.

Compass Operation

The Micromag 3 is useful for various magnetic sensing applications, but we are interested only in its compass sensor abilities. Although the sensor value is represented in a signed 16 bit format, only a portion of this range will be used when the earth's magnetic field is measured exclusively. The earth's magnetic field is fairly weak with a total magnitude of about 60 μT [9]. As described earlier, for compass operation we are only interested in the field component parallel to the earth surface. Knowing that the inclination on the northern hemisphere is approximately 70° [8], the component parallel to the earth's surface is about 21 μT.

According to the data sheet [41], the sensor's internal measurement counter counts 62.48 for every μT (gain value) at the highest period count configuration. When subjected to the earth's magnetic field only, the sensor will measure field strengths in the range $[-21 \dots 21]$ μT when it is held perfectly level

[5] Internally, this is implemented as a configurable frequency divider which can divide the frequency from the oscillator by a factor ranging from 32 to 4096. The output of the divider is connected to the enable signal of the measurement counter to stop it when the desired number of periods was received [40].

[6] Actually, the 16 bit measurement counter counts up on the first measurement and it counts down on the second measurement. Then, after the second measurement, the result is directly available in the measurement counter [40].

and $[-60 \dots 60]$ μT when it is held in an arbitrary orientation. This yields a digital output range of $[-60 * 62.48 \dots 60 * 62.48] = [-3748 \dots 3748]$. This lies well within the bounds of the sensor's signed 16 bit output representation. Therefore, the sensor's measurement range is always adequate to measure the earth's magnetic field, regardless of the period count configuration. The choice for a proper period count depends only on the trade-off between the achieved resolution and the achievable sample rate.

The period count can be configured at 8 values ranging from 32 to 4096. Each value is a duplicate of the previous one. Due to the proportional relation between the period count value and the gain of the sensor, the gain value of 62.48 is halved when the period count is halved. Thus for a period count of 2048 the gain value is 31.24 counts/μT. The magnetometer resolution for a single compass axis is equal to the reciprocal of this gain value. Table 2.1 shows all available period count values and the achieved magnetometer resolution for each axis. The table also lists the worst case measurement delay that results from the possible configurations. This dictates the attainable maximum sampling frequency[7].

Period Count	**Resolution (μT)**	**Maximum Delay (ms)**
32	2.1	0.5 ms
64	1.0	1.0 ms
128	0.51	2.0 ms
256	0.27	4.0 ms
512	0.13	7.5 ms
1024	0.064	15 ms
2048	0.032	35.5 ms
4096	0.016	60 ms

Table 2.1: The available period count configurations for the Micromag 3 compass sensor along with the resulting worst case measurement delay and magnetometer resolution for an individual axis [41].

When compared to the previously discussed accelerometer, the Micromag 3 is slow. For proper velocity integration, it is desirable to measure the acceleration at high frequency. Therefore, it is important that the compass sensor does not dictate the sample frequency for the accelerometer as well. For the software implementation, this could prove to be problematic. Luckily, the Micromag 3 can be instructed to start a measurement in the background. The SPI bus does not need to remain occupied by the compass sensor during measurement and therefore the node's processor can continue sampling the accelerometer several times while the compass sensor is measuring. This is discussed in greater detail in Chapter 3.

2.4 Prototype Test Platform

Before we build a definitive implementation of the leader-follower implementation, we implement a simple test platform to acquire experience with the sensor hardware. We evaluate the performance of the sensors and the ability of the μNode hardware to process the data on this platform. In this section we present a short overview of the prototype design and the results that we obtain from testing the sensors with this platform. We also test the initial version of the heading calculation with this prototype, but this is described in Chapter 4 where this algorithm is first presented. A somewhat similar platform was built by Chang [12] for testing an accelerometer connected to a sensor node.

The prototype consists of three nodes of which one is the gateway connected to a laptop computer by means of a serial connection. The gateway laptop is responsible for logging the stream of measurements into a file for later evaluation. The two other nodes are equipped with a compass sensor and an accelerometer. Using a very simple static 4-slot TDMA protocol, the three nodes can communicate. Unlike for the definitive leader-follower implementation, all nodes can send and receive messages in this prototype. We choose to use a wireless means of communication for this prototype to have the full freedom to place the nodes anywhere.

To obtain the best possible measurement results, we choose to aim for the highest attainable sample frequency. This is limited particularly by the MAC protocol and to what extent this can be scheduled on the node's processor. Through extensive tuning of scheduling and protocol parameters we obtain a protocol

[7]Keep in mind that only one axis can be sampled at a time. Consequently, the maximum delay of a full sensor reading is equal to three times that delay value plus overhead to start and stop the measurements for each individual axis.

frequency of 25 Hz, meaning that each node can send 25 messages per second. Each communicated message can be 32 bytes long, of which 2 bytes are occupied by protocol overhead. Consequently, only 30 bytes remain for measurement payload. Both compass and acceleration measurements need to be communicated to the gateway.

Keep in mind that such a high communication frequency is unrealistic for a real WSN with more than these three nodes. With 4 slots in each frame of the TDMA protocol, 100 slots occur in one second. Each node sends in its own slot and receives in all the others to resynchronize. Much of the CPU's available processing time is spent handling the MAC protocol, even though the the actual raw radio activity is performed in an external radio chip (refer to Section 2.1).

The sampling frequency attainable with the accelerometer is, according to its data sheet, much higher than the protocol frequency of 25 Hz. If we want to sample the accelerometer faster than the measurements can be communicated individually, multiple measurements need to be accumulated into a single message. An accelerometer or compass sensor measurement is 2 bytes wide for each axis[8]. We can fit 4 successive accelerometer measurements and one compass sensor measurement in the outgoing message, which matches the number of slots in the MAC protocol. During each slot, the accelerometer is sampled and, during the three slots in which the node is not transmitting, the three axis of the compass sensor are sampled successively.

In summary, the maximum sampling frequency of the accelerometer is 100Hz for this prototype and the sampling frequency of the compass sensor is 25 Hz.

2.4.1 Compass Sensor Evaluation

In this section we demonstrate how the compass sensor works. We discuss an example of the compass sensor's output, we demonstrate how the compass is influenced by nearby metal objects, we evaluate the effect of the sensor's period count configuration and finally we discuss the effect of the sensor's inclination.

Output Example

The easiest way to evaluate the performance of a compass is to let it make several full rotations in an approximately level plane. The Micromag3 compass sensor we use is a three-axis magnetometer, so it will yield three separate raw measurement streams. For the Micromag3, the z axis points upwards and therefore one might expect to see a sinusoidal signal only on the x an y axes when the sensor is rotated. However, this is obviously only true when the sensor is perfectly rotated around its z axis. Otherwise, the z axis will show a sinusoidal signal as well. The 70° inclination of the earth's magnetic field that we describe in Section 2.3.3 causes the z axis to measure a significantly higher offset.

Figure 2.7 shows an example of the compass sensor output. The sensor was placed at the center of a table and turned manually a few times. The x and y axes show the most significant sinusoidal signal. As expected, the z axis changes during movement as well with much smaller amplitude, yet at a high offset.

It is also evident from Figure 2.7 that the x and y axis signals do not have a zero offset and that they differ in amplitude. This is caused by factors like nearby large metal surfaces, magnetic fields other than the earth's, non-level inclination and manufacturing differences in the sensors.

The compass signals are mostly smooth without noticeable noise with levels not exceeding more than a few bits values in the output signal. However, sometimes large single-sample spikes occur in the output signal. A small example of this is shown in 2.7 around 17 s. The cause of these is not certain, but it is assumed to be some sort of transient external influence.

The table on which this example experiment is performed is constructed with metal bars. The tabletop is made from a synthetic non-ferrous material, but during the tests we do not consistently keep the sensor in

[8] Actually, the accelerometer measurements are 12 bits wide and the compass sensor measurements rarely exceed 9 bits. But, to keep the prototype simple, the byte boundaries are adhered.

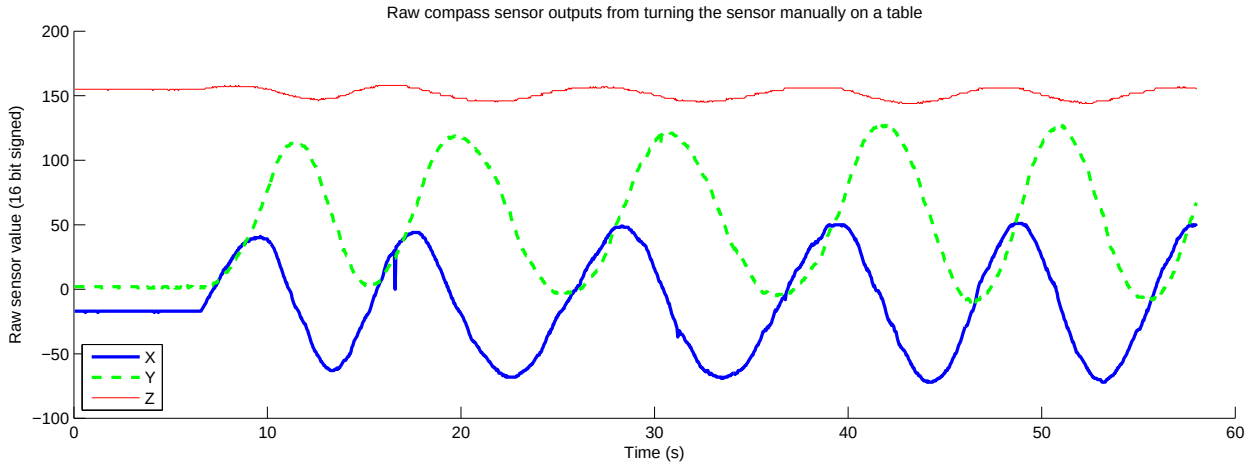


Figure 2.7: An example of the output from a compass sensor being turned manually on a table.

the middle of the table. Eventually it is moved away from the center, putting it closer to one of the metal bars. This can be seen in the figure as a small gradual increase of the output signal's amplitude.

Metal Influence

To be able to calculate a reliable heading value, the offsets and amplitude differences need to be calibrated out. This is something we discuss further in section 4.2 when we describe the heading calculation in detail. Here we only show what these effects do to the raw measurements of the sensor. Note that transient effects, like moving towards a metal object, cannot be calibrated for. In section 1.1 we state the assumption that no external factors influence the system and therefore we do not give a solution. Instead, we use a driving arena for the cars without nearby metal surfaces and therefore the compass sensor should have no significant problems.

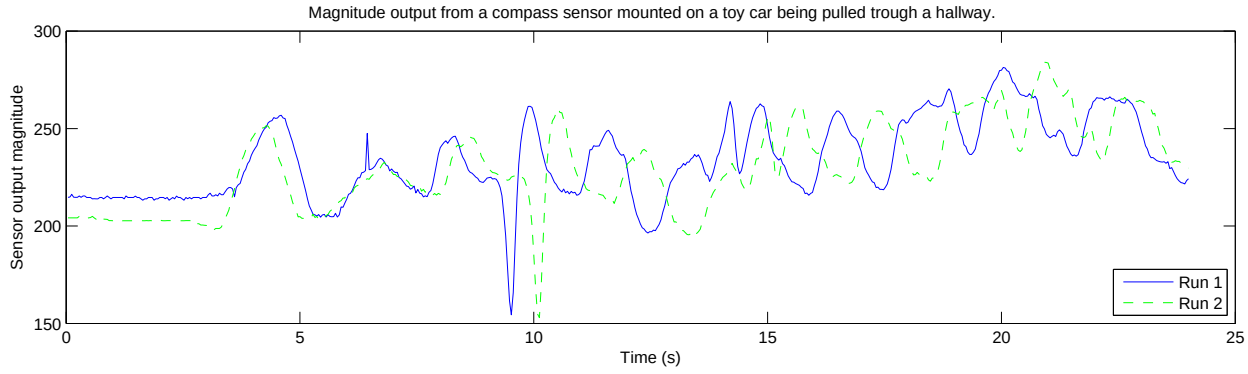


Figure 2.8: The effect of large metal objects on the compass output value.

To give an indication of how badly the compass is interfered by nearby metal objects, we show the results of a couple of tests performed in our faculty building in Figure 2.8. The building has long straight hallways lined with large concrete pillars. Each pillar contains a significant amount of steel as reinforcement. In such a hallway, we perform two successive tests in which a node with a compass sensor is mounted on a toy car. The car is pulled manually over a distance of about 25 meters along an approximately straight path in the middle of the hallway. Figure 2.8 shows the magnitude calculated from the raw measurement values of all three axes. If the sensor is moved through a constant magnetic field with a constant direction, one would expect to read a mostly constant magnitude from the sensor[9].

[9]If the sensor were calibrated during this test, the output would be constant regardless of any changes in the heading or tilt of the sensor.

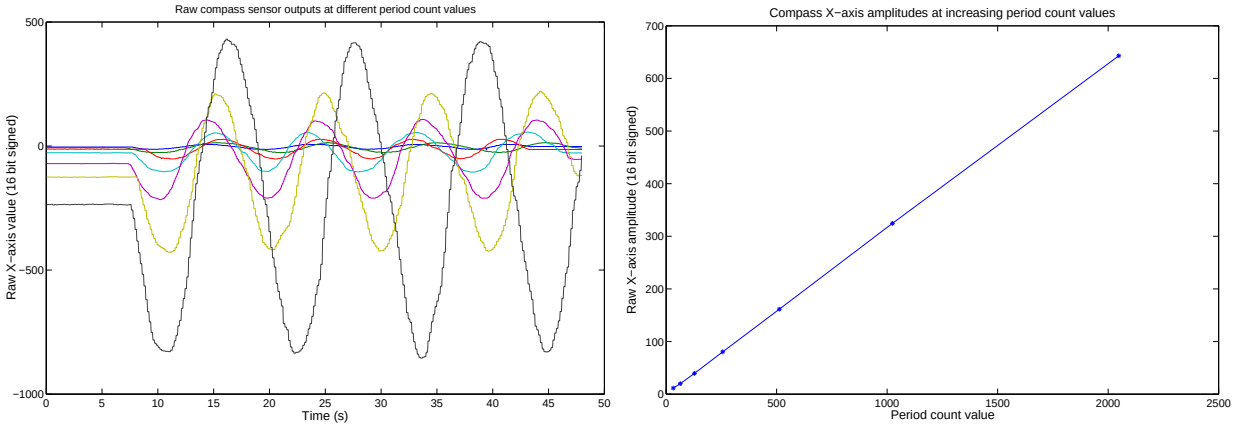
Clearly, during these tests, the earth's magnetic field is significantly influenced, or rather distorted, by nearby metal objects. The amplitude of the signal caused by interference is in the same order of magnitude as the results we present in Figure 2.7, making indoor use very problematic. Interestingly, except for a difference in movement speed of the sensor in each test, the two separate runs give very similar results, indicating that the influence of nearby metal is constant as long as these objects are not moved.

Magnetic Resolution

As described earlier in section 2.3.3, the Micromag 3 compass sensor can be configured to use different *period count* values. This configuration directly influences the sensitivity of the sensor to magnetic fields. The sensor can be configured to use 8 discrete period count values as listed in Table 2.1.

To show the effect of this period count configuration parameter, we perform a series of tests at increasing values. The tests include only the first seven possible period count values of Table 2.1. Like before, the sensor is placed on the table and turned manually a few times. For the tests with a period count of up to 256, the sensor could be sampled at the previously determined 25 Hz. At larger values, the sensor is sampled at appropriate lower frequencies, but the results are up-sampled to match the other data.

Figure 2.9 depicts the results of these tests. In Figure 2.9(a), the x axis output signals of the individual tests are presented. Each test involves slowly turning the sensor 4 times. Because the tests are performed manually, the sinusoidal signals do not align. This does not matter, as we are only interested in the amplitude of the signals. Figure 2.9(b) shows what that amplitude was for each configured period count. The markers indicate the available configurations.



(a) Tests at various period count values

(b) The effect of period count versus signal amplitude

Figure 2.9: An assessment of the effect of the compass measurement period on the amplitude of the output signal.

In Section 2.3.3, we correlated the amplitude of the output signal of the compass sensor with the attainable compass resolution. Looking at Figure 2.9(b), we see that the configured measurement period has an approximately linear influence on the amplitude of the output signal. Reminding what the period count actually is, this linear behavior is to be expected. In Section 2.3.3, we describe that the output of the actual magnetic sensor is a frequency that depends on the strength of the applied magnetic field. Regardless of the period count, this frequency remains constant for a specific field strength. The period count only determines the number of output signal periods that are required to stop the measurement timer. With constant frequency, the influence of the period count on the measurement timer's final value is linear and this value is what is used as sensor output.

In Section 4.2, we show how exactly the magnetometer resolution of an individual axis of the compass sensor influences the compass' angular resolution. That is where we explain the calculations involved to obtain the current heading from a compass sensor.

Tilt Effects

To assess the effect that the inclination of the compass sensor has on its outputs, we perform a couple of tests with two separate compass sensors. We first show how well the measurements from two distinct sensors correlate when the sensors are subjected to identical orientation changes.

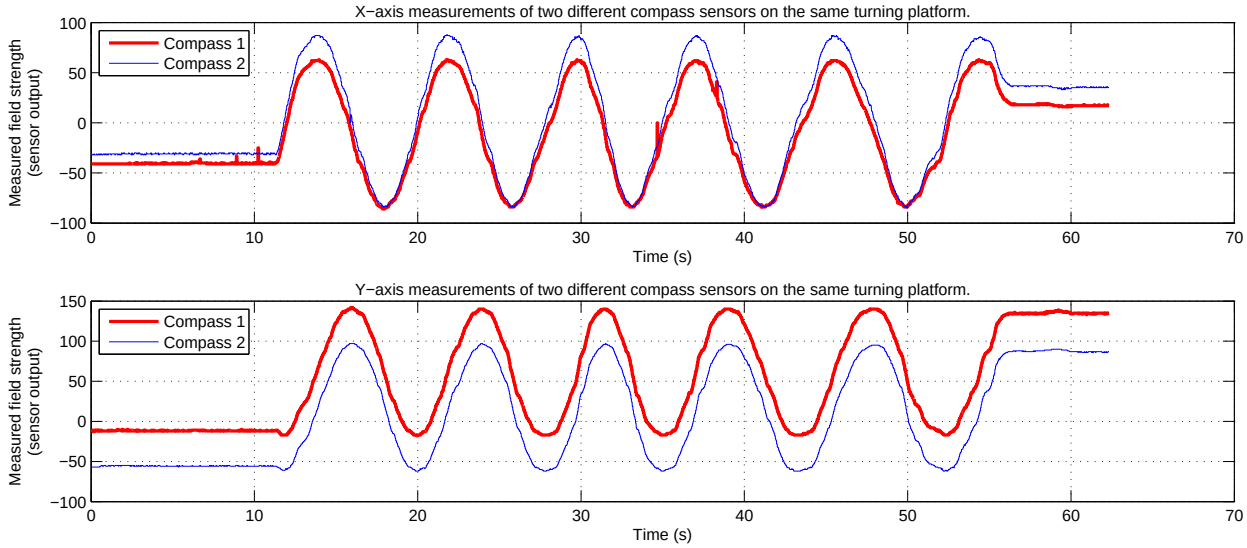


Figure 2.10: A comparison of the measurements of two compass sensors placed level on the same turning platform.

Figure 2.10 shows the x and y axis measurement results from the two sensors. The test involves manually turning a small platform with the two sensors placed on it on a table. Both sensors have approximately the same orientation relative to the platform.

As shown, the measurements from the two sensors are sinusoidal signals that correlate well. However, although the frequency and phase of both signals is very close to identical, the signals do not overlap as they do not have the same offset and amplitude. This is caused by differences in the sensors themselves and the external influences these were subjected to. The sensor nodes with the sensors are placed very close together on the platform. The batteries and any other metal in one node cause an influence on the compass sensor of the other node. This influence will differ, because, for example, on one node the other node is on the positive side of its axis and for the other node this will be the other way around.

In Section 4.2 we show that this difference can be compensated for by calibration. In effect, this makes the offset of the signals zero and it makes the amplitude uniform over all axes of the sensor. This calibration procedure aims to only let the phase of the axes signals have a significant influence on the calculated heading. Therefore, the compass headings calculated from the two compasses in Figure 2.10 would be very close to identical.

Now we evaluate what happens if one of the two sensors is tilted around its x axis by approximately 45° . As shown in Figure 2.11, the measurements still correlate well in terms of frequency and phase. However, on the y axis, the offset and amplitude of the signal changed significantly. Consequently, if the compass sensors are calibrated using measurements collected with the sensors mounted level, this calibration becomes useless as soon as the inclination of the sensors changes.

A common solution to this problem is to actually measure the inclination of the sensor and use that information to calculate what the compass measurements would have been if the compass were level. With this technique, the compass calibration remains valid, regardless of the compass inclination. In Section 4.2, we describe exactly how this works.

In Section 4.2 we also provide a description of the factors that influence the achievable angular resolution with our compass sensor. One of the requirements for a reliable and high sensor resolution is a low level of

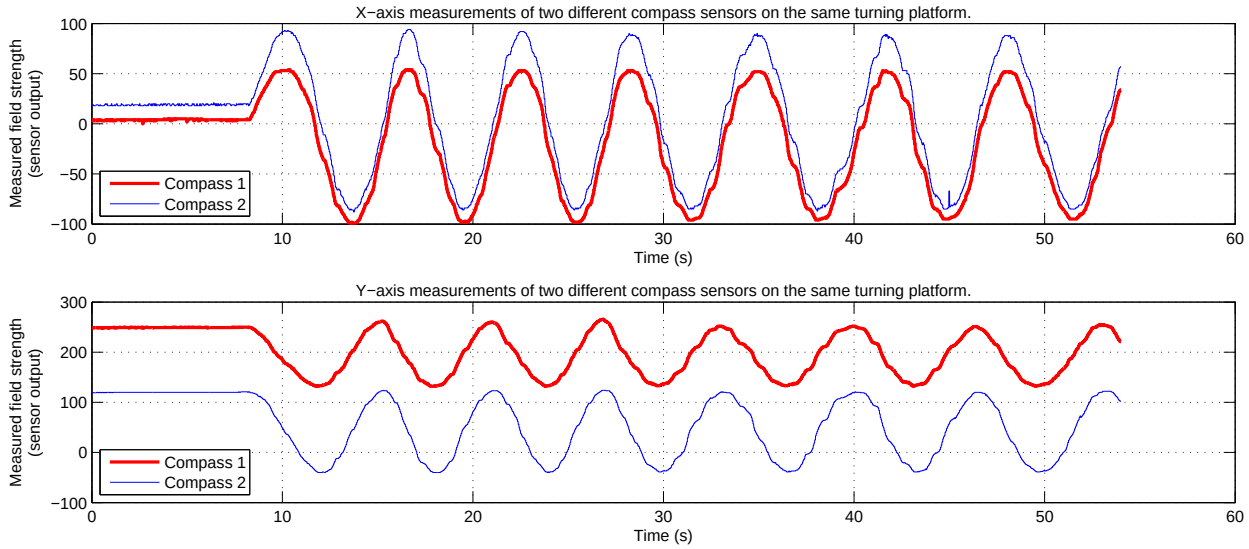


Figure 2.11: A comparison of the measurements of two compass sensors on the same turning platform of which the first is placed level and the second has an inclination of 45° (pivoting around the x axis).

sensor hysteresis, meaning that the sensors readings should be deterministic at a given field strength. The rough manual experiment of Figure 2.10 shows that the sensor approximately returns to the same reading at the same angle in continuous circular movements. This suggests that the hysteresis of this sensor is low enough to provide a reliable reading. This of course is not necessarily true if the sensor is being moved around on a vehicle were it may be subject to changing metal influence.

2.4.2 Accelerometer Evaluation

In this section we show examples of readings from the accelerometer attached to one of our sensor nodes. Being a three-dimensional sensor, the accelerometer yields three measurements per sample. Figure 2.12 shows the results of a short acceleration experiment. The sensor node with the accelerometer is suspended below a table with a short length of string. Its z axis approximately points upwards, while its x and y axes are more or less level. After about two seconds, the sensor is pushed manually into a pendulum motion in which it oscillates in both the x and y direction, making a circular motion.

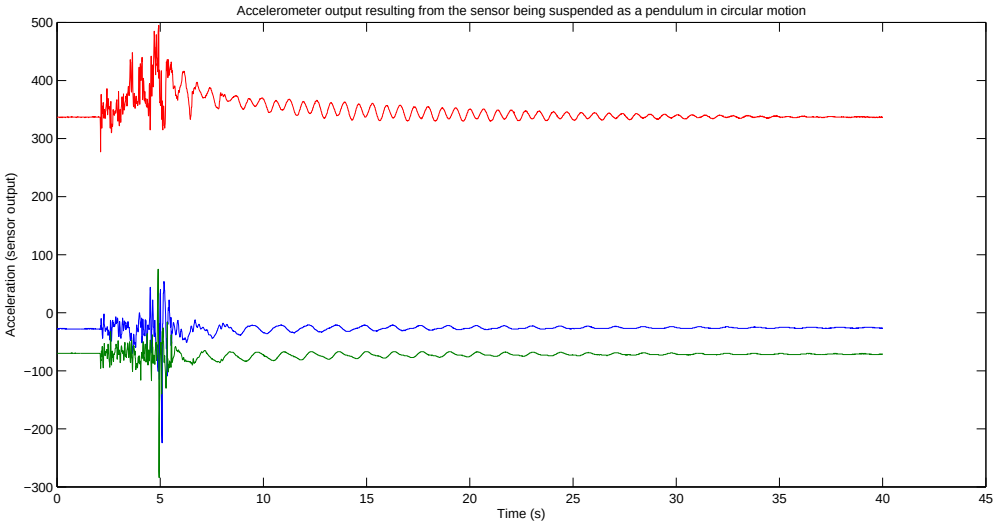


Figure 2.12: An example of output signals from the accelerometer.

The manual push causes a significant amount of high-frequency noise on the acceleration signals. After 6 seconds the sensor is released to swing freely. The swinging motion continues for more than 40 seconds until it diminishes completely. As the figure shows, the sensor does not produce much noticeable noise on its own. If the sensor is put somewhere on a perfectly stable surface that is free from any vibration, the sensor only presents single-bit errors at its output.

The gravity vector is also visible in Figure 2.12. Because the z axis of the sensor points up, the gravity will pull the sensor's internal proof mass down, giving the accelerometer the sense that it is accelerating upwards. The sensor is configured for a measurement range of -6 to 6 g, and its output value is represented as a signed 12 bit value. This means that a positive acceleration of 1 g will be output as a value of about 340, which matches the figure. Note that the sensor is never perfectly level during this experiment, because the signals from the x and y axes are always offset from zero, meaning that part of the gravity vector is measured on those axes as well.

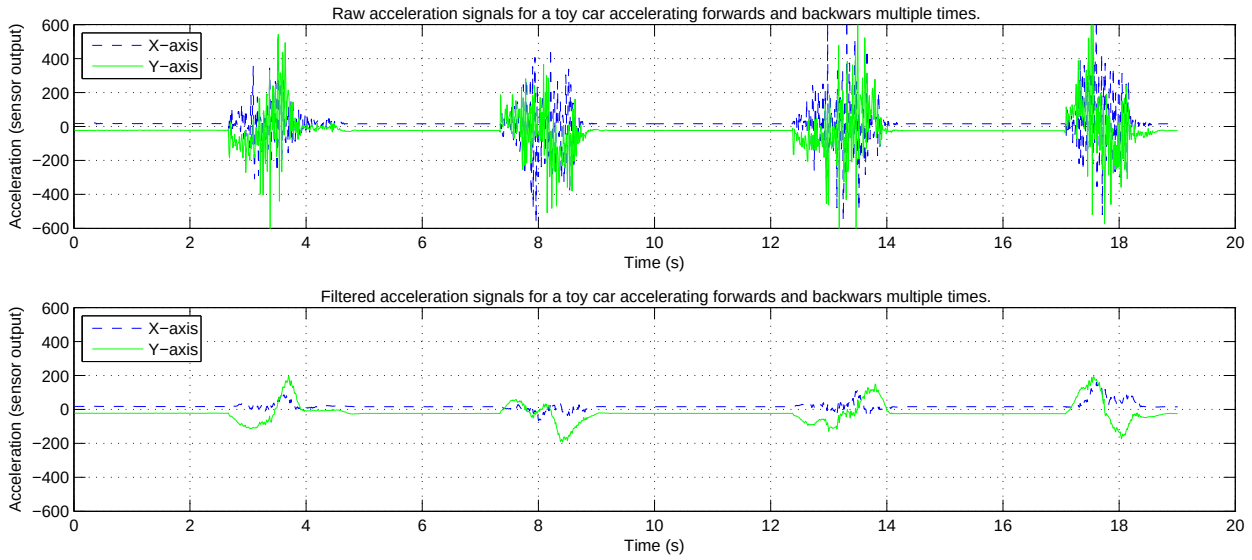


Figure 2.13: Output signals from the accelerometer mounted on a toy car that accelerates forwards and backwards multiple times.

To evaluate how the sensor behaves for our purposes, we mount it on top of one of our toy cars. The car makes four approximately linear movements, alternating forwards and backwards. The top graph of Figure 2.13 shows the raw sensor outputs for the x and y axes. The data is almost unintelligible due to the significant amount of noise in the signals. This noise disappears as soon as the car stops and is caused by the vibrations that the car incurs during its movement. In Section 4.1, show how we use this effect to our advantage to detect whether the car is moving.

To provide a better picture of the car's actual acceleration, the sensor data is filtered using a 20 tap running average filter. The filtered result is shown in the lower graph of Figure 2.13. The y axis points backwards during this test. The y axis clearly shows the acceleration and deceleration of each movement. Interestingly, the x axis also shows significant acceleration. This is either caused by the fact that the sensor is not mounted properly in line with the vehicle or by the possibility that the sampling frequency of 100 Hz is too low causing aliasing effects. We show the effect of the sampling frequency on the velocity integration in Section 4.3.2.

The two plots of Figure 2.13 clearly show that the noise magnitude exceeds the magnitude of the actual movement signal. Because the noise results from vibrations, it is harmonic in nature, meaning that its average value will be very close to zero. This means that, when the noise is accumulated during integration, its net effect is insignificant. That is why we do not need to filter the accelerometer signal. This does, however, require that the sample frequency is sufficiently high to avoid aliasing effects as we show in Chapter 4.

2.5 Summary

This chapter provides an overview of the hardware used for this work. The system consists of the following hardware components:

- Vehicles: Two different toy cars are used as vehicles. Both are equipped with wireless sensor nodes and sensors. One car is left unmodified and designated the *leader*. The other is the *follower* and modified to facilitate control by the attached node using our own motor controller (refer to Appendix A). The leader car can drive much faster than the follower and it is more agile in terms of steering.
- Wireless Sensor Nodes: We use the Ambient μNode 2.0 platform, running on the low-power MSP430 microcontroller from Texas Instruments. The nodes run the AmbientRT real-time multitasking operating system. Apart from the two nodes mounted on the vehicles, there is a third gateway node connected to an external computer to log the data exchanged by leader and follower for later evaluation.
- Sensors: To measure the movements of the vehicles, we mount an accelerometer and a compass sensor on each vehicle. We use the LIS3LV02DQ three-axial accelerometer from ST Microelectronics and the Micromag 3 three-axial magnetometer from the PNI Corporation as compass sensor. The sensors are connected to the sensor node using a software SPI bus. The accelerometer can measure the three axes simultaneously in a single command, but the compass sensor can only sample the axes individually.

We pay special attention to the sensors in this chapter. We outline the following important notions regarding the used accelerometer and compass sensor:

- Inclination influence: Both compass and accelerometer are sensitive to changes in their inclination. A non-level inclination can be compensated for using calibration, but when the vehicles move, this is not possible. Dynamic inclination compensation could be implemented using a gyroscopic sensor, which we currently don't use due to its poor availability and the involved calculations required to perform the compensations. Currently, we assume that the vehicles move over a relatively level surface, meaning that they will not encounter significant inclination changes.
- Calibration: The compass sensor is calibrated only manually before performing a series of tests. The accelerometer is calibrated for inclination changes automatically each time the vehicle stands still. The calibration of the accelerometer and the motion detection technique are described in Section 4.1.2.
- Compass influence: Because a compass sensor is a very sensitive magnetometer, it is highly susceptible to stray magnetic fields and distortions in the earth's magnetic field. The static external influences like metal distortions by the vehicle itself are compensated for using calibration. Dynamic effects like moving past metal surfaces and magnetic fields emanating from the vehicle's actuators are much more difficult to handle. Regarding the actuators, it is often easier to keep the sensor away from the source of these influences or to shield it using ferrous metal.
- Resolution and measurement range: Both the compass and the accelerometer have configurable measurement ranges with different resolutions. The accelerometer can measure acceleration either in the range ± 2 or ± 6 g. During experiments with the vehicles it is observed that the acceleration can easily exceed the 2 g boundary due to vibration noise, therefore a range of ± 6 g is configured. The compass sensor can be configured a various so-called *period count* values. This configuration dictates the measurement range and the resolution of the compass' individual magnetometer sensors. However, a higher resolution decreases the attainable sampling rate. For measuring the earth's magnetic field any period count value is adequate, because the field strength will then never exceed the configured measurement range. In Section 4.2.1 we outline the compass sensor resolution and sampling frequency requirements and choose an appropriate period count value.
- Sample frequency: The accelerometer can only be configured at a series of fixed sampling frequencies, because it has its own sampling timer and thus needs to external trigger to start individual samples. The configurable frequencies range from 40 to 2560 Hz. The compass sensor is slower with a sampling frequency ranging from 6 to 660 Hz. However, unlike the accelerometer, the attainable sampling frequency depends directly on the configured resolution. In Section 4.2.1 we show that attaining sufficient compass resolution severely limits the sample frequency. In Chapter 3 we show how sampling the accelerometer and the compass sensor at different frequencies is interleaved. This is primarily

facilitated by the fact that the compass sensor does not need to occupy the shared SPI bus during a measurement.

Before the sensors are incorporated in the definitive implementation, we describe exploratory experiments with the sensors using our prototype platform. This platform uses a simple TDMA MAC protocol to transmit sensor data from up to three nodes to a gateway node for recording. The accelerometers are sampled at 100 Hz and the compass sensors are sampled at 25 Hz. This evaluation yields the following important results:

- The sensors show mostly insignificant inherent noise: Both accelerometer and compass sensor rarely show significant noise when not subjected to movements. The compass sensor infrequently shows large spikes, but the cause of these is not ascertained. The accelerometer shows only single bit errors when it is not moving or vibrating.
- The compass is useless indoors: Indoor testing shows that the compass sensor is highly sensitive to steel contained in our building's pillars. The measurements from the compass sensor are useless there due to the large deviations from what is expected for the performed experiments.
- The compass shows low hysteresis: It is interesting to note that the compass sensor is very consistent with its measurements: moving the sensor through the building along the same path produces a very similar output plot. This is also indicated by the experiments that involved turning the sensor on a table: the amplitude and offset of the resulting signal remains very constant when the sensor is kept on the same position, indicating that the hysteresis is low. The hysteresis is not measured directly however.
- The accelerometer measures much noise on a driving toy car: When the accelerometer moves freely, it incurs very low noise levels. However, attaching the accelerometer to a toy car results in high-frequency noise that exceeds the magnitude of the actual movement. This noise consists of the vibrations caused by the car rolling over the floor surface. In Section 4.1.2 we show how this noise is used to detect whether the car is moving. Due to the harmonic nature of the vibration noise, it can safely be integrated using the technique we describe in Section 4.1.1, as long as the sample frequency is high enough. This avoids the need to filter the accelerometer signal.

Chapter 3

Driver Software

The system's software needs to interact with the hardware components described earlier in Chapter 2. In this chapter we describe these low-level software-concerns. The driver software of the leader and follower is similar in many respects. The only difference is the motor controller included in the follower. First, in Section 3.1, we describe how data is acquired using the sensors connected to the nodes. Subsequently, we outline the interaction between the follower's node and motor controller in Section 3.2.

3.1 Sensor Sampling

The sensors chosen in Section 2.3 differ significantly in their maximum sampling rates. Usually, choosing higher frequencies yields better measurement results. However, the system is bound to consume much more power when sampling at high frequencies and it might prove to be very hard to schedule such high-frequency sampling operations on the relatively slow sensor node processor. Consequently, it is best to choose the lowest frequency that meets the requirements it terms of accuracy and response time. Unfortunately, these requirements are also not identical for the compass and the acceleration sensor.

3.1.1 Sampling Requirements

In Section 2.3.3 of the previous chapter, we explain the effects of the period count configuration parameter of the compass sensor. Depending on the configured period count, an axis of the Micromag 3 compass sensor we use can be sampled at frequencies ranging from 2 kHz to 16 Hz. In that section we also note that the period count value influences the resolution of the sensor, because it directly controls the sensor's sensitivity to magnetic fields.

Because the measurement resolution depends on the period count value, the very high period count values that make high sample rates possible likely do not provide enough accuracy to achieve the 1 degree resolution that we require in Section 1.3. In section 4.2, we show that a period count value of 512 provides sufficient sensor accuracy. This yields a theoretical maximum sampling frequency of 133 Hz. However, as we discuss in Section 2.3.3, only one axis of the sensor can be read at a time, yielding a maximum sampling frequency of about 44 Hz for reading all three axes. In Section 6.2, we choose a control frequency of 16 Hz, so a matching sampling frequency for the compass sensor should be adequate to provide up-to-date heading information for each controller execution.

In contrast, for accurate velocity integration, it is imperative to sample the acceleration at a rate higher than the Nyquist frequency of any significant noise in the input data produced by the driving car. In Section 2.4.2 we show that this noise is very significant when moving. Not following the Nyquist rule causes aliasing effects that map frequency components of the noise to seemingly arbitrary positions in the spectrum. This cannot be filtered in the digital domain and thus it can have a severe detrimental effect on the accuracy of the velocity integration. The actual frequency spectrum of the acceleration data resulting from the driving car is hard to determine, since this requires sampling the sensor at a very high frequency to obtain reliable results.

In stead, we experimentally determine an appropriate sampling frequency for the velocity integration by means of experimentation in Section 4.3.2. There we find a sampling frequency of 160 Hz to yield satisfying results. This is well below the maximum sample rate of 2560 Hz that the sensor can handle for all three axes simultaneously.

The compass sensor and the accelerometer differ significantly in the required sample rate. The sensor sampling software must be able to handle this difference gracefully. Furthermore, it must be able to handle the individual required sampling frequencies. The relatively slow sampling frequency of the compass sensor should pose no problem, but the high frequency of the accelerometer may prove to be hard to schedule on the sensor node's processor.

3.1.2 Sampling Strategy

The chosen sampling frequency of the accelerometer is ten times faster than the frequency of the compass sensor and the controller. It is logical to synchronize the execution of the controller of the follower with the sensors to directly process the incoming data. This way, ten accelerometer measurements and one compass measurement a performed for a single controller execution. With an accelerometer frequency of 160 Hz, the compass and controller frequencies are then fixed at 16 Hz. Refer to Section 6.2 for more information on the relation of these frequencies.

As we explain in Section 2.3.3, the compass sensor we use can do its measurements in the background without occupying the SPI bus. This means that the compass sensor can be setup to start a measurement, in the mean time the accelerometer can be sampled ten times, and at the end the compass measurement is read from the sensor. However, this simple version of the solution cannot be implemented, because the three axes of the compass sensor must be read sequentially and thus a measurement cannot be started for all three compass axes at one time. This means that, in the period in which the accelerometer is sampled ten times, the compass sensor is setup to measure three times and each time it is ready, the measurement is read from the sensor.

Figure 3.1 presents this sampling schedule schematically. Note that the duration of the sampling tasks is not to scale; only the intervals are of significance in this figure. The slowest sampling frequency is that of the compass sensor and we call the associated period the *main sampling period*. The accelerometer is sampled ten times faster; each 6.25 ms at 160 Hz. After the fist accelerometer sample, the compass sensor is setup to start measuring on the x axis. With the period count configured as 512, this axis measurement takes 7.5 ms at worst. This means that the compass measurement finishes some time after the second accelerometer sample, as indicated in the figure by the arrow labeled 'compass x axis'.

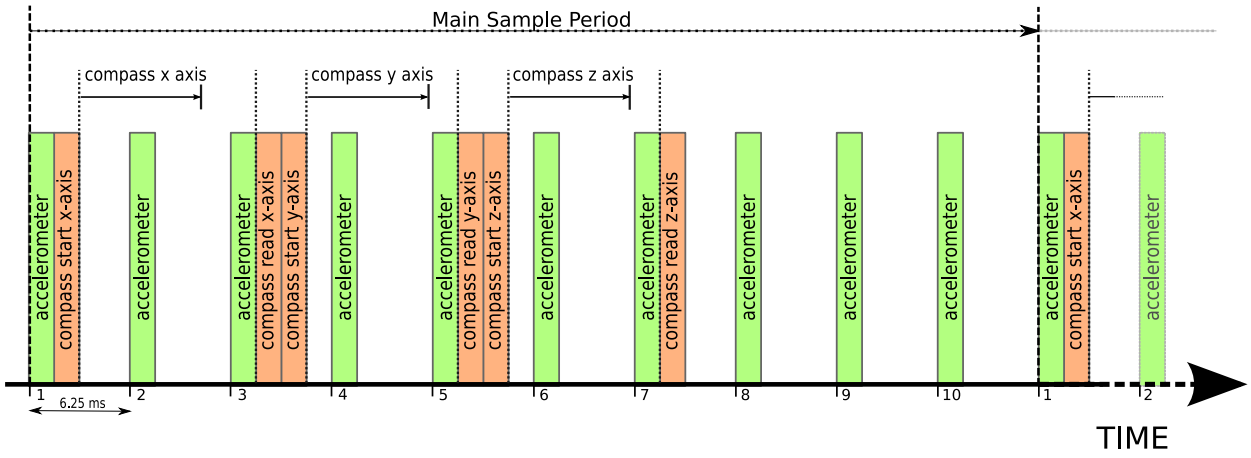


Figure 3.1: The strategy to sample the accelerometer and the compass sensor simultaneously.

When a measurement is complete, the compass sensor activates its ready signal. Although it is common to connect this signal to an interrupt input of the node, we decide to poll the compass' ready line after each subsequent accelerometer sample. This is why the compass measurement is not read until after the third

accelerometer sample. This procedure omits using the scarce interrupt lines of the node. It also makes the schedule more deterministic. This avoids interference with the accelerometer sampling task, which could cause the CPU to miss reading some of accelerometer's samples[1].

Directly after reading the x axis result, a subsequently triggered task resets the compass sensor and configures it to measure the next axis. This procedure continues until all compass axes are sampled once. Then, three more accelerometer samples remain until 10 accelerometer samples are collected and the whole sample process restarts.

This is still a simplistic view of the sensor sampling schedule. As described earlier in Section 1.6 other software tasks are active on the nodes and these need to cooperate gracefully with this process. We present the full set of tasks and their scheduling on the node in Section 8.2.

Also, this discussion assumes that we sample all three axes of the compass sensor. However, this is only useful when tilt compensation is performed on the compass sensor readings. In Section 4.2.3 we show that we cannot do that without reliable continuous measurements of the compass' inclination. We currently cannot perform inclination measurements for the sensor board and therefore only two axes of the compass sensor are sampled for efficiency.

3.2 Motor Control

In this section we briefly describe which concerns are handled by the follower's motor controller and which commands are communicated to it. Refer to Appendix A for a more elaborate description of the motor controller.

Currently, we use an I^2C interface to establish communication between the follower node and the motor controller. This choice is not very practical, considering that the specific MSP430 micro-controller for the the motor-controller has no hardware support for slave I^2C. Thus, the I^2C bus is fully implemented in software, making it slow and less power-efficient. SPI is a better choice, since the MSP430 on the slave controller has hardware support for slave SPI, making the reception of individual bytes just a matter of handling an interrupt for each byte. Although SPI is a better choice, we do not give priority to change a controller that works without problems.

The following movement controls are handled by the motor controller:

- *Throttle*, controls forward or backward acceleration
- *Steering*, controls the angle of the front wheels
- *Brake*, short-circuits the rear drive motor to induce inductive friction on the rear wheels

The toy car employed as follower has two motors: one for driving and one for steering[2]. Both motors can be driven with varying intensity. For the drive motor this controls the throttle of the car and for the steering this controls the angle of the car's front wheels. The throttle and brake controls concern the same motor and are mutually exclusive. The command structure for the motor controller is divided between the two motors but the available commands for each motor is identical, except for the availability of a brake command on the drive motor:

- *Disable*, turns-off the motor; the axle can turn freely
- *Drive Forward/Steer Right*, turns the motor in one direction
- *Drive Backward/Steer Left*, turns the motor in the other direction
- *Brake*, the motor's axle incurs inductive friction (drive motor only)

The intensity at which the motor is driven is controlled by a separate parameter. Setting the intensity has no effect for the *Disable* and *Brake* commands. The command and intensity for each motor is the only

[1] There is no way to give a task an absolute priority with the given AmbientRT operating system. Which task is activated next depends entirely on the deadlines and resource constraints specified for each task.

[2] Actually, our particular toy car model has a few more actuators to make it jump. Obviously, this is not useful for this research and those actuators are left unconnected.

information passed from the node to the motor controller through the I^2C bus. No feedback is produced other than the ACK bits as specified for the I^2C bus protocol [38]. Much like other I^2C devices, the motor controller specifies a set of *registers* in which the master, i.e. the follower node, can write control parameters. In this case, four single-byte registers are defined for the command and intensity of both motors.

3.3 Summary

In this chapter we describe the low-level interaction of the software running on the nodes with the connected hardware: sensors and actuators. For the sensors we describe the sampling requirements and the resulting sampling strategy. On the follower, the node controls motors for steering and accelerating. The follower node does not drive the motors directly, but rather defers the necessary actions to the follower's motor controller.

We list the following sampling requirements for the two sensors:

- The compass sensor is sampled at a rate of 16 Hz: The maximum attainable frequency is 44 Hz at the required compass resolution, but 16 Hz matches the control frequency, which means that each controller execution has fresh heading information available.
- The accelerometer is sampled at a rate of 160 Hz: The experiments of Section 4.3.2 indicate that 160 Hz is adequate and is not improved much by choosing a higher frequency. This frequency matches one of the native frequencies the accelerometer provides, meaning that the sensor does not sample more frequently than the CPU reads the results.

A sampling implementation that needs to achieve these sampling frequencies faces the following challenges:

1. The required sampling frequencies differ significantly: This means that while the compass is measuring, multiple accelerometer samples need to be collected. This is further complicated by the fact that the compass sensor's axes cannot be sampled simultaneously.
2. The sampling frequency for the accelerometer is very high for a resource-limited sensor node: Scheduling this on the CPU with the rest of the system's tasks is difficult.

To meet the sampling requirements we devise a sampling strategy that is dictated by the accelerometer sample instances: the task that reads the accelerometer at 160 Hz is also responsible for triggering compass measurements. Both sensors share the same SPI bus, but luckily the compass sensor does not need to occupy the bus while it is measuring. The compass is sampled ten times less frequent than the accelerometer, therefore a compass sensor measurement is completed every ten accelerometer samples. During this period of ten samples, the measurements for individual compass axes are started successively. Starting and reading the compass is always performed synchronous to the acceleration samples, avoiding conflicts. Section 8.2.2 shows how this sampling strategy fits into the task scheduling of the full node software implementation.

To drive and steer the follower vehicle, the follower node connects to our custom motor controller through a software I^2C bus. The motor controller is a separate circuit board with another MSP430 microcontroller providing the necessary hardware to control the steering and drive motors of the vehicle. The controller can drive the motors in either direction or turn them of completely. For the steering motor, this implements left-right steering and for the drive motor this provides forward-backward driving. Additionally, the drive motor can be short-circuited to obtain an inductive brake. This works by means of the increased motor friction caused by the self-inductance in the short-circuited motor. The intensity at which the motors a driven is controlled by means of PWM control (refer to Appendix A.2.1).

The motor controller removes the burden of directly controlling the motors from the node microcontroller. The only thing the driver software on the node needs to do is write the control signals over the I^2C serial bus to the motor controller.

Chapter 4

Data Processing

After the raw sensor data is acquired, it needs to be processed, yielding the required movement information for the follower's controller. This movement information consists of the measured heading and estimated velocity of the node. Keep in mind that this data is used locally on the follower to control its movements, but on the leader it is transmitted directly to the follower without further consideration. In this chapter we describe how the raw sensor data is processed to obtain the required movement information.

The current velocity of a node can be estimated by integrating the acceleration samples acquired using the accelerometer. As our system does not provide inertial orientation data for the acceleration sensor, we cannot use the straightforward integration techniques commonly used with inertial navigation [49]. In stead we have devised a very simple integration technique that incorporates the vehicle's movement properties to obtain a valid velocity estimate. We describe this velocity estimation technique in Section 4.1.

The current heading is derived from the compass sensor's raw output values using an arctangent function. This is a well-documented process. We provide a survey of the involved calculations and problems in Section 4.2.

At the end of this chapter, in Section 4.3, we show how the data processing techniques described in this chapter work on practical data. For the heading calculation we show the results of letting two nodes make the same movements. The velocity integration is tested directly on a toy car that makes linear movements. Finally, we show the simultaneous results of the two sensors being mounted on a car that drives continuous circles.

4.1 Velocity Integration

To obtain a velocity estimate from a stream of acceleration data, integration must be applied because acceleration a is the derivative of velocity v. The velocity relates to the acceleration in the interval $[T_1 \dots T_2]$ as follows:

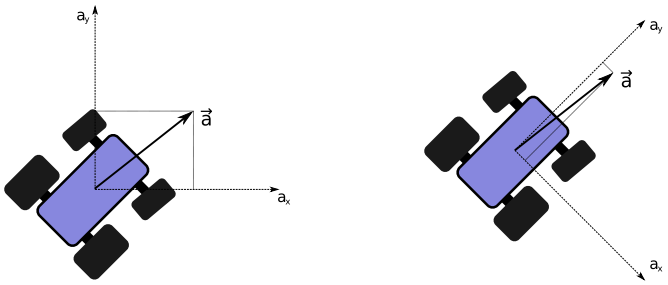
$$v(T_2) - v(T_1) = \int_{T_1}^{T_2} a(t)dt \tag{4.1}$$

Assuming the car starts with $v = 0$, the current velocity can be estimated by continuously adding incoming acceleration measurements to the previous velocity result. However, with Equation 4.1 the velocity integration works only if the vehicle is moving along a straight line. If the vehicle also steers in other directions, the integration problem becomes more complex. In that case, the acceleration vector is no longer necessarily in line with the current velocity vector.

4.1.1 Vector Integration

To integrate velocity and displacement from acceleration data in any direction, it is common to map the acceleration data into a global reference frame (e.g. the inertial frame or the Earth frame). This is the central idea of inertial navigation [49]. A reference frame is in this context a coordinate system used for navigation. Mapping the acceleration data into a global reference frame means that the acceleration vector is transformed to be represented in a coordinate system with a constant orientation relative to the world. The local reference frame in which the accelerometer directly measures is commonly called the *body reference frame*. Unlike a global reference frame, the body reference frame always has the same orientation as the sensor with the axes of the reference frame aligned with the measurement axes of the sensor.

Figure 4.1 shows a two-dimensional comparison of a global reference frame and the body reference frame of a wheeled vehicle. Figure 4.1(a) uses the picture itself as global reference frame, meaning that the Y axis component of the acceleration vector $\vec{a}$ always points upwards and the X-axis component always points to the right. The body reference frame of Figure 4.1(b) always has the same orientation as the vehicle. Mapping the acceleration vector from the body to the global reference frame results in an acceleration vector that follows the orientation of the vehicle relative to the world. The main advantage of this approach is that the resulting acceleration vector can directly be integrated into a velocity vector and subsequently into a position estimate within the global reference frame. This position value is directly useful as the global position necessary for navigation applications.



(a) Global reference frame (b) Body reference frame

Figure 4.1: Two-dimensional reference frame comparison.

Unfortunately, inertial navigation requires the orientation of the accelerometer to be known at all times. If the vehicle is assumed to move only in a single plane, a compass sensor could provide this information. Otherwise, an additional gyroscope is necessary to keep track of the roll and pitch angles. In Section 2.3, we explain that we currently avoid using a gyroscope. In either case, the use of additional error-prone sensor measurement in the integration process will likely make the velocity result even more unreliable.

Also, because we only aim to implement common movement rather than common navigation, we are not going to infer the position of the vehicles. We only want to obtain the magnitude of the velocity of our cars, meaning that the current direction of the velocity vector is not used in the integration process. For the integration of acceleration measurements into a velocity estimate, we are only interested in the forward or backward velocity of our toy cars, since - for this particular type of vehicle - these are the only directions the velocity can have in normal conditions: the cars cannot drive sideways[1]. We make the assumption that the vehicles drive in on a level plane (refer to section 1.1) and therefore we only consider two-dimensional navigation. The direction of the velocity vector in this level plane is assumed to be equal to the vehicle's heading, which is separately measured using the compass, involving no integration[2].

Concretely, this means that we assume that the car's velocity vector always has the same direction in the body reference frame of the sensor. This is mostly a valid assumption, as our vehicles cannot change their heading on their own without velocity and, as the vehicle steers, the velocity vector follows its new direction.

[1] Admittedly, with excessive steering at high velocity and bad surface traction a vehicle can make a spin with wheels slipping over the surface, throwing it in a sideways movement. For this work, we do however not consider such rude driving. And all other movements that are not forward or backward can only be caused by external factors.

[2] At the end of this chapter in Section 4.3.2 we show that it is possible to extract an estimate of the velocity's direction, i.e. the vehicle's heading, from the integration process.

When an accelerometer is mounted on the vehicle such that one of the axes points in the exact direction of the forward velocity, it will measure the pure forward and backward acceleration on that axis. When the car is not steering, this would be enough information to perform the linear integration as presented at the beginning of this chapter as Equation 4.1. With measurements form the accelerometer axis that is perpendicular to the vehicle's movement, the integration can be extended to account for the vehicle's steering.

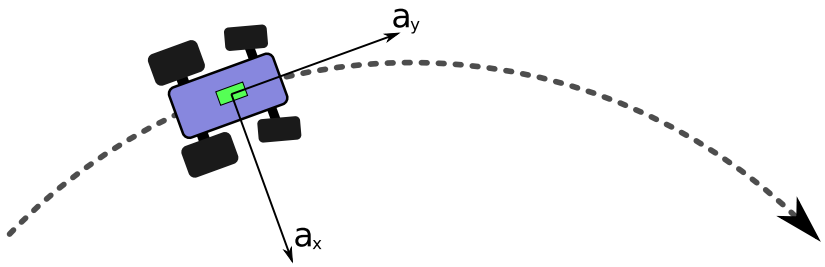


Figure 4.2: Orientation of the accelerometer on the vehicle

Figure 4.2 shows this in a schematic representation of the vehicle driving in a curve. The acceleration sensor is mounted in the middle of the vehicle with its a_y axis pointing forward and its a_x axis pointing to the right. Because the sensor is mounted rigidly on the vehicle, it will always measure these acceleration vector components relative to the car's current orientation. Approximately, this means that the a_y axis measures the tangential component of the acceleration vector along the vehicle's curved path, whereas the a_y axis measures the acceleration component perpendicular to the curved path. Remind that, looking at Figure 4.2, the velocity vector, like the a_y acceleration vector component, is assumed to be tangential to the vehicles curved path at all times, i.e. it always points in the direction the vehicle is facing.

Figure 4.3 schematically shows the vectors involved in the integration calculation. Again, the sensor node is shown as the small rectangle in the middle of the vehicle. The vector v is the current velocity vector and it is assumed to always point into the same direction as the vehicle itself. Only the magnitude of the current velocity vector is considered in this calculation and not the direction of the velocity vector. Therefore, the resulting vector v' is assumed to point forward again in the next integration step, i.e. the vehicle follows the change in direction of the velocity vector. Of course, it is necessary to distinguish between driving forward and backward, so the current velocity value can be negative to indicate that a vehicle is currently driving backwards.

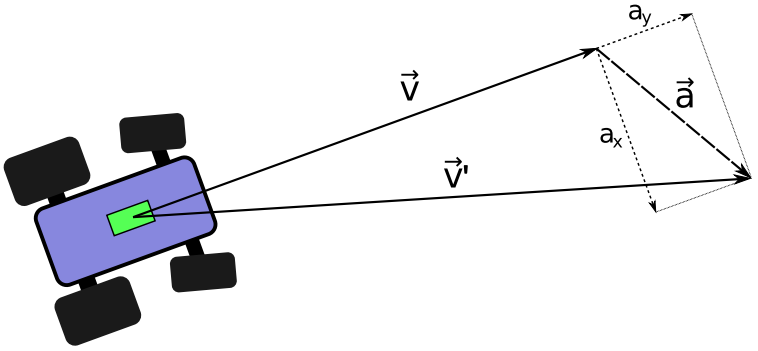


Figure 4.3: Velocity vector integration

During integration, the next velocity value v' is obtained by adding the current acceleration vector $\overrightarrow{a} = \langle a_x, a_y \rangle$ to the current velocity vector as shown in figure 4.3. The magnitude of the resulting vector is the new velocity value. Note that the magnitude $\|\overrightarrow{a}\|$ is greatly exaggerated in this figure: in reality the acceleration vector would be much smaller than the current velocity vector. Mathematically the calculation looks as follows:

$$v' = \begin{cases} \sqrt{a_x^2 + (v + a_y)^2} & v + a_y >= 0 \\ -\sqrt{a_x^2 + (v + a_y)^2} & v + a_y < 0 \end{cases} \tag{4.2}$$

a_x, a_x	acceleration vector components
v	current velocity magnitude
v'	next velocity magnitude

In Section 3.1.1 we explain that a minimum sample frequency is necessary to avoid aliasing effects on the vibration noise that is picked up by the accelerometer. These aliasing effects would cause significant error in the velocity integration. At the end of this chapter in Section 4.3.2 we show that a frequency of 160 Hz is sufficient to obtain a useful velocity estimate. However, this frequency is very high considering the processing bandwidth of our node's processor.

Therefore, we decide perform linear integration on the individual axes first. This yields partial integrations for each axis. Like the compass sensor, we decide to synchronize the execution of the velocity integrator with the controller executions, so that new velocity data is available just before the controller execution. This means that ten accelerometer samples are summed into such a partial integration to yield a frequency of 16 Hz (refer to Section 6.2). In essence, the partial integration is nothing more than taking the average of the ten samples without scaling them down. Therefore, performing the vector integration on this result is still valid, albeit less accurate.

The velocity inference technique we describe here is simple and only uses the accelerometer to obtain a useful result. However, because it is based on integration, it will accumulate sensor errors. As we explain in Section 2.3.2, gravity is the most prevalent cause of sensor error for the accelerometer. In the following two sections we explain how we deal with the accumulating errors and how the accelerometer is regularly re-calibrated for a changing inclination.

4.1.2 Reset Strategy

A problem common to all processes involving the integration of unreliable sensor data is that measurement errors are accumulated as well. This leads to a gradually increasing error in the velocity estimate. If no external reference is available, this error can potentially increase towards infinity, meaning that the velocity estimate is only useful within a specific time span. The length of this time span is primarily dictated by the magnitude and the frequency of errors that occur in the measurements and what level of error is acceptable for the velocity estimate.

If the integrating system is envisioned to work over prolonged periods, a reset method must be designed, setting the velocity to a known value every once in a while. For example, if it is possible to detect that the vehicle is standing still, the velocity estimate can be reset to zero at those instances.

An important characteristic of a moving vehicle is that it produces internal vibrations while moving. If it is standing still and the motor is shut down, no vibrations will we present other than external influences. For this project we assume that the vehicle is not prone to external influences (refer to section 1.1), so this possibility is discarded. This gives us the ability to detect whether the vehicle is moving or not. Vibrations commonly cause high-frequency acceleration components on all three axes of the accelerometer. Therefore, the vehicle's movement status, i.e. whether it is moving or not, can be assessed by keeping a short history of the acceleration measurements and calculating the standard deviation of these. If this deviation is very small, the vehicle is likely to stand still. This was done earlier by Gaysse [18], but he takes it one step further by applying fuzzy logic to further improve the velocity integration.

There is a significant downside to this design. The standard deviation is calculated on a history of measurements, implying that there is a certain delay between the vehicle stopping and the software detecting this status change: the longer the history, the longer the delay. But, if the history is too short, intervals of relative silence on the accelerometer can cause it to reset the velocity to zero at inappropriate times. Such situations may occur if for instance the motor is shut down and the car is coasting forward on a smooth surface. On the other hand, if the delay is too long, the node software may not notice an intermittent stop at all and in any case the velocity is reset to zero much later than is to be desired.

Throughout this thesis we call this technique *motion detection* and we call the involved software the *motion detector*. In Section 4.3.2, we show how this motion detection technique operates on real data and what size is necessary for the measurement history.

4.1.3 Tilt Effects

If we assume that the inclination of the vehicles remains perfectly level all the time, the integration strategy we describe will only suffer from sensor noise. Unfortunately, that situation is highly unlikely; a perfectly level surface to drive on is not realistic. Also, our cars have spring-loaded suspensions, meaning that the car's chassis can tilt even through the surface is perfectly level. Therefore, while driving, the vehicle will gradually tilt to some extent causing the inclination to change.

As we explain in Section 2.3.2, the effect of this change in inclination is that the gravitational acceleration presents itself with changing magnitude on the x and y axis of the acceleration sensor. In section 2.3 we also explain that the only reliable way to compensate for this effect is to use a gyroscopic sensor to keep track of the inclination of the vehicle, i.e. to follow the direction of the gravity vector relative to the vehicle. Currently, we do not have a proper gyroscopic sensor available and we did not assess the feasibility of the necessary calculations on the sensor nodes.

Our current solution is to recalibrate for the vehicle's inclination each time it is standing still. When the vehicle stands still, the stationary acceleration values on the x and y axes are recorded. To prevent outliers from negatively influencing the calibration, an average of measurements is used. The most recent half of the samples recorded for the motion detector is used for this average to make sure the samples are stable. After this calibration process, the recorded calibration vector is subtracted from each acceleration measurement.

If the vehicle maintains the same approximate inclination while driving, the error should be minimal. However, this design is unsuitable for driving the vehicles on rugged terrain. When the inclination changes after the calibration, the acceleration vector used in the velocity integration will become offset. This causes the integration process to accumulate too much acceleration, making the velocity result integrate towards a very large value.

The presented velocity vector integration strategy is evaluated with real data at the end of this chapter in Section 4.3.2. This is also where we present an example of the effect of tilting the vehicle while moving.

4.2 Heading Calculation

In Section 2.3.3, we describe the technical details of compass sensors. In this section, we present how the current heading of the nodes is calculated from the compass sensor data[3]. Figure 4.4 provides an overview of the situation. The compass sensor is subjected to the earth's magnetic field vector $\vec{H}$. The figure presents a two-dimensional top view of the compass sensor with the x and y axes of the sensor in the plane normal to the earth's gravity vector, i.e. the compass sensor is perfectly level. The compass' axes measure the components H_x and H_y of the magnetic field vector $\vec{H}$.

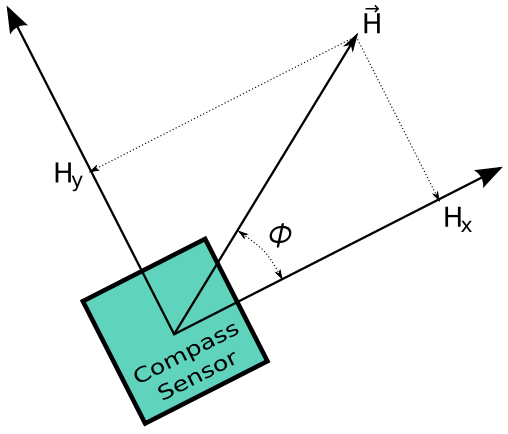


Figure 4.4: Schematic top view of a compass sensor subjected to a magnetic field vector.

[3]Most compass theory presented in this section is explained in much more detail in [8] and [9].

We define the heading of the sensor as the angle between the magnetic field vector and a reference vector relative to the sensor itself. For simplicity, we choose the x axis of the sensor as reference vector[4]. Figure 4.4 shows this angle labeled as ϕ. We apply simple trigonometry to obtain the heading ϕ in the interval $[-180 \ldots 180]$:

$$\phi = \begin{cases} 180 + \frac{180}{\pi} \arctan \frac{H_y}{H_x} & \text{if } H_x < 0 \text{ and } H_y \geq 0, \\ 90 & \text{if } H_x = 0 \text{ and } H_y \geq 0, \\ \frac{180}{\pi} \arctan \frac{H_y}{H_x} & \text{if } H_x > 0, \\ -90 & \text{if } H_x = 0 \text{ and } H_y < 0, \\ -180 + \frac{180}{\pi} \arctan \frac{H_y}{H_x} & \text{if } H_x < 0 \text{ and } H_y < 0 \end{cases} \tag{4.3}$$

where: H_x, H_y magnetic field vector components (axes) measured by the compass
ϕ heading or azimuth in degrees

4.2.1 Compass Resolution

To make the vehicles follow each other, it is important that the compass sensors measure the vehicle's heading with sufficient accuracy. If the compass accuracy is low, the follower will not notice small differences between its own heading and the leader's. If such a small heading difference is maintained while driving a large distance, the distance between the vehicles will change significantly. We deem an accuracy of 1° on the compass to be adequate for our needs, because this means that, with perfect follower control, the cars will deviate less than 2 meters on a traveled linear distance of 100m (refer to Section 1.3).

The explanation in [9] mentions A/D converter resolution, magnetic sensor errors, temperature effects, nearby ferrous materials, compass tilt errors and variations in the earth's magnetic field as sources for error in the compass measurements. In this work we only consider magnetic sensor errors, nearby ferrous objects and compass tilt errors. According to [8] and [9], we need a compass sensor that is sensitive to angular changes as small as 0.1° to achieve a reliable compass resolution of 1° . If that cannot be achieved with a given compass sensor, a 0.5° angular sensitivity is sufficient to maintain an overall resolution of 1° . The reason for the $< 1^\circ$ sensitivity requirement is that noise will influence the measurement and requiring a higher sensitivity makes sure that the compass achieves the required resolution, even with a moderate noise magnitude. Also, the sensor must present a low amount of hysteresis and a high degree of linearity. Regarding linearity, according to the data sheet, the sensor deviates less than 1 % from the straight line in the range ± 300 μT. If the sensor has how hysteresis, it will show deterministic results for a specific field strength, which is something we test empirically in Section 2.4.1.

First, we need to assess whether we can achieve an angular sensitivity of 0.1° using our compass sensor. We use Equation 4.3 to infer the required resolution of the individual magnetometer axes for achieving a 0.1° angular sensitivity on the whole compass. The critical points in this equation that determine the achievable sensitivity occur when either H_x or H_y is close to 0, because that is where changes in H_x and H_y, respectively, are smallest when compared to the change in angle ϕ. When we consider Equation 4.3 for when ϕ is close to 0 ($H_y \to 0$) we obtain:

$$\begin{aligned} 0.1^\circ &= \frac{180}{\pi} \arctan \frac{H_y}{H_x} \\ \frac{H_y}{H_x} &= \frac{1}{573} \end{aligned} \tag{4.4}$$

This means that, for a 0.1° change in the heading angle, the fraction $\frac{H_y}{H_x}$ will change by an amount of $\frac{1}{573}$. When $\phi = 0$, the value of H_x will have reached its maximum and $H_y = 0$. When the compass is level, as required for Equation 4.3 to be valid, the maximum of H_x will be equal to 21 μT (refer to Section 2.3.3). Using the result of Equation 4.4, we can now infer what the magnetometer axis resolution must be to achieve

[4]We do not need to choose this reference vector to be pointing towards the front of the vehicle on which the compass sensor is mounted, because there is no problem if both leader and follower use the same reference. Thus, if the compass sensors are mounted in the same orientation on both vehicles, the x axis is a straightforward choice.

an angular sensitivity of $0.1°$:

$$\begin{aligned} \frac{H_y}{21\mu\mathrm{T}} &= \frac{1}{573} \\ H_y &= 21\mu\mathrm{T}\frac{1}{573} = 0.037\mu\mathrm{T} \end{aligned} \tag{4.5}$$

This result indicates that the compass sensor must have sufficient resolution to represent changes in the magnetic field as small as 0.037μT on each of its axes. This calculation yields the same result at the other three critical positions of Equation 4.3 on the unit circle.

In Section 2.3.3, we explain that the Micromag 3 compass sensor we use has a gain value of 62.48 $counts/\mu$T when configured at the highest period count value. This means that the digital output value will increase by an amount of 62 for every μT increase in magnetic field strength. The reciprocal of this value yields a potential magnetometer resolution of 0.016 μT, which would be more than sufficient to achieve the $0.1°$ sensitivity.

However, a high period count dictates a low sample frequency and that is why we aim for a value optimal to our needs. Table 2.1 at page 24 lists the various configuration options along with the achieved resolution and measurement delay. From that table we can conclude that, if we aim for a $0.1°$ angular sensitivity, a period count value of at least 2048 is required. If we only aim for a sensitivity of $0.5°$, we obtain a required magnetometer axis resolution of 0.18 μT, making a period count value of 512 sufficient.

According to the table, requiring a $0.1°$ angular sensitivity limits the sampling rate of the sensor to 9Hz. This is very low considering that the used vehicles can steer at a rate beyond 90 degrees per second (refer to Section 2.2). At a compass sampling frequency of 9 Hz, this would mean that the heading measurements can lag behind more than $10°$. That is why we choose to use a less sensitive period count of 512, which achieves an angular sensitivity better than $0.5°$ and that is according to [9] sufficient to maintain an overall compass accuracy of $1°$. At this period count, the sample frequency can be as high as 66 Hz when only the X and Y axes are sampled as described in Section 4.2.3. In Section 6.2 we choose a sampling frequency of 16 Hz to match the follower's control frequency.

4.2.2 Compass Calibration

In Section 2.4.1, we briefly show the sensitivity of the compass sensor to ferrous metal objects that distort the earth's magnetic field. If these ferrous objects are part of the vehicle itself, their presence cannot be avoided. However, the position of those vehicle components will remain constant relative to the sensor and therefore those distortions are constant, meaning that the effect of these distortions can be compensated for. We can also compensate for the static effect of any magnetic materials used in the vehicle using the same method. Note that we do not consider the effect that external objects might have on the compass sensor; we consider only interference by parts of the vehicle itself.

When an ideal compass sensor is rotated in a uniform magnetic field, i.e. without distortions, in the plane of its x and y axis, the compass should yield sinusoidal signals on the x and y axis with zero offset and equal amplitude. When y is plotted against x one would obtain a perfect circle with its center at the origin. When compass is influenced and not calibrated accordingly, this plot will be not centered at the origin and possibly have an elliptic shape, indicating that the arctangent function of Equation 4.3 will produce incorrect results.

The interference caused by ferrous or magnetic components of the vehicle can be compensated for by using calibration. These effects are subdivided into two types of magnetic distortion: *soft iron distortions* and *hard iron distortions* [9]. These relate to the common distinction between hard and soft magnetic materials. Hard magnetic materials like steel can be permanently magnetized when subjected to a strong magnetic field, whereas soft magnetic materials do not retain the magnetism, but are easily magnetized temporarily. Soft magnetic materials are often specially-crafted alloys, e.g. for the manufacture of electrical transformers. Hard iron distortions cause a constant offset in the compass axes readings as long as the position of the objects relative to the sensor remains constant. These effects can be compensated by determining and subtracting that offset. This means that hard iron distortions are relatively easily compensated for. In contrast, the soft iron distortions cause orientation-dependent errors in the compass output.

Before calibrating it is important to make sure that the sensor is mounted rigidly at its definitive position, orientation and inclination on the vehicle, since the calibration will only be valid for that specific configura-

tion. The calibration procedure presented here is suitable to compensate offset and scale errors caused by metal distortions and manufacturing differences between the sensor's individual axes magnetometers. Soft iron effects are often more complex than a simple scaling error and will only be partially compensated ([9] describes details and a more complex alternative). However, soft magnetic materials are much less common than hard magnetic materials on a vehicle, meaning that the procedure we describe here should be adequate.

The goal of the calibration procedure is to obtain compass axes signals with equal amplitude and a common offset at zero. The calibration procedure involves turning the sensor along with the platform it is mounted on, e.g. the vehicle, a few times to determine the maximum and minimum of the axes signals. For the vehicles it is often easiest to make them drive circles. The following calculations yield the scale and offset values for each axis using the maximum and minimum measurements:

$$\begin{aligned} X_{scale} &= \max\left(1, \frac{Y_{max}-Y_{min}}{X_{max}-X_{min}}\right) \\ Y_{scale} &= \max\left(1, \frac{X_{max}-X_{min}}{Y_{max}-Y_{min}}\right) \end{aligned} \tag{4.6}$$

$$\begin{aligned} X_{offset} &= \left(\frac{X_{max}-X_{min}}{2} - X_{max}\right) * X_{scale} \\ Y_{offset} &= \left(\frac{Y_{max}-Y_{min}}{2} - Y_{max}\right) * Y_{scale} \end{aligned} \tag{4.7}$$

Equations 4.6 yield scale values X_{scale} and Y_{scale}, of which one is chosen as reference (= 1) and the other has a scale value relative to that. Equations 4.7 yield the offsets X_{offset} and Y_{offset} of the signals, scaled appropriately with the previous results. The scale and offset values are applied to raw sensor values as follows:

$$\begin{aligned} H'_x &= H_x X_{scale} + X_{offset} \\ H'_y &= H_y Y_{scale} + Y_{offset} \end{aligned} \tag{4.8}$$

The result of this procedure is that, as long as the calibration remains valid, the offset of both signals is zero and their amplitude is identical. This yields a good circular response for the compass.

4.2.3 Compass Tilt Compensation

Just like the accelerometer, the compass sensor is not going to be kept at a perfectly constant inclination when it is mounted on a vehicle. Without measures to compensate for inclination effects, Equation 4.3 will produce significant heading errors when the compass inclination changes. The calibration procedure we explain in Section 4.2.2 is only valid for the inclination the compass was calibrated at. This means that it is possible to use the calibration procedure for any given inclination as long as that does not change. A compass subjected to dynamic inclination needs active tilt compensation.

To compensate for tilt effects, the exact tilt angles need to be known. This can be measured accurately using a two-axis accelerometer when the vehicle is standing still [50]. But, as we explain in Section 2.3, this will not work well when the vehicle is moving. A more dynamic means to keep track of the current inclination of the sensor platform is to use a gyroscope.

When the tilt angles are known, we can apply the equations provided in [9]:

$$\begin{aligned} H'_x &= H_x \cos(\psi) + H_y \sin(\theta)\sin(\psi) - H_z \cos(\theta)\sin(\psi) \\ H'_y &= H_y \cos(\theta) + H_z \sin(\theta) \end{aligned} \tag{4.9}$$

where:	H_x, H_y, H_z	tilt-influenced compass measurements
	H'_x, H'_y	compensated output
	ψ	pitch tilt angle
	θ	roll tilt angle

Although these equations are shown to be valid in Section 4.3.1, we do not make use of these. Currently, we assume our vehicles to maintain a mostly constant inclination. Especially on the hockey field we describe

in Chapter 9 this should be a valid assumption. In this situation, the proposed accelerometer solution for measuring the inclination is bound to do more harm than good, because it adds a new error source to the heading calculation while the inclination remains mostly constant. The gyroscope alternative is currently not available to our research.

Because we do not use dynamic tilt compensation, sampling the Z-axis of the sensor is not very useful. To make scheduling the compass sampling a less daunting task, we currently omit sampling the Z-axis (refer to Section 3.1 and Section 8.2.2).

4.3 Real Data

In the first sections of this chapter we outline the theory behind velocity integration and heading calculation. In this section we show how these sensor data processing steps work with real data from the toy cars. Part of this evaluation deals with the effects of tilting the sensors. Also, the accelerometer sample frequency necessary for sufficiently accurate velocity integration is determined by experimentation.

4.3.1 Heading Calculation

To show how the heading calculation presented in Section 4.2 works with real data, we take the data from the tilt experiment presented in Section 2.4.1 and apply Equation 4.3 to obtain a heading result for the two separate sensors. If calibration works efficiently, both sensors should yield the same heading result.

The measured signals are first filtered and calibrated to obtain clean signals with a zero offset and uniform amplitude. Note that the final system we present in this work does not apply a filter on the magnetometer data, apart from a simple function that removes spurious spikes from the signal. This filter would otherwise cause additional delay in the data processing, which in turn would increase the reaction time of the follower. For this presentation this is not an issue and to obtain the best possible results a rather aggressive 40-tap running average filter is used to smoothen the signals.

Figure 4.5 shows the results of this filtering in the top two graphs. Clearly, the signals are shown in their uncalibrated form. As we notice in Section 2.4.1, the signals from the two separate sensors correlate well. The third graph shows the heading result for the two sensors and the fourth graph shows the difference between the two results. Clearly, the calibration works very well for these clean signals and yields very similar results for both sensors. According to the bottom graph, the standard deviation between the two signals is less than 2°. And considering the average error of 1°, the sensors may not have had exactly the same heading. The maximum absolute error barely exceeds 4°.

On the nodes, however, the results will be less reliable, as the signals from the sensor are subjected to a significant amount of noise caused by the moving car (refer to Section 9.2.1). And, without filtering, this noise will have an effect on the resulting heading. That is why [9] advises to aim for an angular sensitivity of 0.1° to provide sufficient margin for the noise. The 0.5° sensitivity that we can achieve with our chosen sensor configuration (refer to Section 4.2) will present heading errors of more than 1° if the noise level rises above 1 bit in magnitude. On the assumed smooth driving surface (refer to Section 1.1) this problem should be less significant.

As we discuss in Section 4.2, the heading result is also influenced by inclination changes. To show this, we perform another similar experiment with the first sensor mounted in a 45° inclination. We first calibrate the setup with both sensors in level position. Then, with the first sensor inclined, the platform is turned several times. Figure 4.6 shows the result of the heading calculation. The inclination change after calibration causes the offset and amplitude of the signals to change. Clearly, the effect on the calculated heading is devastating: the result does not even span the full circle anymore, meaning that the vehicle will never be able to achieve a large range of measured headings.

Section 4.2 presents a solution to this problem using the pitch and roll tilt angles to infer what the magnitude of the magnetic vector components would have been in the level plane. Figure 4.7 shows the potential merit of this technique: the values of the level and the inclined sensor match up again, although with less precision

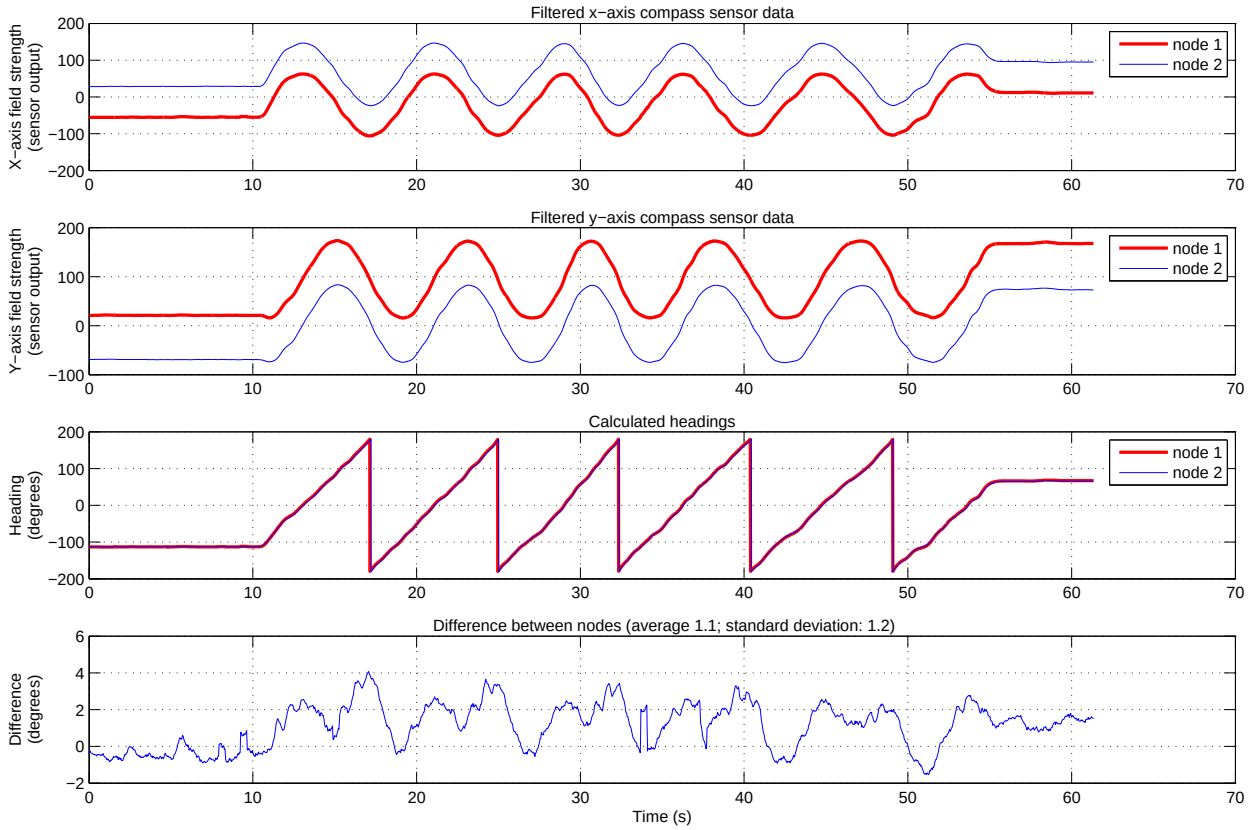


Figure 4.5: Heading calculation using the signals from two compass sensors revolving continuously on the same platform.

than the results of Figure 4.5. Probably we did not measure the inclination of the sensor correctly, because the results of Figure 4.7 were achieved by filling an angle of 38° into Equation 4.9. Also, because only one sensor is tilted, their relative orientation changes, which has an impact on the calibration as well due to the metal influence the sensor nodes can have on each other.

We decide not to use the tilt compensation technique presented in this chapter, because we cannot obtain continuous tilt information while the car is driving. For that, we would need a gyroscopic sensor. Intermittently reassessing the current inclination as is done for the velocity integration is deemed of little use. As shown for the experiment above, a high accuracy tilt measurement is necessary to properly compensate for inclination changes; slight changes in the vehicle's inclination will make intermittent measurement useless. Also, if the driving surface is relatively level, the intermittent inclination measurements will measure approximately the same value all the time and consequently inclination changes are only caused by the movement of the vehicle and not the ruggedness of the surface it is driving on.

4.3.2 Velocity Integration

In this section we show how the velocity integration algorithm we present in Section 4.1 works with acceleration data collected from a real vehicle. Unlike the experiments of the previous section about heading calculation, we do not use the prototype platform we describe in section 2.4 for the evaluation of the velocity integration, because its 100 Hz accelerometer sampling frequency turns out to be too low. We use a modified early version of the definitive system to attain higher frequencies. Only one sensor node is considered in these tests, meaning that the MAC protocol used by the prototype is not necessary to propagate the measurements. In this section, we show the performance of the integration algorithm at various sampling frequencies ranging up to 320 Hz.

All tests are performed indoors in the offices or the hallway of our faculty building. First, we evaluate the properties of the algorithm in its simplest form, i.e. along a single axis, at varying sampling frequencies.

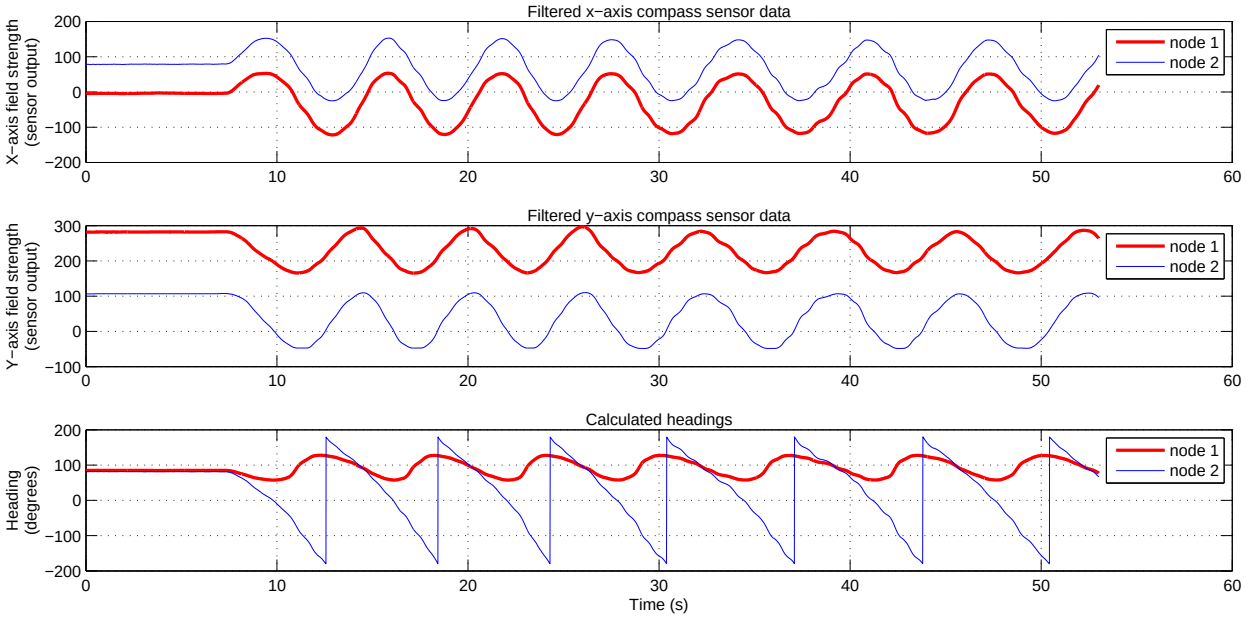


Figure 4.6: Heading calculation with one sensor tilted 45° and the other level (no compensation).

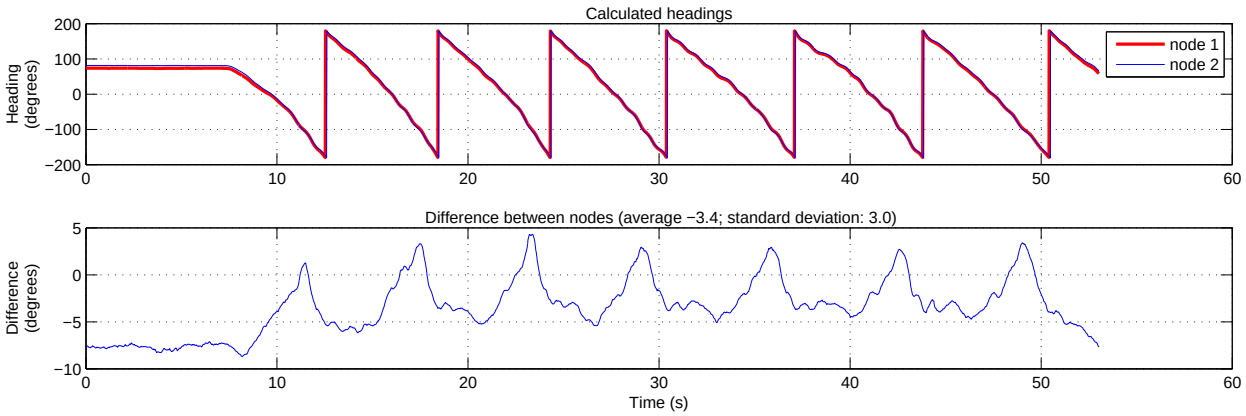


Figure 4.7: Heading calculation with one sensor tilted 45° and the other level (compensated).

Then, the results are improved by applying the motion detection technique we describe in Section 4.1. Finally, we show how the vector integration algorithm works when the vehicle drives circles.

Sample Frequency

Figure 4.8 shows the results of the integration process for the acceleration data obtained from a toy car moving forwards and backwards respectively. In the backwards run the vehicle was stopped more quickly by applying forwards throttle. For this test, the acceleration sensor is initially calibrated for its inclination and therefore the sensor reads zero until the car starts moving. The sensor is mounted on the car with the y axis pointing backwards. The integration is performed only along this axis.

For this particular test, the acceleration data is sent to the gateway at a rate of 25 Hz. Note that the final system only communicates acceleration data for logging purposes: each mobile node runs the velocity integrator locally. For this test, however, the integration process is performed in Matlab.

As shown in the figure, this test yields acceleration data at two different sample rates for comparison: 25 Hz and 250 Hz. Due to the fact that our communication protocol works at 25 Hz and the packet size is only 32 bytes, the 250 Hz data cannot be sent to the gateway for logging in its raw form. In stead, the sum of the

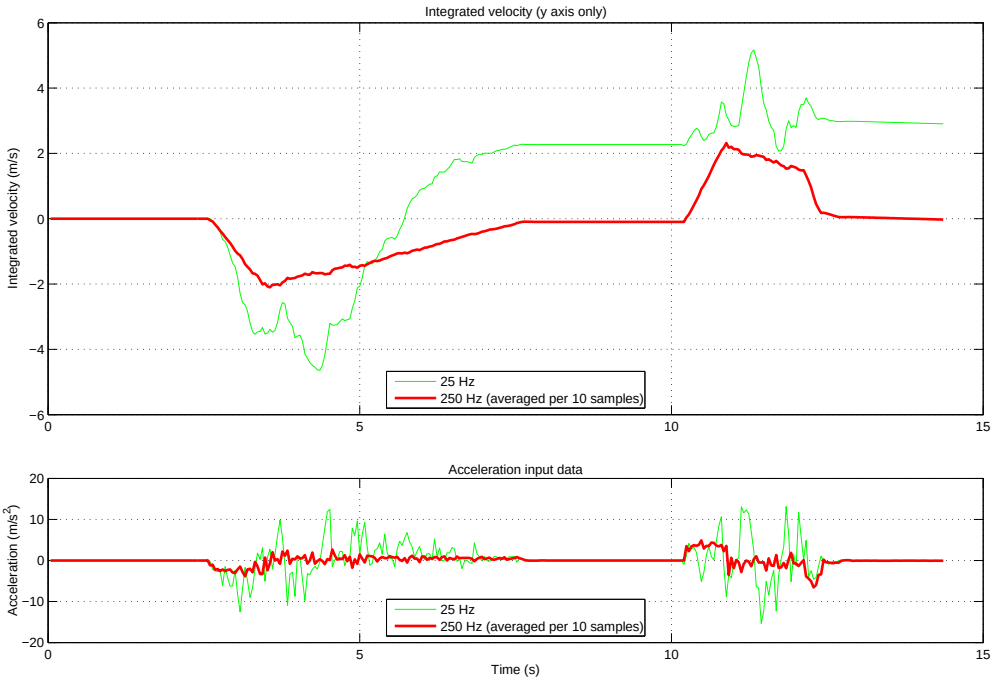


Figure 4.8: Velocity integration of acceleration data collected from a toy car driving forwards and backwards.

last 10 samples is sent over the radio at 25 Hz. This is the same optimization as used for the actual velocity integrator that runs on the node. This is explained in Section 4.1.1. If the result is additionally scaled down to the same magnitude as the 25 Hz result, we obtain the integration result of the 250 Hz samples averaged at 25 Hz. This is what is done for the figure, so that both results have the same order of magnitude.

By looking at Figure 4.8 it is obvious that an acceleration sampling frequency of 25 Hz is inadequate to obtain a useful velocity result. Where the 250 Hz result shows very smooth acceleration and deceleration in both driving forwards and backwards, the 25 Hz result does not show the acceleration and deceleration stages in a distinguishable manner. The 25 Hz result is very noisy, which is most probably due to aliasing effects caused by the low sampling frequency. This noise accumulates into a velocity offset that becomes visible right after the car has stopped for the first time. In contrast, the 250 Hz result returns nicely to zero that first time. However, after the car's second movement the 250 Hz result also shows a non-zero velocity, which is probably caused by inclination changes of the acceleration sensor.

Figure 4.9 shows a similar experiment for which more than two sample frequencies were tried simultaneously. As we describe in Section 2.3.2, the accelerometer can be configured only at a specific set of physical sample frequencies. We configured the sensor at 640 Hz for the experiment in Figure 4.9. The sensor node, however, cannot read the sensor at that very high frequency. Therefore, the sensor node reads the accelerometer at 320 Hz skipping every odd sample. For the lower frequencies in the figure, samples were skipped more often accordingly. Messages are transmitted at 20 Hz to the gateway, meaning that for 320 Hz 16 samples are accumulated, for 160 Hz 8 etc. The message composition is changed to accommodate for the additional acceleration results, meaning that compass measurements are left out.

The experiment involved driving the car forwards and backwards over an approximately straight line again. As expected, the 320 Hz integration yields the best results. Interestingly, the 160 Hz result does not perform significantly worse. Starting with 80 Hz, the performance gradually decreases. Like the 25 Hz result before, the 20 Hz result is very noisy and completely fails to return to a value anywhere near zero when the vehicle stops. The results suggest that the noise mainly consists of frequencies below 80 Hz. The exact value of that noise bandwidth is not evident from these tests, but we choose to use 160 Hz as sampling frequency, since the accelerometer directly supports that. The results also show that aiming for a higher sampling frequency does not improve results significantly. In Section 6.2, we use this frequency as a basis for determining the follower's control frequency.

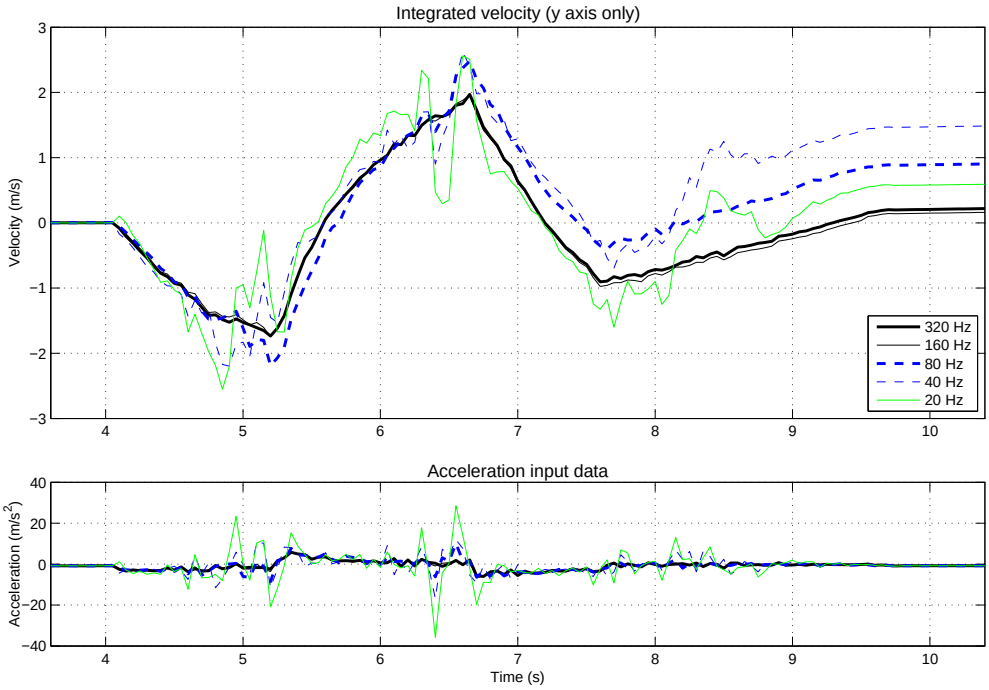


Figure 4.9: Velocity integration of acceleration data sampled at different frequencies.

Improvement using Motion Detection

The test performed for Figure 4.8 involved little or no changes in the vehicle's inclination: the acceleration signals did not gain an offset during the tests although calibration was only performed at the beginning of test. In Section 4.1 we mention the problems that changes in gravity can cause. We also present a solution using a motion detection technique. The idea is to stop integrating when the vehicle is standing still. This way the velocity can be reset to a known value, preventing errors from accumulating indefinitely. Motion is detected by calculating the standard deviation over the latest measurements. If this value is higher than a specific threshold, the vehicle is deemed to be moving and the velocity integration active. If it drops below the threshold, the velocity value is reset to zero and the sensor is re-calibrated for the current inclination.

Here, we present an example of what happens in practice. Figure 4.10 shows the integration results of another test run with the toy car. The vehicle accelerated, drove a short distance and ended up with its front wheels on a bundle of cables. This causes a change in the inclination of the vehicle. The integration result is presented with and without the motion detection technique.

The naive alternative, i.e. the result without motion detection, shows a linear increase in velocity after the car has stopped on the cables. This is caused by the constant offset in the acceleration measurements that emerged with the change in inclination. The velocity quickly rises to a value far beyond what is valid. If this were the leader vehicle, the follower would accelerate to full speed while the leader would just be standing still.

The alternative using the motion detection technique performs much better. In Figure 4.10, the motion detector works on a history of 10 samples for both axes. As shown, the detector quickly resets the velocity value to zero after the vehicle stops moving. The velocity reset causes a characteristic discontinuity in the velocity output. This is also encountered in the results we present in Chapter 9. Through more experimentation we found that a history of 100 samples with a standard deviation threshold of 4 on the raw accelerometer data is is sufficient to provide reliable motion detection. At 160 Hz this means that it takes at worst $\frac{100}{160} = 625$ ms for a stand still to be detected after the vehicle has stopped completely.

Interestingly, the acceleration input data is noisy sometimes even when the vehicle is standing still. This noise is caused by an error in the sensor driver software used during this test. However, this noise is smaller than the threshold of the motion detector in the test shown in Figure 4.10. Therefore, the velocity integration result that uses the motion detection technique remains unaffected. In contrast, the naive integration yields

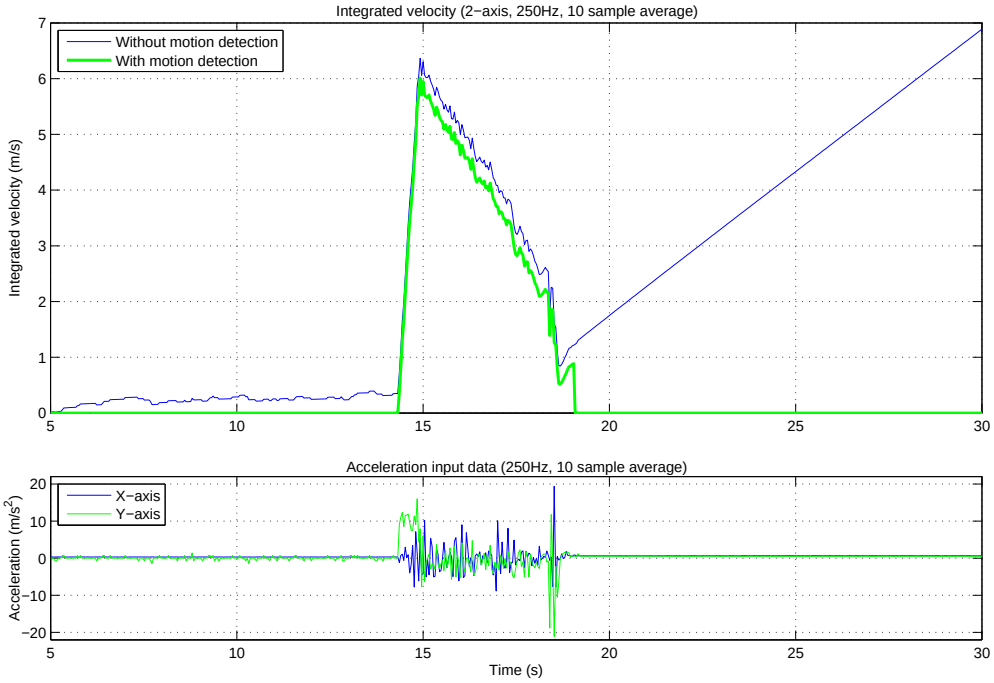


Figure 4.10: Velocity integration of acceleration data collected from a toy car briefly driving forwards and ending up slightly inclined.

a slowly increasing velocity although the vehicle is standing still.

Driving Circles

The results we present above all pertain to linear vehicle movements. To truly show the operation of the vector velocity integration we let the vehicle drive circles. This produces a very recognizable result if the integration is effective and it is relatively easy to drive circles continuously with the given toy car.

We present circular input acceleration data for the integration algorithm in Figure 4.11. Just like before, this acceleration is acquired at 250 Hz and summed per 10 samples. To make the acceleration progression more recognizable in the graph, it is additionally filtered with a running average filter with a window of 10 samples. The x axis of the sensor pointed to the left of the vehicle; towards the center of the circular movement and the y axis pointed towards the back of the vehicle.

Interestingly, the x axis shows a very constant and large signal that lasts just as long as the circular movement. This is the centripetal force that keeps the vehicle in the circular movement. It is positive because the x axis of the sensor points towards the center of the circle. The y axis first shows the initial acceleration towards the forward velocity that is upheld during the circular movement. Subsequently, the acceleration measured on the y axis maintains a constant value. Finally, after 40 seconds, the y axis shows the deceleration towards a full stop. The constant value on the y axis during the circular movement is a component of the centripetal force as well, which is present because the x axis does not point precisely towards the center of the driven circle. Otherwise the y axis would be close to zero, because the velocity appeared very constant during the test.

The integrated velocity is shown in the top graph of Figure 4.12. Clearly, the motion detection technique we demonstrate above is used, because the velocity is zero when the car is not driving. As one would expect from a circular movement, the forward velocity of the car remains relatively constant. If, like before, linear integration on the y axis were applied, the velocity would increase linearly, because this axis measures a constant non-zero value during the movement. Such velocity result would of course neither be valid nor useful.

It is very fortunate that this particular test resulted in the velocity integration returning to a value very close

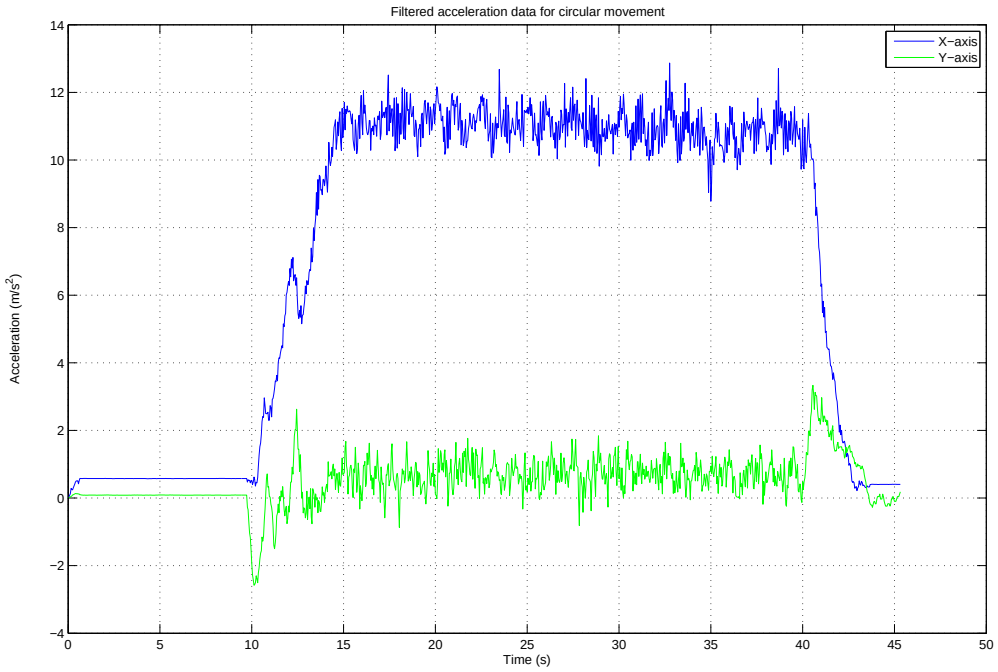


Figure 4.11: Filtered acceleration data from vehicle driving continuous circles.

to zero on its own. The car was driven at a very constant velocity, avoiding changes in its inclination that could be caused by the car's suspension. For that reason, this test should not be interpreted as an evaluation of the vector integration algorithm. As we show in Chapter 9, the results can be much less satisfying if the vehicle is driven in a random manner.

The lower graph of Figure 4.12 shows the heading of the vehicle during the test. First of all, this heading was measured using the compass sensor. Although the compass sensor is calibrated and the velocity of the vehicle was very constant during the test, the heading calculated from the compass readings is not a linearly progressing value. Again, the compass is affected by the nearby metal influences of the faculty building.

The other heading plot in that graph is very interesting. It is the heading as integrated from the acceleration data. The vector integration algorithm presented in Section 4.1 only considers the magnitude of the velocity vector. As Figure 4.3 shows in an exaggerated manner, the velocity vector v changes its heading due to the acceleration vector a. The angle between v and v' is the angular change that results from the acceleration vector a. If this angular change is integrated starting at heading zero, we obtain an estimate of the current heading. This heading estimate is shown as the second plot in the lower graph of Figure 4.12.

The resemblance with the compass data is striking. In terms of smoothness, it looks much better than the compass data. Note that it is pure coincidence that the graphs overlap very well in the middle of the circular movement. This no prerequisite for the heading results from the accelerometer to match up with the compass results. The zero start value of the angular integration is arbitrary and the compass reports a different initial heading. Due to an integration error they align just after the movement starts. To compare the performance of the angular integration with the compass result it is enough compare the slope of the heading graphs, which is also seen as the frequency of the sawtooth graphs. If this slope is approximately equal, meaning that the angular velocities are equal, the compass and accelerometer angle measurements can be said to match up.

The heading results we present in Figure 4.12 give rise an interesting research question: can the angle estimate that is integrated from accelerometer measurements be used to improve the compass measurements? The advantage of the compass is that it provides an absolute heading reference, while the integrated heading progresses very smoothly without influence from nearby metal objects. Combined, these values could provide a more reliable heading estimate for the car. This combination could be achieved using sensor data fusion techniques [19], perhaps even fuzzy logic [25, 30].

Unfortunately, the test presented in Figure 4.12 is very ideal with respect to the velocity integration results.

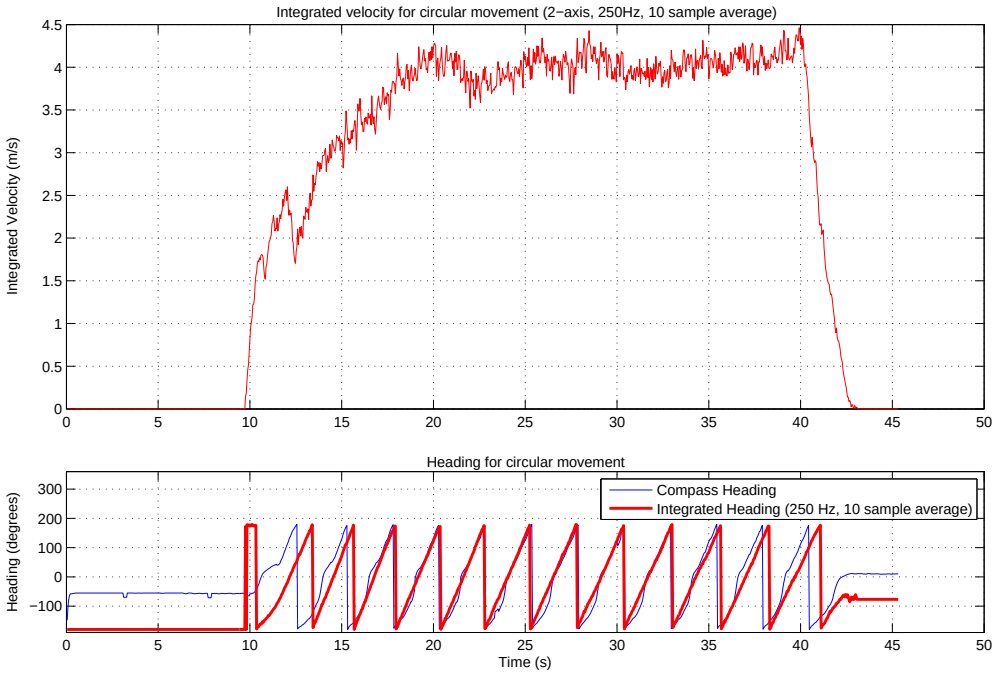


Figure 4.12: Integration results from circular movement showing integrated velocity and heading.

If the velocity estimate integrates to a far too high value due to gravity influence or noise, the angular change would be calculated using invalid velocity vectors, yielding erroneous results. So, the option suggested above is only possible as long as the velocity integration is not compromised by accumulating errors. Otherwise, the fused heading result would only get worse. If a three-dimensional gyroscopic sensor is used, the effect of gravity can be compensated. However, this would make the inference of angular velocity from the linear acceleration less useful, because the gyroscope can measure this directly and more reliably. Also note that this can only work for vehicles that cannot change their heading without moving forward.

To provide a rough idea of how well our integration can perform in the more ideal situations, Figure 4.13 shows the next integration step using the results presented in Figure 4.13. The integration of the complete velocity vector, i.e. using both its magnitude and heading, over the positional x and y axis yields an estimate of the path the vehicle traveled during the test. The path starts at location $(0, 0)$. The circular movements are clearly distinguishable. Apart from the first few, the diameter of the circles is very constant. The circles gradually deviate from the initial position, but this could partly have happened in reality as well: this was not measured, but it is difficult to keep the car on the same path all the time.

4.4 Summary

This chapter describes the data processing required to obtain the heading and velocity of the vehicles from the raw sensor data. The velocity is obtained by integrating accelerometer measurements and the heading is calculated using an arctangent on the horizontal magnetic vector components from the compass sensor.

We describe the following considerations regarding velocity integration:

- Our velocity integration uses only the accelerometer: We do not use the commonly employed inertial navigation techniques that require an additional gyroscopic sensor. We integrate the velocity purely using the accelerometer. We do this by assuming that the car can only drive backwards and forwards and only changes its heading when it has velocity. In each integration iteration, the current acceleration vector is added to the current velocity vector, which always points to the front of the vehicle.

- Velocity integration will accumulate sensor errors: As any integration process, the integration of accelerometer samples into a velocity estimate accumulates sensor errors into the velocity result. There

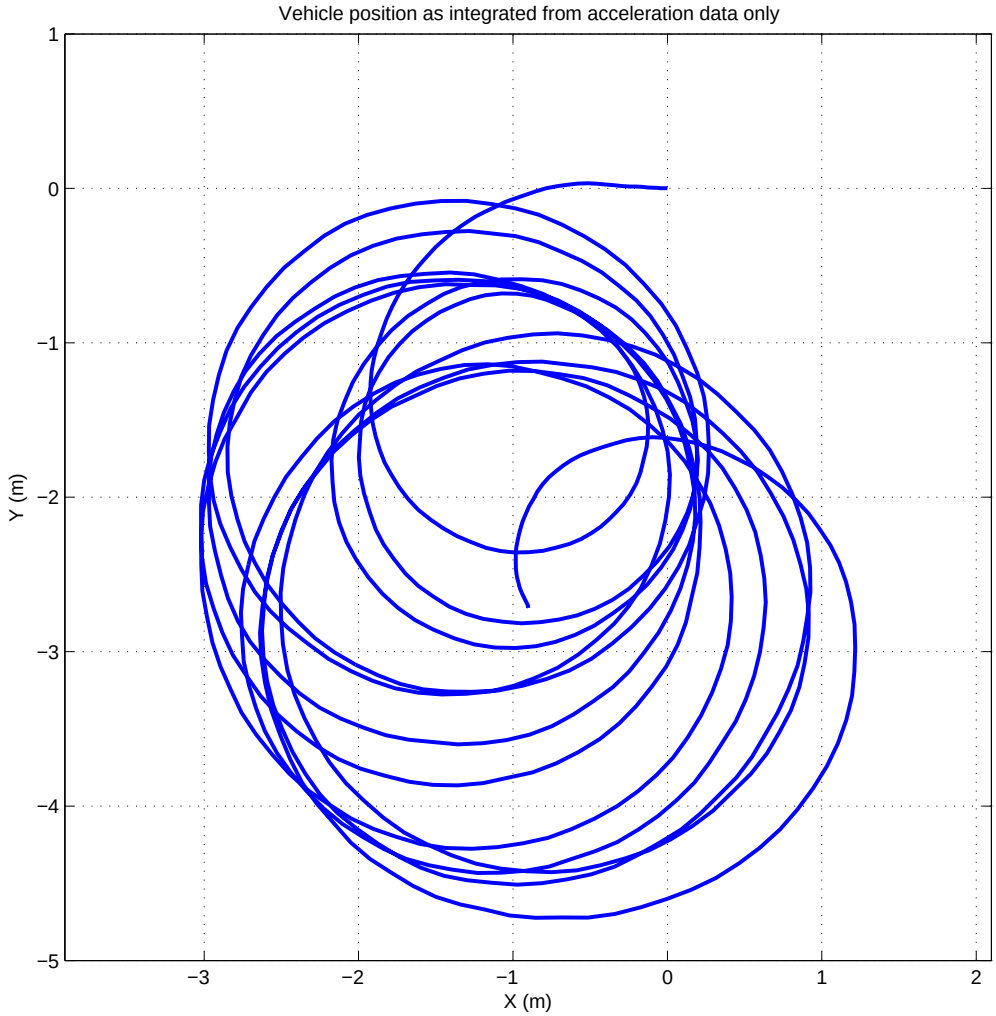


Figure 4.13: The position of a vehicle driving circles as integrated using acceleration data.

is no way to prevent this, other than providing noise-free sensor readings. A prominent cause for error in the accelerometer readings is the gravity influences caused by changing inclination. To prevent the accumulating error from increasing towards infinity, the velocity estimate must be reset to a known value at regular intervals.

- Accelerometer samples are first summed per 10 samples: In Section 4.3.2 we show that a sample frequency of 160 Hz for the accelerometer yields smooth velocity integration results. However, it is a bad idea to run the velocity integration algorithm at this rate on the resource-limited sensor node. Therefore we run the algorithm on the sum of 10 samples at 16 Hz to match the control frequency chosen in Section 6.2.
- Vehicle motion is detected using the standard deviation of the acceleration signal: To counter the potentially infinite accumulation of errors by the velocity integration algorithm, the velocity estimation is reset to zero when the vehicle is standing still. This situation is detected by assessing the standard deviation of the accelerometer signal. If the vehicle moves, it vibrates and the acceleration signal is very noisy yielding a high standard deviation. If it is standing completely still, it is mostly silent yielding a low standard deviation.
- The accelerometer is calibrated for gravity at each standstill: Each time the vehicle is detected to stand still the current accelerometer measurements are recorded. These values are subtracted from the accelerometer measurements during subsequent driving to compensate for the static gravity influence.

We outline the following considerations for the compass heading calculation:

- We aim for a 1° compass resolution: This limits the deviation of the vehicles to a value below 2 m

over a linear stretch of 100 m. To obtain a reliable 1° resolution, an angular sensitivity of 0.1° would necessary to account for significant noise. This, however, limits the sample frequency of the compass sensor to 9 Hz, which is low considering the agility of our toy cars. In stead we configure an angular sensitivity of 0.5° .

- The compass is calibrated by equalizing the scale and offset of the axes signals: An uncalibrated compass sensor shows significant differences in the scale and offsets of the signals from its X and Y axis. Calibration entails making several full revolutions with the sensor and recording the maximum and minimum output values on each axis. Using these values the relative scale and offset for each axis can be calculated using simple formulae. The scale and offset values are used to convert the uncalibrated measurements to appropriate input values for the arctangent function.
- We do not use dynamic compass tilt compensation: When the pitch and roll angles are measurable during driving, dynamic inclination changes can be compensated, making a calibration for the level plane valid for any other inclination. This requires a solution like a gyroscopic sensor, which is something we currently avoid using. Also, the necessary calculations are quite involved for our resource-limited sensor nodes.

Towards the end of this chapter we describe how the presented data processing algorithms perform on real sensor data. We make notice of the following important results:

- Measurements from different compasses correlate well: If two compasses are mounted level on the same moving platform and are calibrated appropriately, they measure approximately the same heading with, for a typical experiment, a standard deviation of little more than 1° and a maximum of 4° on some positions on the full circle.
- The effect of inclination changes on the compass is very significant: Inclination changes of more than a few degrees have a very significant effect on the calculated heading. Compensation is possible, but that requires the pitch and roll angles to be known with high precision during driving.
- A sample frequency of 160 Hz is appropriate for the accelerometer: In an experiment where the linear movement of a toy car is measured at 5 different frequencies ranging from 20 to 320 Hz, the 160 Hz velocity integration result shows smooth acceleration and deceleration. The 320 Hz result shows only marginal improvement and the 80 Hz is significantly worse. That is why we use 160 Hz as sample frequency. That is also conveniently one of the natively supported sample frequencies of the accelerometer, meaning that no samples are skipped by the node CPU.
- The motion detection works correctly: The motion detection technique using the standard deviation of the acceleration signals is in practice an effective means to assess whether the vehicle is moving or not. Using this assessment it is possible to reset the velocity to zero within one second after the vehicle has stopped. If not applied and the inclination of the vehicle changed during driving, the integrated velocity increases linearly to infinity after the vehicle has stopped.
- When driving circles our velocity integration algorithm shows valid results: The integration of the acceleration vector of a toy car driving circles at a relatively constant velocity yields a relatively constant velocity result. The integrated velocity returns to a value close to zero when the vehicle eventually stops. The velocity integration algorithm can be amended to also yield an angular velocity estimate based on the angle between current and new velocity vector. When integrated, this angular velocity estimate yields a heading estimate that correlates well with the compass output, as long as the integrated velocity is not affected significantly by accumulated errors.

Chapter 5

Communication

For this work we employ a set of only three wireless sensor nodes: a leader, a follower and a gateway. To propagate the movement messages from the leader node to the follower node and from both these mobile nodes to the stationary gateway, a radio protocol is needed. In this chapter we outline the design of this protocol.

It is not one of the goals of this work to design an intricate protocol in which many nodes can communicate. The only thing we really want to achieve is basic communication between the three nodes involved. The protocol we built for the prototype platform that we discussed in Section 2.4 is very demanding in terms of synchronization and the mobile nodes always need to have contact with the gateway. To prevent protocol-related issues from influencing our results, we keep the protocol very simple.

Note that future work can involve more nodes. But, as long as the leader is the only node that broadcasts movement data, the one leader to multiple followers communication pattern remains feasible. For evaluation, however, logging the movement data of all nodes towards the gateway is necessary, making communication more difficult as nodes are added to the system.

5.1 Radio Protocol

The radio protocol must support two flows of information: from the leader to the follower and from both these nodes to the gateway for logging purposes. Although it would be useful to give the gateway the ability to send back messages to the nodes, e.g. to stop the follower node if it is misbehaving or to set new calibration values, we feel that it is more important to keep the protocol simple and straightforward. Therefore, in this protocol, the gateway is strictly an observer that never sends any messages of its own. Figure 5.1 provides an overview of the communication paths in the protocol between the sensor nodes. While the leader broadcasts its measurements to both follower and gateway and the follower broadcasts to the gateway, the gateway never sends any messages on the radio. The gateway just forwards the incoming radio messages over the serial connection to the computer for logging (refer to Appendix A.3 for a more detailed description of the gateway and its serial link).

The designed protocol assigns the coordination task to the leader. The leader continuously broadcasts movement messages at regular intervals. At the instant the follower receives the message, it schedules a transmission of its own. After a minimum delay period, it sends a similar message of its own. Messages from leader and follower are subsequently picked up by the gateway. The short delay period at the follower ensures that the gateway is ready to receive its message, since it needs to process the incoming data from the leader first.

A sequence diagram of the protocol is shown in Figure 5.2. This sequence diagram displays two subsequent protocol iterations and the start of a third. The leader continuously measures its own movements at regular sampling intervals. Each time a new set of measurements is ready, it broadcasts a message containing this and other data over the radio. The exact content of such a message is outlined in the next section. Both

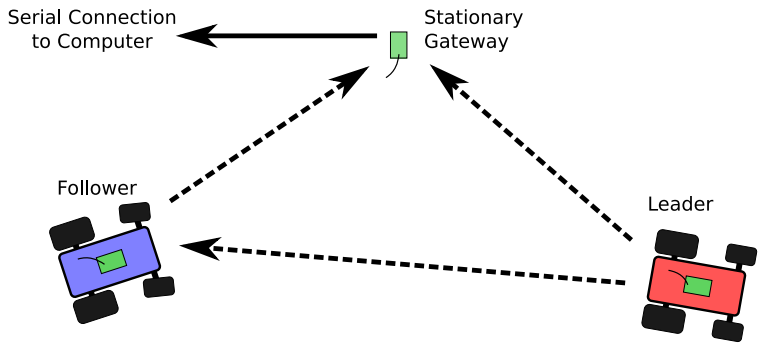


Figure 5.1: Protocol overview.

follower and gateway receive the message. The follower records the contained movement data for later use by its controller and schedules its own transmission. The gateway forwards the incoming message over the serial link to the computer for logging. After a short delay, the follower broadcasts its own message which is only received by the gateway. This message is processed identically by the gateway.

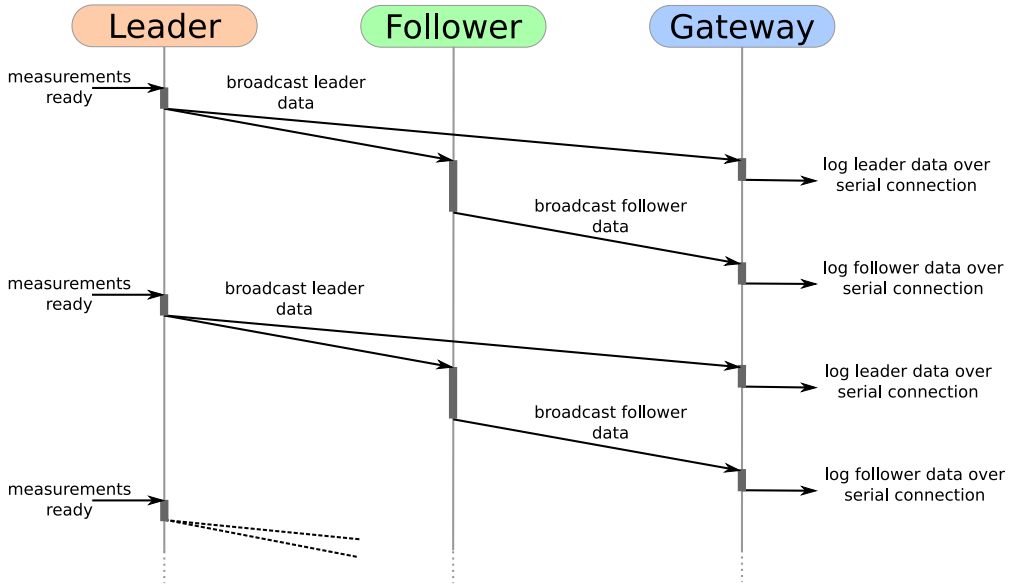


Figure 5.2: Protocol sequence diagram.

Note that the follower does not synchronize the execution of its controller with the reception of a message from the leader: the execution of the follower's controller is not triggered by the radio but rather by its own sensor sampling timer. This is not shown in the diagram of Figure 5.2 as this does not relate to the protocol. The received leader movement parameters are simply recorded as the new desired heading and velocity. Each time the follower obtains a new local velocity and heading measurement, it executes its controller. This ensures that the follower's controller is executed at regular intervals even in the presence of packet loss. In case of packet loss, it will try to maintain the leader's last recorded heading and velocity. At the leader node, the transmission of a message is triggered by the sampling timer. Consequently, a leader message is sent as soon as new data is available.

5.2 Message Content

The mobile nodes, i.e. the leader and follower, continuously broadcast messages. The most important data contained in these movement messages is the node's current heading measurement and velocity estimate. At the follower node, this data is used as input for the fuzzy controller along with the follower's own heading

measurements and velocity estimates (refer to Chapter 6 for a full explanation of the controller). The gateway logs movement messages from leader and follower and, using the logged movement messages, it is possible for us to see what perception the nodes had of their own movements.

In addition to this movement data, the follower includes the output from the controller in its outgoing messages. Thus, the gateway also logs the steering and throttle commands from the follower. If the follower is misbehaving during a test, this data is invaluable to trace the problem.

Additionally, a movement message from both mobile nodes contains the raw readings from the compass sensor and the partly integrated readings from accelerometer (refer to Section 4.1.1). This data is also logged by the gateway and can be used to recalibrate the sensors and for debugging. This way we can see whether oddities in the perceived heading or velocity were caused by the sensors or rather by the calculations performed during data processing.

A high level of packet loss can cause the follower to miss changes in the leader's movements. To be able to trace whether a failure in the follower's behavior is due to packet loss or another problem, it is important to know what movement parameters the follower is trying to match at any moment. This is why the movement parameters of the leader are included in the follower's outgoing messages. Consequently, the input values of the follower's controller are all logged, giving us a view of what the follower is trying to achieve in any point in time.

In summary, apart from the data necessary for the system's basic operation, a protocol message also contains trace data for performance evaluation and debug purposes. With all the given logged data it is possible to see what perception the nodes had of their own movements during a test. For the follower, all the input and output values of the controller are available, making controller issues much easier to trace. And the raw sensor data is logged for both nodes, making problems with data acquisition and data processing easy to find.

Table 5.1 gives an overview of the protocol message content. The total size of a single message cannot exceed 32 bytes due to radio hardware limitations (refer to Section 2.1). Thus, if more needs to be transmitted, a series of packets would need to be transmitted. This is something we try to avoid to keep the protocol simple. Luckily, we managed to achieve that for all the previously described data as shown in the table. There is even space left for a debug value.

Message Field	**Size (bytes)**	**Description**
pkt_seq	2	Packet sequence number
compass_x	2	Compass raw axes measurements
compass_y	2	
compass_z	2	
accelero_int_x	2	Accelerometer 10 sample partial axes integrations
accelero_int_y	2	
accelero_int_z	2	
steering	2	Follower steering and throttle controls
throttle	2	
heading	2	Measured and desired heading
des_heading	2	
velocity	4	Integrated and desired velocity
des_velocity	4	
debug	2	Free space in message (arbitrary debug value)
Total	32	

Table 5.1: The contents of a protocol message.

Table 5.1 also lists a packet sequence number. This is a value that is increased with the transmission of a new message. This field is not used by the nodes during protocol exchange and it is directly recorded by the logging computer. During data evaluation this field is used to detect packet loss. Also, the most significant bit of this field is used to distinguish between leader and follower messages. The leader has this bit set while the follower resets it.

Using the collected data, it is possible to run a simulation of the follower on the recorded data from the leader. If something goes wrong during the tests, a simulation may reproduce the problem. This technique

is described further in Chapter 7. We describe the recorded results in Chapter 9.

5.3 Summary

The follower node uses its own movement measurements and the measurements communicated form the leader to control its actuators in an effort to mimic the leader's movements. A gateway node receives messages from both leader and follower to keep a record of the vehicle's perceived movements and the activities of the follower controller for later evaluation. To facilitate the communication between the two mobile nodes and the stationary gateway we define a simple communication protocol in this chapter. The protocol has the following properties:

- One-way communication: The follower receives movement messages from the leader and the gateway receives messages from both mobile nodes. The leader never listens for any incoming messages and the follower only listens to messages from the leader. The gateway is strictly an observer and never sends any messages. This means that directly controlling the follower from the gateway is not possible.
- Transmission by the follower is synchronized to the leader: The follower sends a transmission of its own right after the leader has transmitted a message. As there are no other transmitting entities in our system, this means that the protocol is inherently synchronized and no collisions will occur. Note that on the follower the protocol is not synchronized to the execution of the controller or the sampling of the sensors
- Exchanged messages contain movement data, raw sensor readings and values useful for debugging: The most important content of the exchanged messages is the perceived heading and velocity of the transmitting node. Other than that, the raw sensor readings are included to trace problems in data processing and to calibrate the sensors. To be able to trace controller-related problems in the follower, a message from the follower contains all values used as input to the fuzzy controller. This also includes the movement data received from the leader.
- Messages contain sequence numbers to detect packet loss: During data evaluation, packet loss can be identified as missing packet sequence numbers. The messages from leader and follower are distinguished by differing packet sequence ranges.
- Protocol is inherently feasible for a large group of followers: When logging data from all nodes to the gateway is not considered, the leader is the only node transmitting in the network. This means that a large amount of nearby and strictly listening followers can be employed.

The messages recorded at the gateway are used for evaluation of test results (Chapter 9), validation of simulations (Section 9.3), compass sensor calibration (Section 4.2.2) and debugging (Section 8.1.2).

Chapter 6

Fuzzy Control

In the preceding chapters we outline how the leader and follower node can measure their movements and how the follower node can control its vehicle through its actuators. In this chapter we describe the implementation of the controller that forms the center of the follower's sense-and-react control loop.

We choose to use fuzzy logic as paradigm to build the controller. Unlike the more familiar Boolean logic, fuzzy logic provides the means to represent intermediate values between absolute true and absolute false. Analogously, unlike the crisp Boolean sets, fuzzy sets define a *degree of membership* for a particular item instead of an absolute membership or non-membership verdict. This way, fuzzy logic can be used to reason with the more human and fuzzy concepts, like 'rather small' and 'not so very good'.

The application of fuzzy logic to build a controller is commonly called fuzzy control. Fuzzy control is an attempt to make computers understand natural language and behave like a human operator [24]. These days, fuzzy controllers appear in every-day objects like washing machines (choosing the optimum amount of soap), video cameras (e.g. auto-focus) and cars (e.g. the Antilock Braking System). In industry, it is for example used for cement kilns, underground trains and robots. Other application fields include medicine and public transport.

First, in Section 6.1, we provide a brief introduction to fuzzy logic and motivate why we choose to use this control paradigm. Then, in Section 6.2, we define what the exact objective of the controller is, we provide an overview of how our controller fits in the sense-and-react control loop, and we show how it is to be executed on the node. Subsequently, in Section 6.3, we outline the overall structure of our fuzzy controller: we choose not to build a monolithic controller, instead we decompose the controller into a sub-controller responsible for maintaining the desired heading and another sub-controller to maintain the desired velocity. Before we describe the internals of our fuzzy heading and velocity controllers in Section 6.5, we provide a simple model of the behavior of the vehicle we need to control in Section 6.4. Similar to the decomposition of the controller in heading and velocity behavior, the definition of the vehicle model is divided into heading and velocity behavior. In Chapter 7, these models are used as a basis for defining simulations of the follower vehicle.

6.1 Background

In this Section we provide a brief outline of the basic fuzzy logic knowledge needed to understand the fuzzy controller we present in the subsequent sections. As fuzzy logic is a very broad technology and research topic, we do not attempt to provide an exhaustive survey of fuzzy logic. We only describe what is necessary for our controller design. For the readers interested in the details much literature is available on the subject (e.g. [24, 26, 37]).

6.1.1 Fuzzy Sets

Traditionally, computers use the Boolean two-valued logic that only allows for exact true or false assessments and yes or no decisions. Boolean logic is incapable of representing subjective ideas like 'very cool' or 'moderately slow'. In 1965 fuzzy logic was first coined by Lotfi A. Zadeh as a method to provide computers with the means distinguish between concepts that lie somewhere between true and false, much like human reasoning. Fuzzy logic can manage virtually any proposition expressed in natural language [6]. An important concept used in fuzzy logic is the linguistic variable, whose value is a word or a sentence in natural language. One may for example define a linguistic variable for the concept room temperature with the possible values 'very cold', 'chilly', 'just fine', 'warm' and 'very hot'.

In order to let a computer reason with such fuzzy concepts, we must clearly define what is 'chilly' and what is 'just fine' in this case. For traditional logic reasoning on what is 'chilly' for instance, a crisp set is defined whose temperature members are considered 'chilly'. Everything outside this set is not 'chilly'. This is a very abrupt approximation of the meaning of 'chilly' and this does not match well with our human concept of this term, which assigns (subjective) gradations to assess the 'chilliness' of the current room temperature. This is where fuzzy set theory comes to play. Unlike traditional sets, an item is not decidedly a member of a fuzzy set, but rather to some degree (usually defined in the range $[0; 1]$). The values of the linguistic variables used for fuzzy logic are all represented as fuzzy sets.

To assess to what degree a measured room temperature is a member of the fuzzy set 'chilly', we must provide a description of this set for all possible temperature values. This done using a so-called membership function for the 'chilly' fuzzy set. In most (if not all) literature a membership function for fuzzy set A on input value x is designated $\mu_A(x)$. The membership function $\mu_A(x)$ expresses to what degree value x is a member of fuzzy set A. The set of all possible values (set members) is called the *universe of discourse*, sometimes abbreviated to just *universe*, and designated $\mathcal{U}$, meaning that $x \in \mathcal{U}$.

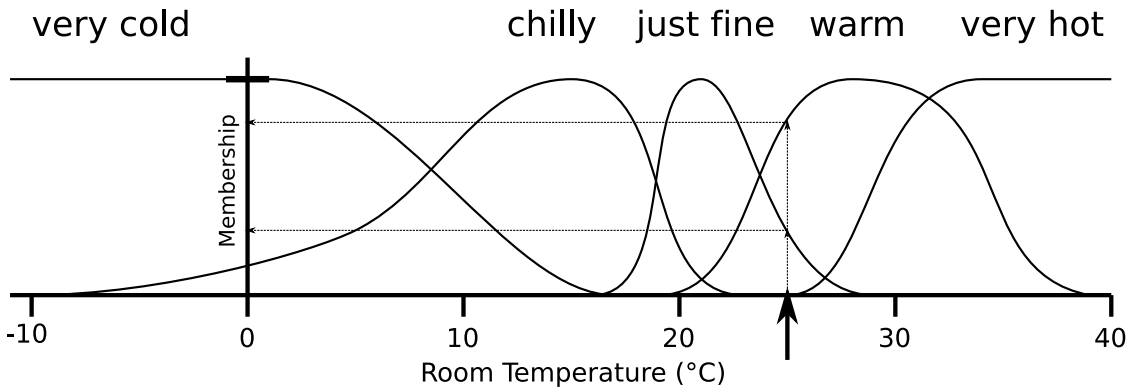


Figure 6.1: A graphical representation of the linguistic variable *Room Temperature* with membership functions for each possible value.

For better understanding, membership functions are often plotted to show how membership degree values are assigned to all possible values in the universe. We show this in Figure 6.1 for all possible values of our example room temperature linguistic variable. The plot shows how the membership functions for 'very cold', 'chilly', 'just fine', 'warm' and 'very hot' may look for a particular person in the universe between -10 and 40°C. Note that this is highly subjective and thus this would differ from person to person. This particular person is for instance not easily pleased with the room temperature, since the 'just fine' membership function is relatively narrow and higher temperatures have some preference since the slope of the 'just fine' plot is less steep on the high temperature side.

Figure 6.1 shows irregularly curved membership functions. Such curves can often capture the fuzzy linguistic notions best. However, for computers this is not a very efficient representation, as the membership functions become relatively complex. Especially for limited hardware, like our sensor nodes, implementing such membership functions is problematic. As a simplification, good performance can be achieved with piecewise linear representations like triangular and trapezoid membership functions.

6.1.2 Fuzzy Control

Now consider that we want to build a thermostat to control the room temperature of Figure 6.1 to reach a value that is considered 'just fine'. The thermostat controls the air conditioning system that can do one of three things: nothing, heat or cool. Assume that the air conditioner is controlled using a continuous signal ranging from -1 (cool) to 1 (heat). Using fuzzy logic we can build a controller that is specified using empirical linguistic IF-THEN rules. These rules can be entirely based on intuitive knowledge.

A human operator would intuitively set the air conditioning to 'heat' when the current temperature is either 'very cold' or 'chilly' and he would set it to 'cool' when the temperature is 'warm' or 'very hot'. A more skilled operator would even set a higher intensity for heating or cooling when it is 'very hot' or 'very cool' respectively. The experience of the operator is captured in the following linguistic IF-THEN rules:

1. If the temperature is very cool then control the air conditioning to heat at high intensity.
2. If the temperature is chilly then control the air conditioning to heat at medium intensity.
3. If the temperature is just fine then control the air conditioning to do nothing.
4. If the temperature is warm then control the air conditioning to cool at medium intensity.
5. If the temperature is very hot then control the air conditioning to cool at high intensity.

To build a fuzzy controller with these rules, we first need to capture the room temperature in the fuzzy sets defined by our linguistic input variable. This means that we map our input value to membership degrees in each of the specified membership functions. The process is called *fuzzification*. In Figure 6.1, this is indicated by the arrows for a temperature of 25° C. That temperature is a member of the 'warm' fuzzy input set and, to a lesser degree, of the 'just fine' set.

Because we build a fuzzy controller, the rules do not yield discrete conclusions that directly point to a single result. Instead, the rules yield new fuzzy sets for the possible values of the control output. The control output is another linguistic variable with in this case the possible values 'heat high', 'heat medium', 'idle', 'cool medium' and 'cool high'. The maximum membership value (scale) of a fuzzy set in a rule conclusion value depends on the firing strength of the rule, which is a fuzzy logical combination of the membership degrees of the values used in the premise of that rule.

With the simple rules stated above this means that firing strength of the rule is the same as the membership degree of the single value used in the premise of each rule. In the simplest case, the scale of the fuzzy set of the conclusion value is equal to the firing strength. So, if the current room temperature is 'just fine' to a degree of 0.1, the third rule will have a firing strength of 0.1 and yield a membership function scaled to 0.1 times its normal maximum value for the 'idle' output value. Certain types of fuzzy inference also apply scaling values to the firing strengths of all rules to give some rules greater impact on the control output.

After the presented rules are evaluated, we have separate conclusions for each individual rule. A conclusion is composed of scaled fuzzy sets for each possible output value. These fuzzy sets can have any curve shape imaginable, but it is convenient to choose a discrete singleton membership function, meaning that it has a membership value greater than zero only at a single position. A rule conclusion is the union of the scaled fuzzy sets. But, in the presented example, only one of the possible values for the linguistic output variable can have a non-empty fuzzy set for each rule, because each rule yields only one conclusion. This means that the conclusions are composed of only a single scaled output membership function.

For many control problems more than a single input is necessary or other properties of the input value are of interest as well, e.g. its current rate of change. Each input is represented as a linguistic variable that can be used in the controller's rules. In the rules stated above only one linguistic variable is used in the premise. If more are used, the firing strength of a rule must be a combination of the membership degrees of the values used in the premise. This is where fuzzy logic is applied as well. The logical combination is specified with *and* and *or* connectives. Depending on the controller's desired properties, various alternative methods exist to implement these connectives. The simplest ones are using the *min* and *max* functions respectively, meaning that *min* chooses the minimum of the two membership degrees and *max* chooses the maximum of the membership degrees. Alternatives involve multiplication, which is best to be avoided on the sensor nodes.

To obtain a crisp output value useful for our air conditioner, we need to combine the conclusions and choose

a control output. To achieve this, the conclusions are accumulated by calculating the union of the conclusion fuzzy sets. In fuzzy logic, there are various ways to calculate the union of fuzzy sets [24], but calculating the maximum is is the most straightforward and computationally simple method. This produces a single conclusion fuzzy set for all rules of the controller. The process of evaluating the rules and combining the conclusions is called *inference*.

Yet, this conclusion is still a fuzzy set. To obtain a valid single crisp output, a single value must be chosen that best matches the fuzzy set. Various methods exist to choose such a crisp value, but for its simplicity we use the *center of gravity (COG)* method. The process of extracting a crisp output value from the fuzzy set resulting from inference is called *defuzzification*. In Section 6.5.2 we explain this for our controller and we show how COG works.

In overview, a fuzzy controller executes three basic steps: *fuzzification*, *inference* and *defuzzification*. During fuzzification, the numeric input values are mapped to fuzzy sets by applying the membership functions. Based on the fuzzified inputs, the controller infers through its IF-THEN rule set and produces an aggregated fuzzy output. The final control action is derived by defuzzifying this aggregated fuzzy output.

6.1.3 Benefits of Fuzzy Control

Other methods to implement the controller are available. In this section we briefly outline why we choose to use fuzzy logic to build our controller. Fuzzy control has several properties that qualify it as an effective tool for WSAN problems:

- It can be implemented on limited hardware and is computationally fast: Depending on the used fuzzy operators and methods, the fuzzy set techniques have low information and time complexity [21]. Previous work [33] shows that fuzzy logic is feasible on limited hardware like our sensor nodes.
- It handles unreliable and imprecise information: Sensor data is usually imprecise an unreliable to a certain degree. Fuzzy logic offers a robust solution to decision fusion under uncertainty [43].
- Fuzzy-based methodology substantially reduces the design and development time in control systems: Fuzzy control is based on an inference process that evaluates set of rules on fuzzy input membership degrees. These rules are commonly based on intuitive knowledge. There is no need to define a complex mathematical model for the system. If a human operator can control the system, it is likely that a set of fuzzy rules that captures the relevant operator's knowledge can be defined that matches the operator's behavior. A downside in this respect is that a fuzzy controller is in general nonlinear, and therefore the design approach is commonly one of trail and error [24]. However, the effect of the inference rules is easier to grasp, making adjustments more intuitive than changing a complex mathematical model.
- Fuzzy controllers handle non-linear systems well: Most real-life physical systems are non-linear. Fuzzy controllers handle non-linear systems better when compared to conventional approaches [6]. For conventional controller design approaches several different approximation methods are used to handle non-linearity. Typical examples include linear, piecewise-linear and look-up table approximations to trade off complexity, cost, and system performance. These approximations tend to be tedious to implement, difficult to debug and tune and yield relatively poor performance. Fuzzy logic can inherently handle non-linearity with its rules and membership functions in an intuitive manner, without a large performance penalty.

6.2 Controller Overview

The objective of our controller is to control the follower vehicle such that it attains a heading and a velocity that match the respective values measured and reported by the leader. For this purpose, the controller can steer the vehicle using the front wheel steering to change its heading and it can apply throttle on the rear drive motor to accelerate or decelerate towards the desired velocity. Additionally, as described in Section 3.2, the follower vehicle has an inductive brake, which should be used to make the vehicle stop reliably. Summarizing, the controller must map the measured and desired movement input values to the

appropriate steering, throttle and brake control signals to make the follower synchronize to the leader's movements.

Figure 6.2 shows how our controller fits in the control loop. The sensors sense the current movements of the vehicle. The data processing stage (refer to Chapter 4) transforms the sensor data into useful heading and velocity estimates. The controller acts upon the local movement measurements and their respective desired values. It yields signals for the actuators by which it aims to achieve the desired movement. The desired movement is the data communicated from the leader. The actuators change the movement state of the vehicle and that response is picked up by the sensors (as indicated by the dashed arrow in the figure).

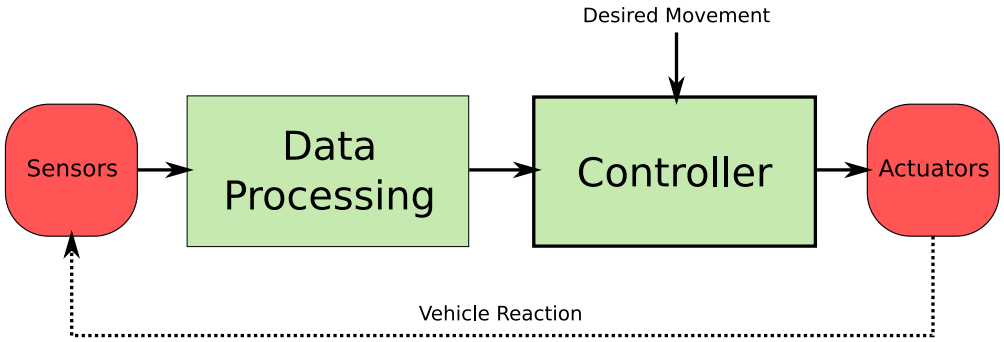


Figure 6.2: An overview of the controller's place in the system.

An important design choice is the frequency at which the controller is executed on the nodes. This control frequency directly influences the response time of the controller. In Section 4.2.1 we choose a lower angular sensitivity for the compass sensor with the primary reason that the sampling frequency otherwise becomes to low to keep up with the high steering rates of the vehicles. The same reasoning holds for the follower's controller. At the chosen sensitivity configuration, the compass sensor can be sampled at 66 Hz on two axes. It is however not feasible to schedule the controller at that frequency on the node. We choose to use a control frequency of 16 Hz, which is conveniently ten times smaller than the accelerometer sample frequency we chose in Section 4.3.2, meaning that exactly ten accelerometer samples are used to update the velocity estimate for the next controller execution. Similarly, the compass sensor is sampled at 16 Hz to provide a single fresh heading measurement for each controller execution. At 16 Hz, the controller's view of the current heading should lag behind only about 5° during fast steering operations.

6.3 Decomposition

It is always a good idea to keep solutions for wireless sensor nodes simple and lightweight, due to their obvious resource constraints. Fuzzy Logic control in its own right is a simple control solution, but much design freedom still exists regarding the type, inputs and outputs of the controller. Whether the controller is to be implemented as a monolithic block or rather as a composition of several sub-controllers is an important concern. If the controller is decomposed, the sub-controllers become simpler with fewer inputs and outputs, making them easier to understand, implement and tune. Such decomposition is not guaranteed to be possible however.

The existence of a valid decomposition depends on how the controlled environment responds to control signals, i.e. the controller's outputs. If any one of the control signals influences the environment such that all inputs of the controller can change, decomposition is not possible. If, on the other hand, the input values are partitioned such that each control signal will only influence a distinct set of input signals, decomposition is likely to be possible.

Controlling the heading and the velocity of a wheeled vehicle can be considered two separate control concerns. The current heading of the vehicle is only changed when the car is driving and the front wheels are not in line with the rear wheels, i.e. it is *steering*. The forward or backward velocity of the wheeled vehicle is changed by the current force applied on the rear axis by the drive motor, i.e. the *throttle*, and the friction the vehicle incurs while driving. If it is safe to assume that the angle of the front wheels will not influence

the speed of the vehicle and that the applied throttle will not change the heading of the vehicle, these two vehicle parameters can be controlled by two entirely separate controllers.

However, this assumption is not entirely justified. For example, preliminary tests show that the vehicle incurs more friction when it is steering (refer to Chapter 9 for examples). Consequently, when steering, it will gain less speed at the same throttle level when it tries to speed up and it will slow down faster. When the velocity controller does not take this influence into account, this can cause the follower node to fall behind when the leader speeds up in a corner at maximum throttle, even when the leader and follower nodes are identical. Also, the four-wheeled vehicles we use as leader and follower cannot change their heading when they are not driving. Thus, the rate at which the heading changes depends not only on the steering intensity but also partly on the vehicle's velocity.

It is still possible to decompose the vehicle controller into separate heading and velocity controllers, but their outputs need to be compensated to account for their interaction on the real vehicle. Alternatively, the sub-controllers can be changed with more aggressive parameters to make these match their desired values more quickly. For the throttle example this means that extra throttle is applied when the steering is active. However, as we explain in the rest of this chapter, the controller can yield an unstable response if it is tuned to be too aggressive.

Another option is to give the velocity controller an extra input that indicates the current absolute steering intensity. When incorporated in the fuzzy inference, this extra input can provide the means to define rules that adjust the outputs of the controller to compensate for the influence of the other controller. This, however, makes the controllers much more complex. In stead, we decide to match the control signals of the two controllers in a simple final post-processing stage.

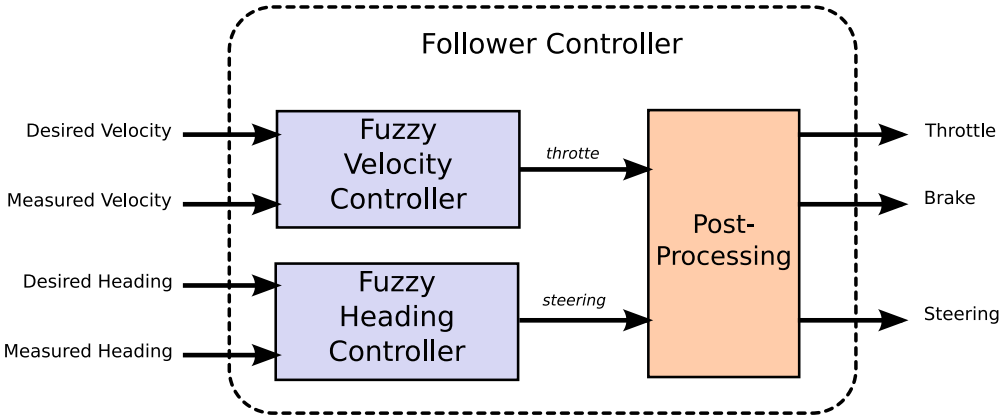


Figure 6.3: The composition of the follower controller.

Figure 6.3 shows the composition of the follower's controller. The inputs of the controller are the desired and measured movement parameters. The velocity controller uses the desired velocity and the measured velocity to produce a throttle control signal, whereas the heading controller uses the desired heading and the measured heading to produce a steering control signal. The post-processing stage performs the matching operations and it activates the brake when the vehicle needs to stop.

We describe this structure in more detail in Section 6.5 along with the internals of the individual controller components. First, we explain what we are trying to control by formulating a model for our follower vehicle.

6.4 Vehicle Model

In the previous section we discuss the decomposition of the follower's controller into two separate fuzzy controllers: one to control the vehicle's heading and another to control the vehicle's velocity. In this section, we present simple models for the follower vehicle that these two controllers act upon.

In this chapter, these models serve as a means to describe the dynamics of the follower vehicle in order to

clarify each controller's operation. In Chapter 7, these models are used to build simulations of the vehicle's movements with the implemented fuzzy controller. Note that the models as presented here explicitly omit the effects of noise. These noise influences are added in Chapter 7 to obtain more realistic simulation results.

The controllers influence the follower's movements through their output signals and feedback is provided by sensor measurements that measure the movement parameters that are of interest. Both controllers have an input that defines the desired movement state of the vehicle. For the heading controller, that value is the heading it must aim to match and for the velocity controller it is the velocity it must aim to achieve for the vehicle. As we describe in Chapter 5, this input data is continuously communicated from the leader to the follower.

6.4.1 Heading

The vehicle has a set of front wheels of which the angle relative to the vehicle itself can be changed. This angle is assumed to be proportional to the steering control signal s. The value of s is defined to lie in the interval $[-1 \ldots 1]$, for which -1 is maximum left and 1 is maximum right steering. The current steering angle δ is assumed to be equal to the steering control s times the maximum steering angle δ_{max}. This means that δ_{max} directly defines how fast the vehicle can steer. The current heading ϕ of the car is updated using the current steering angle δ. This behavior is summarized in the following simple state formulae:

$$\begin{aligned} \phi' &= \phi + \delta \\ \delta &= -s\delta_{max} \end{aligned} \tag{6.1}$$

s	current steering control value, $s \in [-1 \ldots 1]$
ϕ	heading of vehicle
δ	current steering angle; the change in heading per unit of time
δ_{max}	maximum steering angle

Arguably, the model as presented here is over-simplified. The car will not change its heading when it is not driving, regardless of the angle of the front wheels. This alone is a hint towards the fact that the maximum steering angle δ_{max} depends on the current velocity of the vehicle. But for simplicity, we assume the steering angle to be proportional to the steering control value s.

6.4.2 Velocity

The modeling of the velocity behavior of a vehicle is more complex, because it needs to take the effects of friction into account. If a realistic vehicle is in motion and no external force is applied, it will slow down until it eventually stops. This is caused by friction inhibiting the movement of the vehicle. Consequently, less force is needed to make the vehicle slow down than to make it speed up by an equal amount. Therefore, unlike heading control, the velocity control problem is not symmetric.

For example, if a velocity controller that is ignorant of the effect of friction is trying to stop a moving vehicle, it will apply too much negative throttle to slow down. This would result in the car starting to drive backwards in stead of stopping. Because the velocity controller aims for a desired velocity of zero, it will then apply positive throttle to compensate for the resulting negative velocity. Again, too much throttle is applied in this compensation, resulting in the vehicle driving forward instead. Clearly, this control loop would be highly unstable: the car would never stop, but rather accelerate forward and backward all the time[1].

The dynamics of the vehicle's velocity are modeled as follows:

$$\begin{aligned} v' &= v + a_{eff} \\ a_{eff} &= \begin{cases} a_t - a_f & \text{if } v + a_t >= 0 \\ a_t + a_f & \text{else} \end{cases} \\ a_t &= a_{t,max} t \end{aligned} \tag{6.2}$$

[1] We tried this naive implementation on the actual vehicle and this behavior presented as expected.

with:	t	current throttle control value, $t \in [-1 \dots 1]$
	v	velocity of vehicle
	a_{eff}	effective acceleration
	a_t	throttle acceleration; the acceleration resulting from the applied throttle
	$\hat{a_t}$	maximum throttle acceleration
	a_f	frictional acceleration; the 'acceleration' resulting from friction ($a_f >= 0$ and $a_f <= \|v + a_t\|$)

The current velocity v changes by an amount of a_{eff}; the effective acceleration. The effective acceleration is a result of the net force that is applied to the vehicle. If there were no friction in the system, the net force would be equal to the force applied to the vehicle by its engine. In reality the engine's force needs to overcome a certain amount of friction, commonly represented as a frictional force.

Since, according to Newton's second law of motion, the acceleration of an object is equal to the ratio of net force and the object's mass ($a = \frac{F_{net}}{m}$), we can express our equations with accelerations in stead of forces if the mass of the system is not considered. Therefore, in Equation 6.3 the effective acceleration a_{eff} is represented as the difference between the acceleration that results from the vehicle's engine a_t and the frictional acceleration a_f.

For this model, the frictional acceleration is a positive value that always opposes the (potential) motion of the system, so if the velocity of the vehicle is negative, the frictional acceleration works positive on the effective acceleration and negative otherwise. Note that friction only opposes the motion of the vehicle; if no other forces are applied to it, the frictional force will only have effect up until the moment that all motion of the vehicle has ceased: this means that $a_f <= |v + a_t|$.

The frictional force consists of multiple components; part is fixed and part depends on other factors like the current velocity of the vehicle. The fixed component is the minimum force needed to start moving, the velocity dependant part is what limits the vehicle's maximum velocity. The effect of friction can be very complex. We provide a simple model of the frictional acceleration a_f in Section 7.2 where we describe the vehicle velocity simulation.

The value a_t is assumed to be directly and proportionally controlled by the velocity controller through throttle control signal t. If t set to its maximum or minimum value, a_t will equal $a_{t,max}$ or $-a_{t,max}$ respectively. Note that the actual value of $a_{t,max}$ depends on the battery level. For example, if the battery is mostly drained, $a_{t,max}$ will be too low to overcome the frictional acceleration a_f and the vehicle is unable to move.

6.4.3 The Brake

Thus far, we discussed slowing down only as the explicit application of acceleration opposite to the current movement using the vehicle's electric engine, i.e. negative throttle. This technique can stop the vehicle very quickly. However, the problem with this approach is that it can be very unstable: if too much negative throttle is applied for too long, the vehicle will accelerate backwards. Consequently, a relatively high level of accuracy is required from the fuzzy controller. As we show above, the effects of friction further complicate this issue.

To give the controller a wider margin for error, we could stop applying negative throttle when the velocity of the vehicle comes too close to zero. From that moment onwards, the friction does the rest to stop the vehicle. However, this level of friction is relatively low and therefore the vehicle can coast further for a short distance. It is important to have the means to stop the follower abruptly, because the leader can do so as well.

It is very logical to think of using a brake to stop a vehicle, because there is no risk of ending up driving backwards in stead of stopping. Unfortunately, the toy car we use as follower does not have an actual mechanical brake as one would find on a normal car. In Section 3.2 and Appendix A, we describe the use of an inductive brake for our follower car. This braking technique is less effective than the mechanical alternative and therefore we apply the brake only after the velocity of the vehicle has decreased sufficiently through applying negative throttle.

In the model of Equation 6.3 the activation of the brake increases the frictional acceleration a_f. The effect of the brake is modeled further in Section 7.2, where we provide a model of the frictional acceleration. In

this chapter, we are more interested in the fact that the velocity controller has an additional output that activates the brake.

6.4.4 Influence of Steering on Friction

Exploratory experiments with the actual follower car show that the friction significantly increases when the follower is steering. This effect is particularly evident when its battery is low: then it just barely gets through the corners, but manages to drive normal on straight lines.

This effect is much less prominent on the leader vehicle. This is probably due to the fact that, unlike the leader vehicle, the follower has no differential for the rear wheels. This means that the rear wheels cannot spin at different speeds, which is required if a car is to take corners smoothly. As long as the follower takes very wide corners, this is not much of a problem. But, for very quick heading changes, the follower has to apply much more torque on the rear axle, because otherwise it would fall behind.

If the follower's velocity controller does not take this effect into account, the follower is prone to fall behind in the corners. This is especially true if the follower's battery is low. Also, when the follower is pointing in a different direction than the leader and it first starts driving it should not directly apply steering, as it will take much longer to gain speed. It may even incur so much friction that it will not be able to drive away.

In Section 7.2.1 we explain this issue in more detail, as that is where we provide a simple model for the friction the car incurs during driving. We outline how we deal with this effect in Section 6.5.5.

6.5 The Controller

In the previous section we decompose the controller into separate heading and velocity sub-controllers. Fuzzy control is applied for both these controller components. As described earlier Section 6.1, a fuzzy controller consists of three stages: fuzzification, fuzzy inference and defuzzification. In the fuzzification stage, the membership grade of the input values to the controller's fuzzy input sets is determined using fuzzy membership functions. During fuzzy inference, IF-THEN rules are applied on the resulting membership grade values, i.e. the fuzzy input variables. The fuzzy inference yields an output fuzzy set. The defuzzification stage extracts a crisp output value from this fuzzy set.

The sub-controllers, for heading and velocity respectively, are very similar in terms of their goal: achieve desired value for a single movement parameter of the vehicle. That is why we make these controllers structurally identical. The only differences lie in the range of the input values, the dimensions of the input and out membership functions and post-processing. The shape of the input and output membership functions, the fuzzy inference technique and the fuzzy inference rules are identical for both controllers.

Also, the type of fuzzy controller we describe here is by itself not innovative because it is well-documented and used in various earlier books and documents [14, 24, 37]. The innovation lies in the use of this type of controller on a wireless sensor node for distributed coordination. The simplicity of the type of fuzzy control we employ makes the resource requirements relatively small, making this technique appropriate for use on our sensor nodes.

6.5.1 Fuzzy Input

The goal common to both controllers, i.e. achieving a desired value for their movement parameter, can directly be translated into minimizing the error between the actual value of the movement parameter and the desired value. It is a logical and very common choice to use this error as input to the fuzzy controller. Then, the fuzzy inference rules are specified such that if the error is positive a negative control output is applied and if the error is negative a positive control value is applied. If the error reaches zero, the control output is close zero as well, indicating that the controller has achieved its goal.

Although this sounds like a straightforward and valid solution, it is bound to be unstable. The environment of the controller, in this case the follower vehicle, will not instantly respond to its control signals. Consequently, there is a delay between the controller's actions and the emergence of a noticeable result on the controller's sensory inputs. If this delay is not taken into account, a nonzero control signal is applied for as long as an error is reported on the controller's input. This means that the controller will direct the controlled environment parameter past the desired value. When this is finally noticed on the controller's input, the controller applies inverse control to counter the now inverse error. Again, this effort fails in the same manner and the measured value overshoots the desired one again. This process results in an oscillation on the movement parameter at a frequency and amplitude that depends on the response time of the environment.

The common solution to this problem is to incorporate the change in error into the fuzzy inference as well. The general idea is that if the absolute error is decreasing already, the controller needs to be less aggressive to attain the desired state of the environment. If tuned correctly, this stabilizes the controller when the controlled environment parameter gets closer to the desired value. Thus, an extra input is needed for the fuzzy controllers and consequently the fuzzy inference rules involve up to two premise conditions. The inputs for both fuzzy controllers at a particular time instant t are thus defined as follows:

$$\begin{array}{lcll} e(t) & = & x_{desired}(t) - x_{measured}(t) & \text{(absolute error input)} \\ \Delta\, e(t) & = & e(t) - e(t-1) & \text{(differential error input)} \end{array} \tag{6.3}$$

where

e	Error between desired and measured value
t	Time
$x_{desired}$	Desired value
$x_{measured}$	Measured value

For fuzzification, input membership functions need to be specified. Those functions yield membership grade values that indicate to what level the input value is a member the given fuzzy set. Both our controllers have two inputs and both these inputs represent an error value. For the controller's inference rules it is useful to know whether that error is close to zero, positive or negative respectively. If for example only one input were used, the fuzzy rule set would only need three rules that assign proportional zero, negative and positive output set membership values to input values that are in the zero, positive and negative sets respectively. The fuzzy inference for the controller we describe here is more complex as we show in the next section.

For sufficient control granularity, we define five fuzzy sets for both inputs. The two extra fuzzy sets are used to indicate how positive or negative the error at the input is, meaning that the fuzzy controller can provide more fine-grained control. In accordance to common naming conventions for fuzzy controllers that are very similar to the one we describe here [14, 24, 37], we name the input (and output) fuzzy sets as follows:

NM	Negative Medium
NS	Negative Small
ZE	Zero Equal
PS	Positive Small
PM	Positive Medium

Table 6.1: The input fuzzy sets for our fuzzy controllers.

Although several alternatives for the definition of the fuzzy input membership functions exist, we choose to use the simplest alternative. These are the triangular input membership functions as shown in Figure 6.4. As we describe at the beginning of this chapter, the fuzzy input membership functions map the crisp input values to a set of membership grades which indicate to what extent the input value is a member of each of the fuzzy sets.

In our case the input value is an error value; either $e(t)$ or $\Delta\, e(t)$. In Figure 6.4 an example error value is shown to fall within the *ZE* and *PS* fuzzy input sets. As the error value lies closer to the center of the *PS* set, its membership grade is greater for that set than the value for the *ZE* set. What makes the triangular input membership functions a good choice for our controller is that the membership grades are easily calculated from the crisp input values. As Figure 6.4 shows it is just a matter of determining the

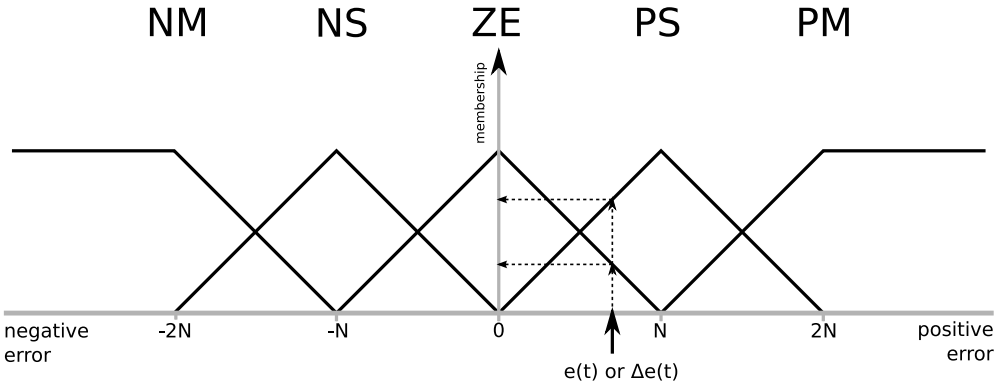


Figure 6.4: The fuzzy input membership functions.

value of the membership function at the error value , which is a linear operation for a triangular membership function. The membership value of fuzzy set PS with input e is for example calculated as follows:

$$\mu_{PS}(e) = \begin{cases} e\frac{M}{N} & \text{if } e > 0 \text{ and } e < N, \\ M - (e - N)\frac{M}{N} & \text{if } e > N \text{ and } e < 2N, \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

where: e the input error value
N distance between the triangular membership functions
M the maximum membership grade value; the height of the membership functions

The value M is the maximum value of the membership functions and N specifies the distance between the membership functions as shown in Figure 6.4. In literature on fuzzy logic the value M is usually defined to be 1, making the membership grade a value in the closed interval $[0, 1]$. However, as we describe in Section 8.2.3, the CPU of our sensor nodes has no support for floating point operations, so we are forced to use fixed point calculations. This in effect means that we are using plain integers to represent fractional numbers. Therefore the maximum value of the input membership functions is set to an integer value high enough to provide sufficient resolution for the membership grade values.

The calculation for the PS membership function in Equation 6.4 can be further optimized. As it is represented in the equation it involves multiplications. This can be avoided by choosing $M = N$, making the slope of the membership functions equal to 1. This way, the $\frac{M}{N}$ scaling factor is removed from the membership equations. This means that only addition and subtraction operations are necessary to calculate the membership grade for any of our input membership functions. On the sensor nodes, it makes a significant difference to avoid the use of multiplications, as adding such operations will significantly increase the processing time involved.

Because our fuzzy membership functions are equally spaced at a distance of N, all have a height equal to N and all have equal slopes as displayed in Figure 6.4, they can be fully described by the N value. This value thus determines the spacing between the membership functions, their maximum value and their width. This way, the N value directly influences the aggressiveness of the controller, i.e. what margin of error it tries to achieve. If the N value is smaller, this means the membership functions are less wide and closer together. A smaller width for a membership function means that a smaller range of input values is a member of that set at a membership grade > 0. If the ZE function is less wide, it means that a smaller range of the input error values is 'close enough' to zero, meaning that the controller will set an output control signal $\neq 0$ for smaller error values. A similar explanation is valid for the other membership functions with the addition that the functions are located at a shorter distance from zero when N is smaller.

In the rest of this thesis, the N value is referred to as *(half) the width of the input membership functions*, as this is exactly what N determines. Both the absolute and differential error inputs of our controller are fuzzified using the membership functions described here. However, the N value can differ, meaning that the sensitivity of the controller towards absolute error changes could be set to differ from the sensitivity to changes in error. Notably, since the N value also determines the height of the membership functions and no

further scaling is applied, setting distinct N values for the fuzzy inputs has further influence on the fuzzy inference we describe in the next section. This effect counters the positive effect that the increased N value has on the aggressiveness of the controller.

Of course, the fuzzy input membership functions must be defined for all valid input measurements. As the individual fuzzy controllers that we build here act upon very different input values, the input ranges for which their membership functions are defined differs as well. Sections 6.5.4 and 6.5.5 describe the exact input ranges for the heading controller and the velocity controller respectively.

6.5.2 Fuzzy Output

The fuzzy inference yields a conclusion in the form of a fuzzy set. This fuzzy set is combination of the conclusions of all inference rules. With respect to fuzzy inference, we choose to use the Mamdani [32] type fuzzy control. This means that the IF-THEN rules of the fuzzy inference are executed with a simple *min* function for *and* relations and a *max* function for *or* relations. Furthermore, defuzzification is performed using the simple centre of gravity (COG) method. In the next section we explain how the inference is performed in more detail. In this section we explain what the membership function of the output fuzzy set looks like and how it is defuzzified into a single crisp output control signal.

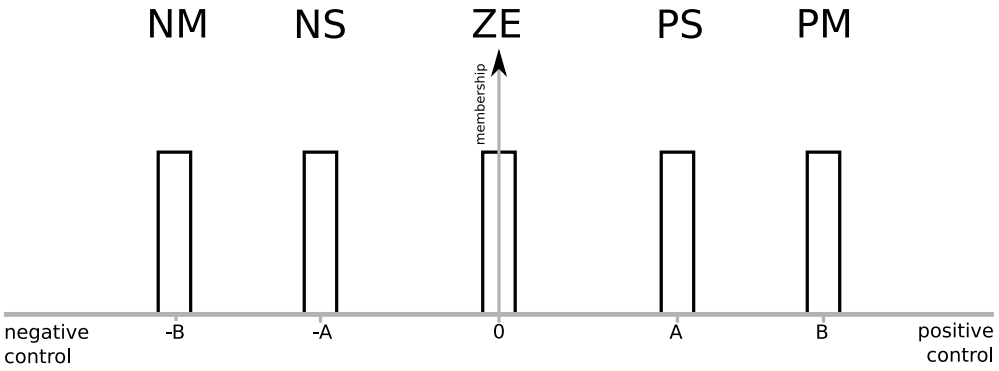


Figure 6.5: The output membership functions.

The fuzzy input membership functions we describe in the previous section are continuous, i.e. they are defined for any and all values in the input signal range. In contrast, the output membership function is discrete. In our case there exist five discrete values, each defined for only one output control value as shown in Figure 6.5. The choice for a discrete output function is part of the Mamdani-type fuzzy inference. This makes the defuzzification using the COG technique particularly simple as only five discrete values need to be combined. The COG technique is called centre of gravity for singletons (COGS) if the fuzzy set is discrete. COGS is commonly defined mathematically as follows:

$$u_{COGS} = \frac{\sum_i \mu_c(x_i) x_i}{\sum_i \mu_c(x_i)} \tag{6.5}$$

The x_i is a point in the universe of discourse for control output, meaning that it is one of the possible output values defined by the output membership function. The $\mu_c(x_i)$ is the output membership value assigned by the fuzzy inference for the conclusion x_i. Any x_i that is not member of the output set cannot be a conclusion from the fuzzy inference, meaning that $x = \{-B, -A, 0, A, B\}$. The members of this set are the positions of the discrete output membership function where the degree of membership is larger than zero, as shown in Figure 6.5. For a symmetric control response, the negative and positive membership functions need to be corresponding mirror images as shown in the figure. The COGS technique described by Equation 6.5 can be explained as taking the average of the discrete output control values weighted by the corresponding membership grade values resulting from the fuzzy inference.

The crisp defuzzified value u_{COGS} is the output of the controller. Equation 6.5 shows that this output value

does not only depend on the result of the fuzzy inference, but also on the structure of the fuzzy output membership function, in particular on the position of the output values on the output axis. Each discrete output value corresponds to a positive or negative control output intensity. The actual intensity for each of these possible inference conclusions can be changed by changing its position on the output universe. If for example the *PS* output is located closer to the maximum control value, the controller will become more aggressive because the rules that have *PS* as conclusion will contribute to a higher control intensity. Usually, the medium outputs (*NM* and *PM*) are positioned on the maximum output control values and *ZE* is located at control output zero. This leaves only the small outputs (*NS* and *PS*) to change the aggressiveness of the controller.

We use this method to change the controller's aggressiveness alongside the technique we describe in the previous section involving tuning the width of the input membership functions. However, changing the position of the output membership functions will not change the controller' behavior in the same manner. The width of the input membership functions determines the controller's sensitivity to changes, whereas the position of the output values determines the relation between the inference conclusions and the crisp controller output.

6.5.3 Fuzzy Inference

We use the Mamdani [32] type fuzzy inference for its relatively small calculation requirement. It only uses *min* and *max* functions to draw conclusions of from each inference rule of the controller. Also, all the conclusions from the inference rules are combined using a simple *max* function. Alternatives involve using multiplications in the fuzzy inference, which, as we describe in the previous section, is best to be avoided on sensor nodes.

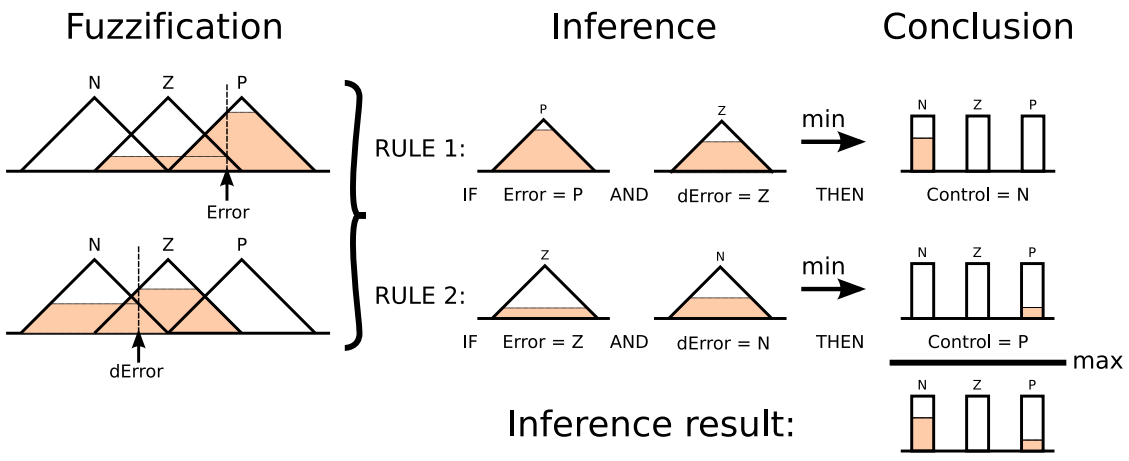


Figure 6.6: An example of Mamdani fuzzy inference.

Figure 6.6 shows an example of the Mamdani fuzzy inference. This simple example controller has two inputs (*Error* and *dError*) and one control output (*Control*). For all inputs and outputs the membership functions are named identically: N (negative), Z (zero) an P (positive). The left side of the figure shows the fuzzification that we explain in Section 6.5.1. As shown in the figure, the controller has only two inference rules. The rules use the fuzzy input sets to yield conclusions. The first rule produces a *Control* output that is a member of output set N if the *Error* input is a member of set P and *dError* is a member of set Z. The scaling of the *Control* membership function N is determined by the firing strength of the rules, calculated as the minimum (*and*) of the membership degrees of the *Error* input in set P and the *dError* input in set Z. The figure shows the membership degree of input *dError* to be smaller, meaning that that determines the result. The same is done for the second rule. The conclusion fuzzy sets of all rules are combined using a max function, meaning that for the output fuzzy set the maximum membership over all conclusions is used. The inference result is defuzzified into a crisp output control value as we explain in Section 6.5.2. In the presented case, the crisp output control value is found somewhere just below zero.

As we explain in Section 6.1, to calculate the result of an *and* connective the *min* function is applied on the membership grades of the referenced fuzzy input sets. An *or* connective would be performed using the *max*

e \ Δe	**NM**	**NS**	**ZE**	**PS**	**PM**
NM	PM	PM	PM	PS	ZE
NS	PM	PM	PS	ZE	NS
ZE	PM	PS	ZE	NS	NM
PS	PS	ZE	NS	NM	NM
PM	ZE	NS	NM	NM	NM

Table 6.2: Fuzzy inference rules.

6.5.1

function. However it is common to split rules that contain *or* connectives into multiple sub-rules that only involve *and* connectives.

The conclusions of Figure 6.6 are also combined using the *max* function. This means that the maximum conclusion over all rules is used as membership grade result for each output fuzzy set. A disadvantage of this approach is that it is not possible to define rules that have a negative effect on certain possible outputs. It is thus not possible to define rules that inhibit the result of other rules. Such rules could for example provide a solution to make the velocity controller respond well to the asymmetric nature of velocity control. This could then be achieved by adding an input that that indicates the sign, i.e. forward or backward, of the current velocity in order to adjust the intensity of forward and backward throttle control accordingly.

It is important that the rule set of a fuzzy controller yields conclusions for any set of input values for the specified input range. Otherwise, the output of the controller will be undefined. The compact tabular representation of inference rules shows the completeness of the rule set very clearly. This is shown in Table 6.2 for the fuzzy inference rules common to both of our controllers. The possible error (e) fuzzy sets are represented on the rows of the table and the possible differential error ($\Delta\, e$) fuzzy sets on the columns. The table cells each represent an inference rule with an *and* connective on its row and column set. In [14] the same rule set is used for DC motor control.

The table also clearly shows the symmetry of the rule set. Intuitively, the control output has lower intensity when both error and change of error are close to zero or when they are in the same order of magnitude with opposite sign. This means that when there is an error and the error is changing towards zero already, the applied control intensity is less to account for the fact that the system state is already changing towards the desired state. This rule set should yield a controller that quickly stabilizes towards the desired value without overshoot.

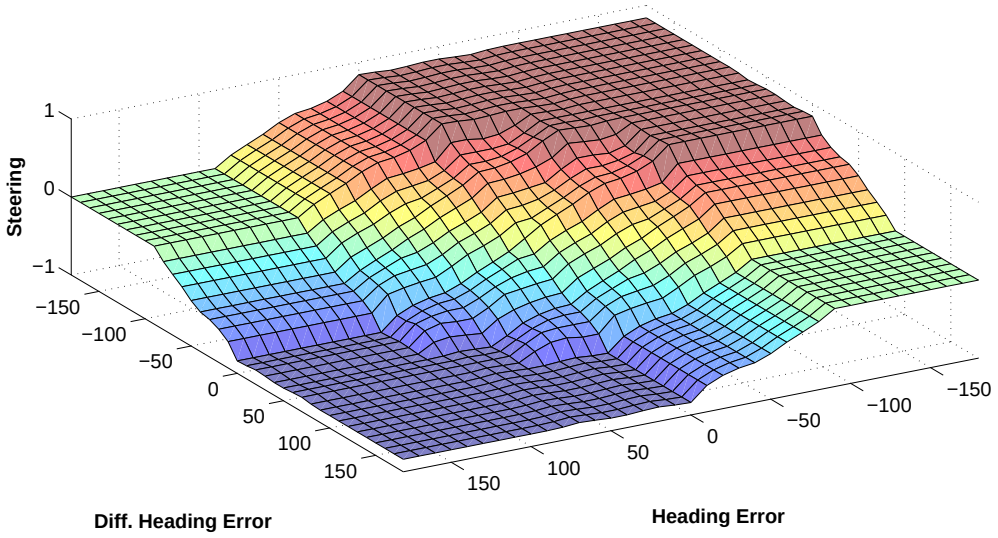


Figure 6.7: Surface plot of the fuzzy controller's response.

Despite its name, a fuzzy controller yields a deterministic crisp output value for any given set of crisp input values. This shown graphically for our fuzzy heading controller in Figure 6.7. For every set of input

values there is only one possible steering control output. The figure shows the region in input space where the controller applies no steering as two square surfaces connected through a line. On the square parts of this region, both error and differential error are large with opposite sign, meaning that the currently observed heading is bad, but quickly getting better. That is why the controller does not need to further amplify the rate of change by actively steering. On the line connecting the regions, the error and differential error are equal and opposite, meaning that the heading is expected to match the desired value in the next controller execution. Applying non-zero steering would in this case control the heading beyond the desired value, overshooting the desired value and thus destabilizing the controller response. The controller applies maximum (positive or negative) steering when both error and differential error are large with equal sign, meaning that the currently observed heading is very bad and getting worse. This warrants aggressive action to ultimately attain the desired value.

6.5.4 Heading Controller

The model we use to describe the dynamics of the vehicle's heading is very simple as shown in Section 6.4. We currently assume that the output of the controller is proportional to the actual in the change vehicle's heading per unit of time. Therefore, the controller we describe in the previous section does not need changes specific to heading control.

The heading of the vehicle is defined to lie in the interval $[-180 \ldots 180]$ degrees. The derived error inputs e and Δe for the fuzzy controller are defined within that same range. The difference calculations of Equation 6.4 need to take this into account by circularly mapping error values that fall outside this range back into this range by applying a simple modulus function. This way, the difference between a desired heading of 130° and a measured heading of -140° yields an error input value of 90° .

As we show in the previous section, there are several methods to change the aggressiveness of the controller. For the heading controller this tunable property of the controller is subject to an important trade-off. If the controller is not aggressive enough, it will not take action until the heading of the vehicle deviates significantly from the desired value. This can take a relatively long time and therefore it will cause a low performance for the follower if it travels great distances at this erroneous heading. If, on the other hand, it is tuned to be too aggressive it will respond to the noise in the measured heading. It will also not account for the delay between a change in the output control signal and a corresponding change in the sensor measurement. These two factors can cause the vehicle to sway heavily.

This means that an optimum value exists somewhere between these extreme situations. That value is not uniformly valid for a particular controller. It depends on the magnitude of the noise that the controller gets on its inputs and how accurate and fast the vehicle's steering is. This configuration parameter can be determined using extensive testing, but it can also be estimated using a valid simulation. We show how such simulation is performed in Section 7.1. That is also where we show how this trade-off works in more detail.

Our steering model of Section 6.4 assumes that the response of the vehicle's steering to the steering control value is linear. In reality, this will not be the case. To make things worse, our follower vehicle does not use a real servo for the steering, but rather a normal motor that deflects the steering from the center position. The steering is centered using a weak spring. This means that the steering mainly responds to the force applied by the motor and not its axle rotation. Therefore, very low intensity steering has no effect and a relatively large range of the highest values yields full steering. To match linear steering more closely, the raw steering output from the fuzzy controller is transformed to fall in the active range of the follower's steering control.

6.5.5 Velocity Controller

While the inputs for the heading controller are only defined within a circular space of $[-180; 180]$ degrees, the velocity controller can accept potentially infinite velocity error values. The actual range is determined by the maximum integer value the controller's platform can represent. On the MSP430, we use a 32 bit integer to represent the vehicle's velocity. The inputs of the controller are saturated in that range. Of course, only a relatively small portion of this velocity range is valid for our vehicle.

The model we define in Section 6.4 for the vehicle's velocity behavior shows the asymmetric effect of the

throttle on the vehicle's velocity. It indicates that the controller needs to apply less throttle when it wants to slow down and more when it wants to speed up. We decide not to complicate the fuzzy logic inference and therefore we maintain the simple two-input structure that we outline in the previous section. To compensate for the asymmetric velocity behavior, we subtract a constant from the fuzzy controller's throttle output value when its sign is opposite to the current velocity: if the sign of throttle and velocity is opposite, the vehicle is trying to slow down. If the current throttle value is lower than this constant, the output is zero. When the vehicle is trying to accelerate, the throttle control output remains unchanged.

This simple adaptation of the output assumes the influence of friction to be constant: when trying to decelerate, the controller assumes a constant portion of the desired deceleration to be handled by the frictional force already and subtracts that portion from its own output value. In Chapter 7, we show that this is not true when the vehicle is moving and steering. However, the dependence of the friction on the movement is relatively small when compared to its constant portion.

Note that subtracting a constant from the throttle control value reduces the maximum deceleration the controller can achieve, as the maximum control value is never reached. A more complex fuzzy controller with an additional velocity input could account for the effects of friction and apply maximum deceleration when necessary. However, for this work, we feel that the merit of being able to slow down faster does not outweigh the increase in the complexity of the fuzzy controller.

For either solution, the errors that the integration described in Section 4.1 can cause will have an effect on the compensation. If the velocity is estimated to be positive while in fact it is negative, the compensation is erroneously used in an inverse manner, amplifying the control instability.

In addition to subtracting the constant from the throttle output, the throttle is scaled down with a time dependent coefficient when the leader has stopped. This makes sure that the follower will stop eventually. If the follower has accumulated a significant error in the velocity integration it will not be able to attain a zero velocity anymore, because stopping will yield a velocity value other than zero. To prevent the follower from accelerating indefinitely in this case, it is only allowed to perform the actions necessary to stop within a limited time interval. Close to the end of this interval the brake of the vehicle is activated as we describe in the next section.

6.5.6 Post-processing

As we indicate in Section 6.3, the heading and velocity controller influence each other. In this section we describe how most of these influences are resolved. Additionally, we describe when and how the follower's brake is activated.

Backwards Steering

The simplest influence is not described in Section 6.3. The heading controller described in Section 6.5.4 assumes that the vehicle always drives forward. When the vehicle has a backwards velocity, those steering commands would be completely inappropriate: it would make the vehicle change its heading in the opposite direction. This way, it ends up driving circles while driving backwards. The obvious solution to this problem is to invert the steering commands when the velocity is negative.

Unfortunately, this means that errors in the velocity integration can also have an influence on the steering when the vehicle is driving at low velocity. Just like the throttle compensation described for the velocity controller in Section 6.5.5, a velocity error can cause the steering to be erroneously inverted. This can happen at low velocities, since that is where a relatively small error can cause the velocity estimate to cross the zero boundary.

Steering Friction

The effect of steering on the friction of the follower vehicle is not very significant if the battery of the follower vehicle is fully charged. It becomes particularly evident when the leader is driving too fast for the follower

to handle. In both cases, the controller will apply full throttle already without any compensation, so with respect to the velocity controller no improvements are available.

One could decide not to perform large steering operations when the follower needs to accelerate significantly, e.g. by making the maximum steering control value dependent on the current velocity error. This is not a real solution, since it will significantly reduce the follower's steering agility when the leader accelerates quickly. This solution just moves the problem from the velocity controller to the heading controller, which is not going to help. That is why we decide not to perform any compensation for this effect. When we use fresh batteries and when we do not accelerate too quickly with the leader vehicle, the follower should still be able to perform adequately.

Activating the Brake

When the velocity of the vehicle is close to zero, applying throttle is risky as the vehicle may end up driving in the opposite direction in stead of stopping. This is particularly true for the situations where the velocity estimates of the vehicles are offset from the real velocities, making the controller act upon erroneous information. This is when a true brake is useful. However, the main problem with the brake facility on our follower vehicle is that is not as effective as applying throttle to counter the current velocity.

Our solution is to activate the brake when the leader is standing still and the throttle value from the velocity controller is lower than a certain threshold. This way, the vehicle is initially slowed down using opposite throttle and, when the velocity decreases, the velocity controller's throttle output will decrease as well. When the throttle value drops below the threshold, the brake finishes the deceleration to a full stop.

The brake needs to be released when the vehicle has stopped successfully: it is energy-wise not a good idea to keep the brake active at all times when the vehicle is standing still. The motor controller that physically activates the brake will draw a significant amount of current to activate the transistors that short-circuit the drive motor[2]. Our solution is to employ a maximum brake interval after which the brake is deactivated irrespective of whether the vehicle has actually stopped; a state which is not properly detected if the vehicle is picked up for example.

6.6 Summary

We use fuzzy logic as control paradigm for the controller running on our follower node, because it can run on limited hardware, is computationally fast, is robust to imprecise and unreliable data, reduces the design and development time and handles non-linear systems better when compared to conventional approaches.

A fuzzy controller executes three basic steps: *fuzzification*, *fuzzy inference* and *defuzzification*. During fuzzification, the numeric input values are mapped to fuzzy sets by applying the membership functions. Based on the fuzzified inputs, the controller evaluates its IF-THEN rules and produces an aggregated fuzzy output. The final control action is derived by defuzzifying this aggregated fuzzy output.

The control objective of the follower is to adapt its velocity and heading according to the movement information communicated from the leader. The vehicle has separate motors for accelerating and steering and therefore it is reasonable to decompose the problem into two independent fuzzy controllers, with the benefit of simplifying the design and tuning. However, these two control concerns are not entirely uncoupled as steering influences the vehicle's friction and the current velocity influences the rate of steering. Also, steering works inverted when the vehicle drives backwards. These and other issues are resolved in the controller's post-processing step, executed just before actuating the car motors.

We define a simple model for the controlled vehicle. We model the vehicle's heading dynamics as a linear angular response to the steering control from the heading controller. The model for the vehicle's velocity behavior is more complex, as it requires a model for the friction incurred during driving. The effective acceleration of the vehicle is the difference between the acceleration resulting from the applied throttle and the friction. Therefore, the throttle control will be less effective when it used to speed up than when it is

[2]The initial version of that controller even had a tendency to catch fire when the brake was not released in time.

used to slow down, making the control problem asymmetric for the velocity controller. A model for the friction is provided in Chapter 7.

The heading and velocity fuzzy controllers are structurally almost identical. The only differences lie in the range of the input values, the definition of the membership functions and the post-processing operations. The two fuzzy controllers follow the design methodology proposed by Mamdani [32]. The building blocks are:

- Inputs: As inputs we use the error e and the change in error Δe between the actual values of the movement parameters and the desired values. This is a common approach for improving the controller stability when the output value is close to the optimal operating point.
- Fuzzy sets: For an increased control granularity, we define five fuzzy sets for each input, namely NM (Negative Medium), NS (Negative Small), ZE (Zero Equal), PS (Positive Small) and PM (Positive Medium).
- Membership functions: To leverage the computational effort, we use triangular membership functions spaced equally with distance N. The width of the membership functions influences the *aggressiveness* of the controller, i.e. what margin of error it tries to achieve.
- Rule-based inference: We use the *max-min* fuzzy inference method over the rule set.
- Output: The controller output is decided by defuzzifying the aggregated inference result according to the *center of gravity* (COG) method. To account for friction, the velocity controller subtracts a constant value from the throttle control when it is opposite to the current velocity, i.e. the controller is trying to slow down.

The post-processing stage of the controller provides no solution for the influence of steering on the vehicle's friction, as the velocity controller applies maximum throttle in those situations already and reducing the steering intensity is not a solution. The post-processing step does account for the following situations:

1. When the vehicle is driving backward, the steering control is inverted.
2. When the leader is standing still and the follower velocity gets close to zero, the inductive brake (see also Section 3.2) is activated. This is a much more efficient braking method than using only the throttle control, as it prevents any forward-backward oscillations.

Chapter 7

Simulation

As for most practical problems, simulations are a powerful tool to evaluate designs and perform preliminary tests. Simulations can significantly reduce development time and cost when practical testing takes much effort and resources. However, it is not always easy to produce a simulation that mimics reality for those aspects that are relevant to the design. In this chapter, we present simulations for our leader-follower problem.

For our leader-follower system, simulation is useful for:

- Testing: The implemented fuzzy logic controllers can be tested and evaluated quickly in a simulation. This avoids the need to take the actual vehicles to the test range and thus saves much development time.
- Evaluation and Tuning: In a simulated environment, it is much easier to gather performance statistics for controller evaluation. Using the statistics, we can tune our controller for optimal performance. This requires that the simulation provides realistic results. However, in any case, practical tests are necessary to evaluate the system in reality.
- Debugging: The simulations are valuable for debugging an implemented controller. It is very hard to find and solve bugs in a remotely running sensor node, especially when it is driving around. So, running the node software inside a simulation can provide the means to reproduce and solve the bug in the lab rather than in the field. Since data from both leader and follower nodes is logged during testing in the field, it is for instance possible to feed the movements logged from the leader to the controller simulation. If the controller shows the same erroneous behavior in the simulation, the error can be traced with relative ease.

To simulate a moving vehicle, it is necessary to define a model for that vehicle and its environment. Using that model, the simulation can calculate the new state of the simulated vehicle from the control signals generated by the controller. A good model is such that the controller's outputs cause the state of the simulated vehicle to change in a way that mimics what would happen in reality. Simulated sensor values are calculated from the simulated vehicle state. These are inputs for the controller. For our simple vehicle simulation, only velocity and heading are considered as vehicle state variables.

All simulations we show in this chapter use artificially generated leader movements to have full control over what is simulated. However, the validity of the simulations must be verified using actual test data. Because during our tests in the field the movement measurements of leader and follower are logged, it is possible to use the leader's data as input to the simulation and compare the output of the simulation to the movements logged by the real follower. Note however that, since the logged measurements of the vehicles themselves is highly unreliable, matching outputs are no guarantee for a valid simulation. Therefore, it is very important to make manual measurements during the tests as well. We evaluate the validity of the simulations presented in this chapter in Section 9.3.

The simulations are written in MatLab, but the fuzzy controllers we describe in Chapter 6 are implemented directly in C for the final implementation on the node. Luckily, MatLab provides the ability to incorporate

external C implementations in the MatLab scripts. This is what we use to link the controllers to the simulations. In Section 8.1.2 we even describe how the controller running on the actual node can be linked to the simulation.

This chapter describes three distinct simulations:

- Steering
- Velocity control
- Full control, i.e. both steering and velocity control

Steering and velocity control are first simulated separately, because these behaviors are implemented in separate controllers as explained in Chapter 6. These simulations are described in Sections 7.1 and 7.2 respectively. As we show in Section 9.1, the steering and throttle behavior of the vehicle influence one another, so it is imperative that the steering and throttle controllers are simulated working together as well. This simulation is described in Section 7.3.

7.1 Steering

In this section, we outline the exclusive simulation of the steering behavior of the vehicle. First, we describe the structure of the simulation model and subsequently we provide a short survey of the results that we obtain with this simulation. Unlike the velocity simulation we describe in Section 7.2, the simulation we describe here can also provide a live animation of the vehicle movements.

7.1.1 Simulation Model

The model for the steering behavior of the vehicle we defined in Section 6.4 is very simple. In this section, we describe a simulation of the vehicle's steering behavior using this model. The structure of the simulation using this model is shown in Figure 7.1. The heading controller has two inputs: the desired heading $\phi_{desired}$ and the current actual heading ϕ. The controller yields one output s, the steering control value. On the real vehicle, s determines the angle of the front wheels.

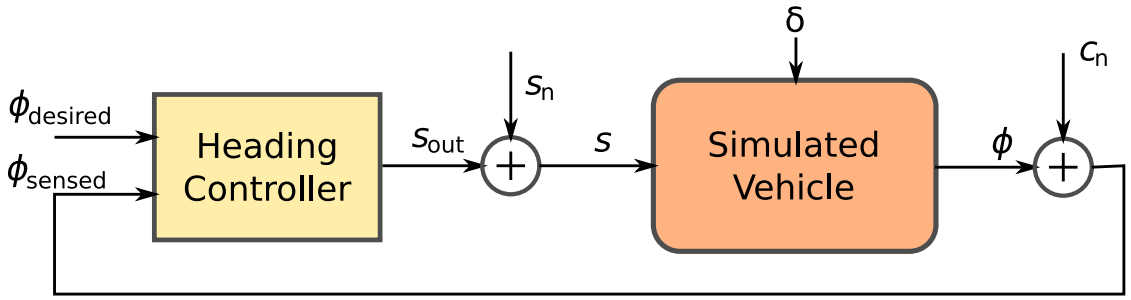


Figure 7.1: Heading simulation model.

In this simulation, the car drives at constant velocity. The only control that is active in this simulation is the steering control value s. The simulation starts with a car at full constant speed, driving in a preset direction. With each iteration of the simulation, the current heading ϕ of the car is updated with the current steering control value s using Equation 6.1 at page 65.

In reality however, the car is unlikely to maintain a constant heading when s remains 0 all the time. Due to inaccuracies in the steering mechanism of the car and external influences like a rough road surface, the actual steering angle is not always proportional to the steering control signal s. Consequently, the car will eventually deviate from its initial course even if it is not actively steering.

We call this factor in the simulation *steering noise* designated s_n. For this simulation, no elaborate analysis of the steering noise was performed and s_n is assumed to be an additive and uniformly distributed random value somewhere in the interval $[-a \ldots a]$. The value a is a simulation parameter specifying the maximum s_n value that can occur at all times. Note that, as specified in Section 6.4, the steering control signal lies within the range $[-1 \ldots 1]$. This means that the additive s_n noise cannot make the steering s exceed that range, which would otherwise mean that the car could steer faster under influence of noise.

Another source of noise in the system is the compass sensor. This noise is designated c_n and assumed to be additive and uniformly distributed as well. This noise is particularly interesting, because it can potentially render the controller unstable if the controller is tuned to be too aggressive (refer to Section 6.5.4).

In response to noise, the fuzzy controller will produce a noisy steering control signal. In reality, the steering of the vehicle is much too slow to follow the overall frequency of this noise. It will follow the average steering control value, thus the signal is in effect mechanically filtered. To prevent noise from creating unrealistic steering operations in our simulation, we apply a running average filter. The order of this filter determines the rate of change the simulated steering can follow.

7.1.2 Results

The heading simulation can be useful in two separate ways. First, it gathers statistics on the movement error, i.e. how far it deviates from the desired heading. Second, it presents an animation of the vehicle driving and the path traveled. The statistics resulting from this simulation are used to tune the heading controller. The animated simulation is used to visualize the vehicle's behavior, making steering problems evident.

As we explain in Section 6.5.4, the input membership functions of the fuzzy controller directly influence the sensitivity of the controller to the heading error. The statistics gathered from a realistic simulation can be used estimate the optimum configuration for the fuzzy controller. This is achieved by running the simulation for a range of fuzzy controller configurations and gathering statistics on the average error.

The results are presented in Figure 7.2. Each point in this figure represents the average of 20 separate simulations in which the simulated vehicle is instructed to drive a straight line. The compass noise magnitude is defined to be 4° and the steering noise magnitude is 8° .

It is apparent from the figure that the optimum value for the width of the membership functions is about 35° in this simulation. Furthermore, the increase in the average absolute error is at worst only a few degrees more than the optimum, meaning that the configuration cannot be considered very critical. The performance remains close to optimum in the range from to 29° to 41° . If the simulation is a good representation of reality, the actual optimum value is likely to be found within that range. We currently use 30° as width (meaning that $N = 15$; refer to Section 6.5.1). In Chapter 9, we show how the controller performs in reality with this value.

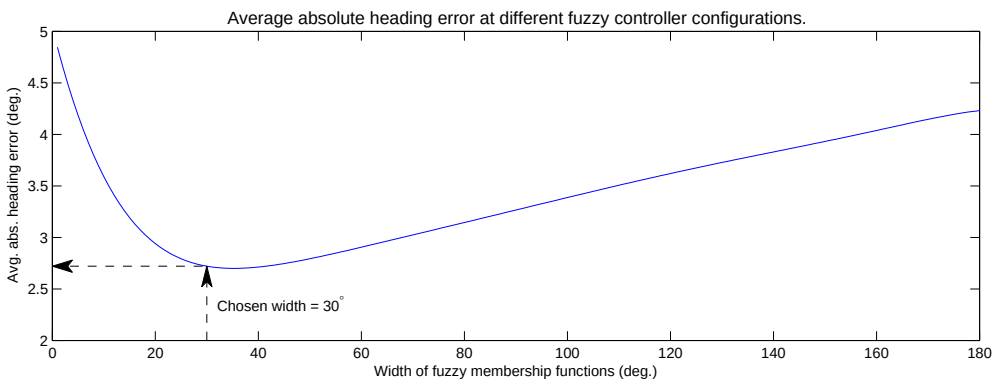


Figure 7.2: Average absolute steering error at different fuzzy controller configurations

In the animation mode, the simulation presents the simulated car in the middle of the screen and a plot of the traveled path in the top right, as shown in Figure 7.3. When the car's heading is zero it points directly towards the right side of the screen and when it is 90 degrees it points to the top. As presented in the figure, the simulated car drives some random path if the controller is disabled and s remains 0 all the time. This simulation is performed with a steering noise s_n magnitude of 5° .

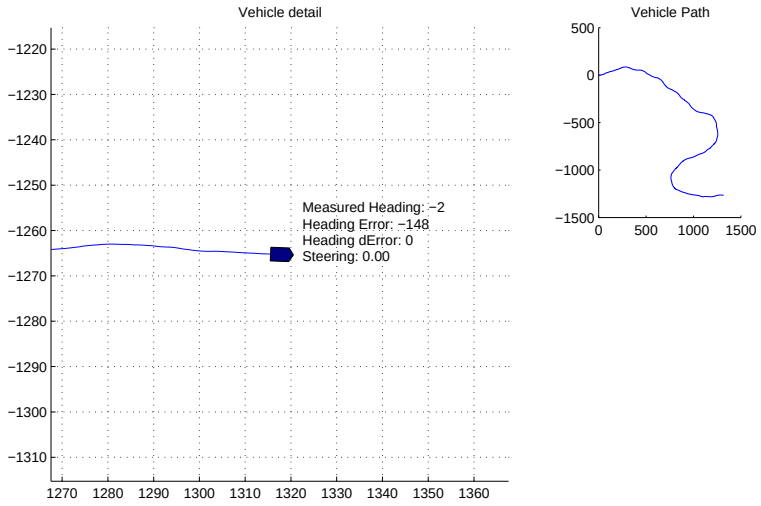


Figure 7.3: Heading simulation with disabled controller.

In Figure 7.4, the controller is enabled. The simulation is instructed to change the desired heading 90 degrees to the right at an interval of 100 iterations, starting with a heading of 30 degrees. Thus, when the controller works properly, this causes the simulated car to drive an approximately square pattern. This can be seen in the figure. The square pattern is not completely accurate and smooth, but that is not to be expected.

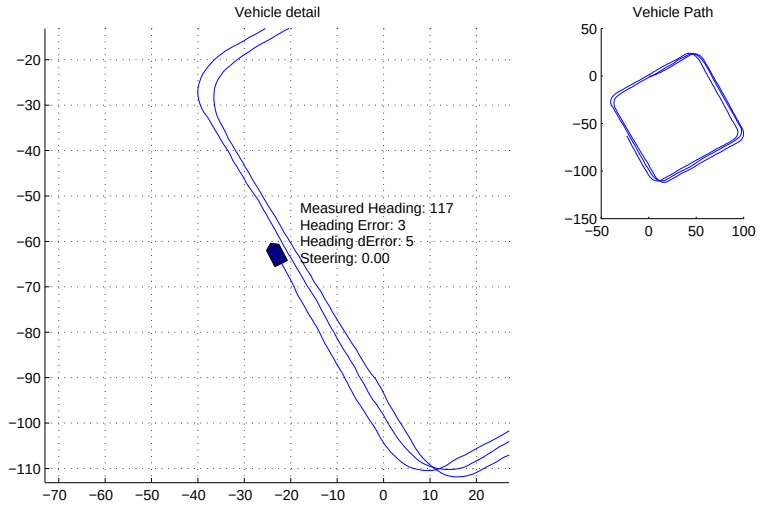


Figure 7.4: Heading simulation view.

The heading simulations show that the heading controller shows appropriate behavior for the simulated heading dynamics we model in Section 6.4. Additionally, the simulations provide an indication for the optimum fuzzy controller configuration.

7.2 Velocity Control

Simulating the velocity behavior of the follower vehicle is more complex than simulating its heading behavior, because more factors are of importance, most notably the various friction causes we describe in this section. Friction makes the system inherently unstable, forcing the controller to continuously apply a certain level of throttle to maintain the desired velocity. In contrast, the steering behavior of the vehicle is inherently stable: if there is no steering noise, the vehicle will maintain its current heading indefinitely with the steering fixed at straight ahead.

7.2.1 Friction Model

We present the velocity model of the follower vehicle in Section 6.4. To build a simulation, we also need a proper model of the friction a_f of the car. We consider the following causes of friction:

- Static friction: When the vehicle stands still, there is a static amount of friction. This is what needs to be overcome by the vehicle's engine to get it to move at all. This friction is primarily caused by friction of the vehicle's wheels with the surface it stands on and friction in the transmission from the engine to the wheels.
- Velocity-dependent friction: In addition to the static friction, there exists a velocity dependent friction component. This is at least partly caused by the drag that the vehicle incurs during driving. This drag is sometimes also referred-to as *fluid friction*: drag is the force that resists the movement of a solid object that passes through gaseous or fluid material, in this case a toy car through air. Generally this force is proportional to the square of the object's velocity v^2, but for very small objects with relatively small velocity it is approximately proportional to v according to Stokes' law. Other causes for velocity-dependent friction are imaginable, but for this simulation the exact cause of this friction component is not important but rather its magnitude and the fact that it is assumed to be proportional to the vehicle's velocity.
- Sliding friction caused by steering: Another cause of friction we consider in this model is the additional friction that presents when the vehicle is steering. As we explain in Section 6.4, the follower has no differential on the rear axle, so the rear wheels must always turn with the same speed. When the follower makes a turn, one of these wheels will partly slide over the ground surface to make up for the speed difference. If the turn that the vehicle tries to make is tighter, the speed difference of the rear wheels increases. Therefore, the sliding friction that is caused by steering is assumed to be proportional to the vehicle's steering angle δ. Note that for the velocity simulation the steering angle is always 0; the effect of this cause of friction will be discussed in the next section.
- Brake: The final cause of friction is the vehicle's inductive brake. We described its operation earlier in Sections 3.2 and 6.4. Its effect is is assumed to be static for the period it is activated.

These effects are summarized in the following equation:

$$a_f = a_{f,static} + c_d \times |v| + c_s \times |\delta| + a_{f,brake} \times b \tag{7.1}$$

with:

$a_{f,static}$	static friction acceleration
c_d	coefficient of velocity-dependent (drag) friction
v	velocity of vehicle
c_s	coefficient of friction by steering
δ	steering angle
$a_{f,brake}$	brake friction acceleration
b	brake control signal, $b \in \{0, 1\}$

Through the fiction model, the simulation relies heavily on a series of simulation parameters: $a_{f,static}$, c_d, c_s and $a_{f,brake}$. To obtain a simulation that closely approximates reality, these parameters must be inferred from realistic tests. It is however not the aim of this work to provide a highly realistic simulation and thus no elaborate experiments are performed to obtain reliable values for these parameters. Also, the fuzzy

controller should be suitable for different kinds of vehicles in various environments. The simulation gives us the ability to obtain a view on how the fuzzy controller would respond to the parameters specified.

For this work, the simulation parameters are based on a series of simple tests with the follower vehicle on a hockey field. In these tests the vehicle is tested in various situations: accelerating forward at full throttle, driving a very tight corner at full throttle, coasting onward from maximum velocity and braking at maximum velocity. The simulation parameters are estimated from the data acquired by the vehicle's sensors[1].

7.2.2 Simulation

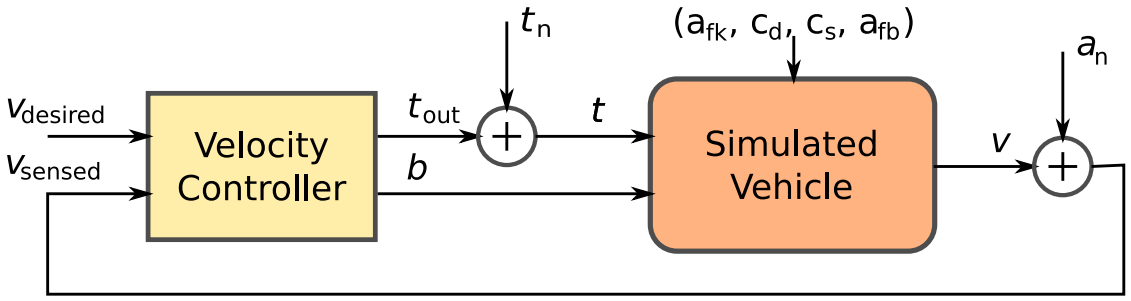


Figure 7.5: Velocity simulation overview.

The velocity simulation consists of the velocity model described by Equation 6.3 on page 65 and the friction model described by Equation 7.1. In Figure 7.5, these two models are combined and shown as the simulated environment. The simulated environment accepts the throttle t and brake b control signals from the velocity controller and yields a simulated velocity v.

To make the simulation more realistic, throttle noise t_n is added to the throttle control signal t and sensor noise s_n is added to the simulated velocity v. As for the heading simulation, all noise variables are assumed to be uniformly distributed values in the range $[-a_n \dots a_n]$ for which a_n a is a noise-specific simulation parameter that dictates the maximum amount of noise. Note that this simulation does not consider the accumulating error caused by the velocity integration.

The simulated noisy velocity measurement v_{sensed} is input to the velocity controller. The controller aims to achieve the desired velocity given by the $v_{desired}$ input using its throttle t_out and brake b outputs. The desired velocity is the main simulation parameter and any sequence of velocity changes can be fed to the simulation through this parameter.

7.2.3 Results

Because the pure velocity simulation omits the effects of steering, its results are only partly indicative of the controller's performance. That is why we do not provide a large list of results in this section. The results provided here are primarily intended as an example of how the controller works in a simulated environment.

Figure 7.6 shows the result of a simulation of the velocity controller trying to match the leader's velocity. The top graph shows the velocity of the leader and the simulated follower. The bottom graph shows the throttle control signal from the velocity controller. Obviously, the leader's velocity data is generated artificially for this case, because the graph is far to smooth for a real run.

First, the leader accelerates to a significant velocity and then coasts further until it stops. The follower manages to match this velocity closely. However, the next time that the leader moves, it reaches a velocity that is too high for the follower: the follower's controller applies maximum throttle to no avail. It decreases the throttle as soon as the leader sufficiently slows down. This situation is realistic, as our leader vehicle is

[1] Actually, these tests were not performed in such structured manner. Rather, the recorded data from various preliminary tests on the hockey field with the cars was used.

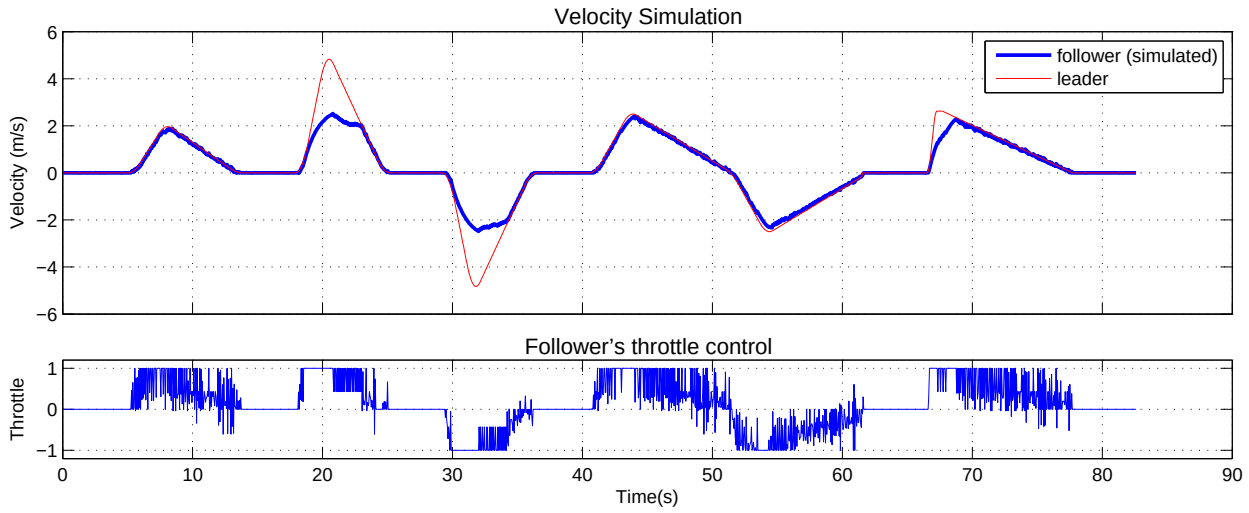


Figure 7.6: Example of a velocity simulation result.

much more powerful than the follower. Subsequently, the leader accelerates backwards to a very high velocity with similar results. The next leader activity shows the behavior of the follower when the leader does not stop between forward and backward movement. For the final leader movement the resulting maximum velocity is not very high, but the initial acceleration is very high giving the follower a hard time to keep up.

Note that this simulation does not take the effects of our velocity integration into account. In Section 4.3.2, we show that the influence of gravity can cause significant errors to accumulate in the velocity estimation, resulting in a significant offset. Obviously, if this occurs at the leader node, the follower will try to attain an erroneously high velocity.

The velocity simulations show adequate behavior for the velocity controller. It manages to match the leader's velocity when the follower physically can. Forward and backward movement makes no difference and the performance is equal.

However, when the simulated follower manages to approach the leader's velocity, it does not entirely match that velocity: the velocity plots do not overlap in Figure 7.6. Rather, the follower consistently managed a slightly smaller velocity. This difference is directly proportional to the width of the fuzzy controllers' input membership functions. This determines the margin below the desired absolute velocity that the controller deems 'acceptable'. Within that margin it does not apply more throttle than necessary to stay within that margin. Due to the asymmetric nature of the velocity control problem (refer to Section 6.4), the velocity does not rise above the minimum value where the effective acceleration is close to zero.

Currently, we do not provide a solution to this problem. A simple solution would be to add a constant to the desired velocity equal to the average deviation caused by this effect. Unfortunately, this deviation also depends on the magnitude of friction, meaning that this is very unlikely to be completely constant. Another solution would be to add a constant to the throttle control when accelerating, making sure the throttle is adequate to attain and maintain the full desired velocity. We do that already for deceleration. A cleaner solution would be to amend the fuzzy controller an extra input that gives the controller the ability to distinguish between acceleration and deceleration.

7.3 Full Simulation

A full simulation that includes both a model for the velocity and the steering behavior of the vehicle is necessary to test the complete controller. This simulation also shows how the post-processing part of the controller behaves and whether it properly compensates the influence that the steering and velocity controller are simulated to have on each other's actions.

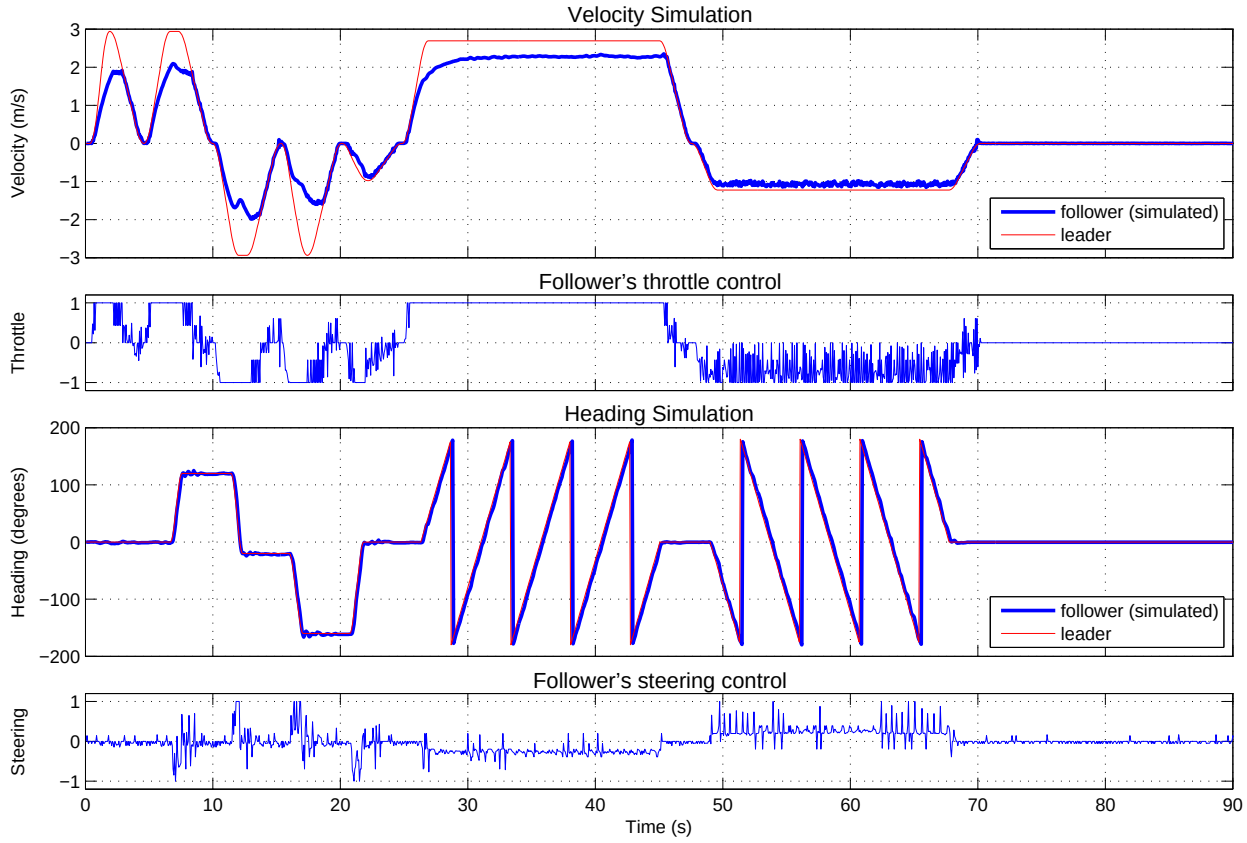


Figure 7.7: Example of a full simulation result.

The full simulation combines the velocity and heading models we describe in the preceding sections. This combination entails that the steering control value of the previous iteration of the simulation is fed to the friction model of the velocity simulation. The current steering control value is inverted if the velocity becomes negative to properly simulate backwards driving. Also, the simulation makes sure that the vehicle cannot change its heading if its velocity is close to zero.

Figure 7.7 shows a full simulation of the follower. Again, the leader's movement data is generated artificially to obtain full control of what is simulated. First, it shows the leader driving straight ahead. This first time it accelerates faster than the follower can. The follower attains the desired velocity when the leader is slowing down already.

The next four movements involve the leader driving and steering at the same time. The first time it steers when it is about to slow down, the second time it steers when it just attained its top speed and the third and fourth time it steers while accelerating. The fourth time the leader accelerated to a much smaller velocity. The simulation clearly shows the that the follower has trouble accelerating when it needs to make significant turns. At the level of steering friction that is configured for this simulation, the follower does not keep up with the leader when it is steering at high velocity. At those instances the throttle control is at its maximum, so the controller could not have done any better.

The final movements of the leader involve it driving four full circles at a very constant speed. The follower has no problem matching its own heading to the heading of the leader. However, at the first circular motion, the velocity of the leader is very significant and with this constant full right steering activity the follower has problems keeping up. In contrast, the second time when the leader is driving circles in the opposite direction at half the velocity, the follower has less problems.

The full simulation shows that the follower's controller works adequately when subjected to our model of the vehicle's dynamics: if the leader does not drive at a very high velocity, the follower manages to follow the leader's movements closely. On the other hand, if the leader drives too fast, the follower only manages to follow the heading correctly: the follower's velocity shows significant decline when it is steering.

The steering influence shown in Figure 7.7 cannot be compensated with a higher throttle, because the throttle control value is already at maximum when the steering influence becomes noticeable. In the presented simulation, the leader is driving too fast for the follower to keep up. As we show in Chapter 9 this is often seen in practice as well. The only way to solve such issues is by driving the leader slower or by using a faster follower than we have now.

7.4 Summary

We describe simulations in this chapter useful for testing, evaluating, tuning and debugging the follower controller:

- Testing: The simulations can be used to generally prove the validity of the controller's behavior.
- Evaluation and tuning: The simulations can be used to largely evaluate and tune the controller's performance without taking the vehicles to the testing range, provided that the simulations are accurate enough.
- Debugging: Problems that rise during testing in the field can be reproduced more easily, provided that the simulations are realistic enough.

The simulations use the models described in Section 6.4 with the addition of noise on sensor inputs and control outputs. For the velocity simulation of the vehicle, we define a very simple friction model. We only consider the heading and velocity of the vehicle as state variables in all simulation models. We simulate the heading and velocity controller first separately and subsequently combined in the full simulation. The simulations provide the following results:

- The controller behavior is valid in general: The controller we discuss in Chapter 6 works as intended in simulated conditions, which is a good indication that it will perform adequate on the actual vehicle. However, the simulations we present in this chapter are entirely based on our very simplistic models. In practice, more factors will have their influence on the controller's performance. In Chapter 9 we show how the actual performance compares.
- The controller is configured for simulated optimum performance: The simulations provide the means to quickly test a large set of possible controller configurations. If the simulation adequately approaches reality, the gathered statistics point to the most optimal configuration.
- The velocity controller does not fully attain the desired velocity: Due to the asymmetric nature of the velocity control problem, the controller never exceeds or even matches the desired velocity completely. The deviation relates directly to the amount of friction and the width of the fuzzy controller's membership functions. If the width is smaller, the controller will accept a smaller margin for error and consequently it will control the velocity closer to the desired value. However, this way it will never be able to match it. This can be solved by adding a constant value to the desired velocity or the throttle control signal when the vehicle is accelerating. Alternatively, the fuzzy controller can be amended to distinguish between accelerating and slowing down.
- The steering influence on velocity control is hard to compensate: The simulations show that the effect of steering on the driving friction cannot be compensated by increasing the throttle. The steering influence is only evident when the leader is driving very fast and the follower is having a hard time to keep up already. When the follower starts steering in those situations, it fails to match the leader's velocity with the throttle at maximum. In other situations, the follower appropriately applies more throttle when the friction increases. The steering problem only shows when the follower vehicle hits its limits, meaning that the best solution would be to use a follower car that can match the leader's performance.

In this chapter, we only show the value of these simulations for the initial testing, evaluation, tuning and debugging of the follower's controller. In Chapter 9, we show how the controller performs on the real vehicle and whether the simulations we present here are realistic enough to predict the behavior of the controller.

Chapter 8

Implementation

In Chapter 7, we present how the system works in simulation. In this chapter we take theory to practice by describing the actual implementation of the system on the nodes and vehicles. We provide a description of the implemented hardware and software components along with a survey of all practical problems we faced while building the system.

We describe the hardware-related issues in Section 8.1. In Section 8.2, we outline the software implementation, with a description of the tasks running on the node, the scheduling of these tasks and a description of how the equations of Chapter 4 are implemented. We also provide a benchmark of the fuzzy controller.

8.1 Hardware

We provide a discussion of the theoretical hardware concerns of this work in Chapter 2. In this section we delve into the practical details of the system's hardware. Figure 8.1, shows detailed pictures of the leader and follower vehicles.

(a) Leader (b) Follower

Figure 8.1: Detailed view of the leader and follower vehicles.

First, we discuss the placement of the sensors on the cars. This not a trivial matter, as we show in Section 8.1.1. Subsequently, in Section 8.1.2, we present the hardware setup used to debug platform-specific problems using simulation.

8.1.1 Sensor Placement

As we explain in Section 2.3, it is very important for both compass and accelerometer that these sensors are mounted rigidly on the vehicle, as changes in inclination during movement cause significant errors in the output signals. And, because both sensors have directional sensitivity, their orientation with respect to the vehicle must also remain constant. Also, the accelerometer must not be able to vibrate independently from the vehicle, as this adds additional noise to its output signal.

The compass sensor requires additional care. This sensor is greatly affected by stray magnetic fields. Therefore, this sensor can easily be interfered by the activation of electric motors or other actuators on the vehicle. It is always best to keep the compass sensor as far away as possible from the vehicle's actuators, but unfortunately this is not always an option. In those cases, the interference can be mended by including ferrous metal between the actuator and the sensor, e.g. a metal plate. Such a metal plate directs the magnetic field emanating from the actuator away from the sensor, effectively shielding the sensor. However, this ferrous object also distorts the earth's magnetic field, requiring the sensor to be calibrated as we describe in Section 4.2. Because the vehicles partly consist of metal and because the sensors are not mounted in an exact level position, calibration is necessary already. That is why this shielding solution does not add extra complexity to the system.

Due to practical reasons and the fact that we use different vehicles for leader and follower, the placement of the sensors on both vehicles is not equal. But, considering the physical parameters that our sensors measure, this should be no problem as long as the sensors are mounted with the same orientation with respect to their vehicle. Small differences in inclination are no problem, because these are calibrated out in the data processing stage as described in Chapter 4.

Leader Sensor Placement

We decided not to modify the leader vehicle and consequently the sensors and the node itself are attached to its hull. For transport, the module containing sensors and node can be removed from the leader. Unfortunately, the top of the leader vehicle is not a flat surface and therefore great care must be taken to attach the sensor module at the same position and orientation. To make sure that this procedure does not influence our measurements, the sensors are re-calibrated when we start a new series of tests on a particular day.

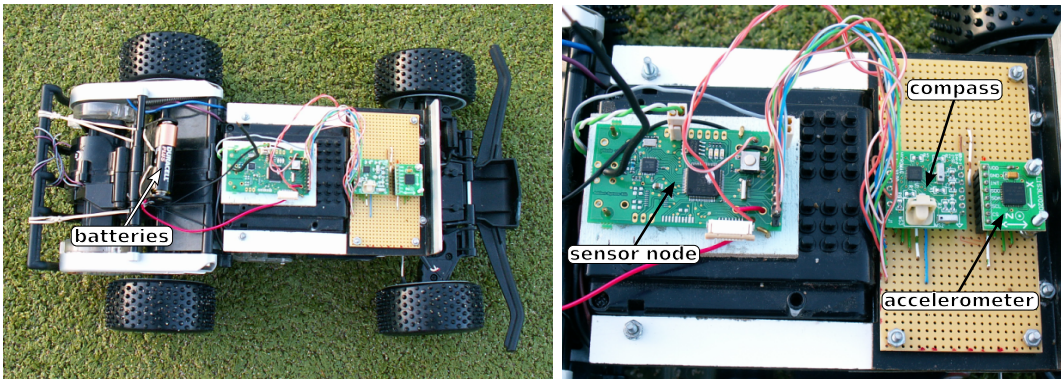
Figure 8.1(a) shows the placement of the sensors on the roof of the leader vehicle. The two sensors are mounted on top of the sensor node on a small piece of experimentation board. The sensors are mounted with identical orientation with their axes aligned. Together with the sensor node these form a removable node module. The node module is attached to the leader using a Velcro bond for easy removal. A disadvantage of this design is that it is less sturdy than a bolted joint, meaning that the node and sensors could potentially move relative to the car to some extent. In Chapter 9, we describe this as one of the potential causes for the leader showing lower velocity integration performance.

On top of the leader, the compass sensor is well away from the steering and drive motors of that vehicle. Measurements show that the compass sensor is not affected by the leader's actuators at that position. A disadvantage of that position is that the hull of the leader car can move significantly relative to the chassis, due to the rather weak suspension. This means that the hull will deflect when the car steers, causing a change in the inclination of the attached sensors. In Chapter 2 and 4, we show that this can have a significant detrimental effect on reliability of the obtained sensor data.

If the red hull is removed, the sensors can be mounted directly on the chassis, without significantly altering the leader vehicle. However, this position is about 5 cm lower and this way one of the compass' axes points directly towards the rear drive motor. This relatively small change in position is enough for the compass sensor to pickup very significant noise from the rear drive motor. That is why we decide to keep the node and the sensors on top of the leader vehicle. We show in Chapter 9 how this choice affects the system's performance.

Follower Sensor Placement

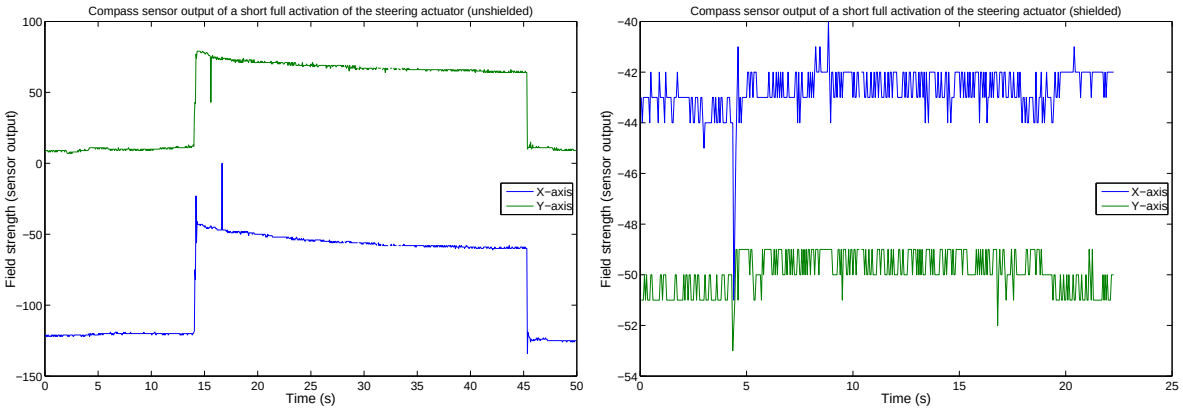
Figure 8.2 shows the placement of the sensors on the follower vehicle. The sensor node is located in the middle of the vehicle. The follower's motor controller is located directly beneath the sensor node. It is inside the vehicle's casing and therefore it is only visible when the follower vehicle is opened up. The compass sensor and the accelerometer are located at the front of the vehicle on a common sensor board. This is the most level position on this particular vehicle and the sensor board provides enough room for the current and any future sensors.



(a) Top view (b) Detail

Figure 8.2: Placement of sensor node and sensors on the follower vehicle.

On the leader, compass interference from the car itself can be avoided with relative ease. The follower is a different story. Unfortunately, the compass sensor on the follower is located directly above the steering actuator. Without proper measures, the compass reads a much increased magnetic field on all its axes when the steering is active. Consequently, steering activity causes significant heading measurement errors. This often becomes a self-amplifying oscillatory process by which the vehicle steers violently in seemingly random directions. If the steering is manually controlled with fixed intensity, its effect on the sensor is relatively constant and smooth as shown in Figure 8.3(a).



(a) Unshielded (b) Shielded

Figure 8.3: The influence of the follower's steering actuator on the compass sensor.

As we explain at the beginning of this section, this influence can be reduced by shielding the sensor with ferrous metal. We experimented with various objects as shielding between the sensor and the actuator and we find that the best shielding is achieved using a simple, thick (> 5 mm) sheet of steel. It is about 8 by 4 cm in size and we mount this directly below the sensor board. The result of this shielding effort is presented in Figure 8.3(b). Upon the start of a full intensity steering activity there is still a spike sometimes, but the constant offset is mostly gone.

Other Compass Interferences

As is evident from Figure 8.2, the sensor node's batteries are relatively far away from the compass sensor on the follower. On the leader, however, the compass sensor is located directly above the node and its batteries. We found that, when the node's batteries are replaced, the compass on the leader needs to be re-calibrated even if the batteries are of the same brand. Normally, one would assume that this is due to the change in supply voltage. Surprisingly, this also happens when the two current batteries of the node swap places, particularly when their charge level differs significantly. In this case, the supply voltage remains constant, but the compass' output is different in both cases anyway, particularly in terms of offset. We decided not to investigate this peculiar behavior further. Now we have the policy to re-calibrate each time something changes with respect to the leader node's batteries.

On the follower, the compass sensor is not only close to the steering motor, but also close to the front wheels. The particular toy car model we use as follower includes a sensor in the left front wheel to count its revolutions: a so-called odometer. We do not use this in our custom controller, so it is left unconnected. However, this sensor consists of two small, yet strong magnets and a coil to detect their passing. And therefore, as the left front wheel turns, it creates a changing magnetic field that is picked up by the nearby compass sensor as a sinusoidal signal. We removed the magnets from the wheel to end this interference.

We describe an additional cause for compass interference in Section 8.2: the radio also produces spikes in the raw compass measurements when it is activated during a compass measurement. We use scheduling to work around this issue by avoiding the activation of the radio during compass measurements.

8.1.2 Hardware Debugging

The simulations prove to be a very valuable tool for debugging the controller software. As we describe in Chapter 7, problems that occur in the field can be reproduced in the lab using data logged from the leader. This avoids the need to go in the in the field with the nodes to find problems. However, some problems that we encounter cannot not be reproduced in our simulations right away. These problems are of a platform-dependant nature. This means that the controller works when compiled for the PC, but somehow fails to work on the actual μNode. In this section we describe a simple hardware setup to tackle such problems.

Most such problems are related to the fact that a modern PC uses a word width of at least 32 bits by default. The MSP430, however, has only a 16 bits wide data path, meaning that the compiler will generate 16 bit computations by default. The follower's controller uses 32 bit computations at various instances of the code, for instance to perform multiplications with two 16 bit fixed-point fractional values. If the compiler is not explicitly told to use 32 bit computations, it will generate 16 bit computations on a 16 bit platform like the MSP430.

It is very hard to find such bugs if the controller is not running on the actual target platform, so a solution is needed to run the controller on an actual μNode in the simulation. Obviously, the simulation cannot be run completely on the node. Consequently, the only viable solution is to forward the simulated measurements to the node and read back the control signals. This is shown in figure 8.4.

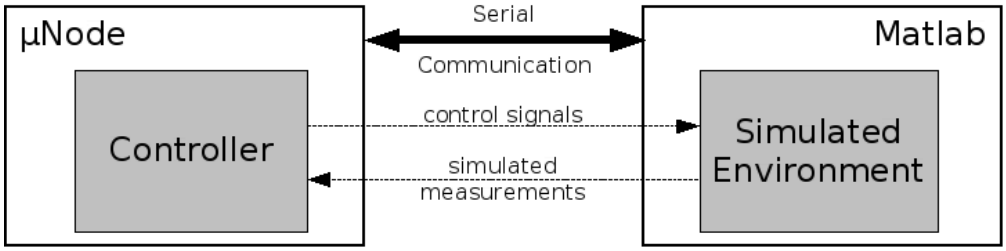


Figure 8.4: Simulating the controller on the node itself.

The used node is a gateway node equipped with a serial port. This can be hooked up directly with the PC that is running the Matlab simulation. Of course, both node and PC have the means to encode and decode

the input and output signals from the serial stream. For each simulation step, the PC sends the current simulated measurements over the serial connection to the node. On the node, the incoming measurements are fed to the controller and the resulting control signals are instantly sent back to the PC through the serial connection.

This is technique bound to take much longer than running the full simulation on the PC, so this would only be used in situations in which the presence of a platform-dependant problem is suspected. The debugging technique we present here is also used for debugging the data processing algorithms, albeit not in a simulation. Testing these simply involves piping the recorded raw sensor data through the software function under test and reading back the results for evaluation.

8.2 Software

This chapter provides information with respect to the actual implementation and the practical problems that rose during the making of that implementation. This section deals with the software aspect of that story. Particularly, this section describes how the software responsibilities are mapped on concrete operating system tasks and how these are scheduled on the definitive system. Also, because the sensor node's processor does not natively support floating point calculations, we briefly describe how we perform fractional arithmetic and how the equations of Chapter 4 are implemented. Finally, in Section 8.2.4, we assess the CPU time requirements of the fuzzy controller using a benchmark.

8.2.1 Tasks

In Section 1.6, we provide a brief overview of all the software activities. Subsequent chapters provide detailed explanation for each of these activities. In this section we show how all these activities map to software tasks and how these combined interact in the system. In Section 8.2.1, we show how the tasks described here are scheduled.

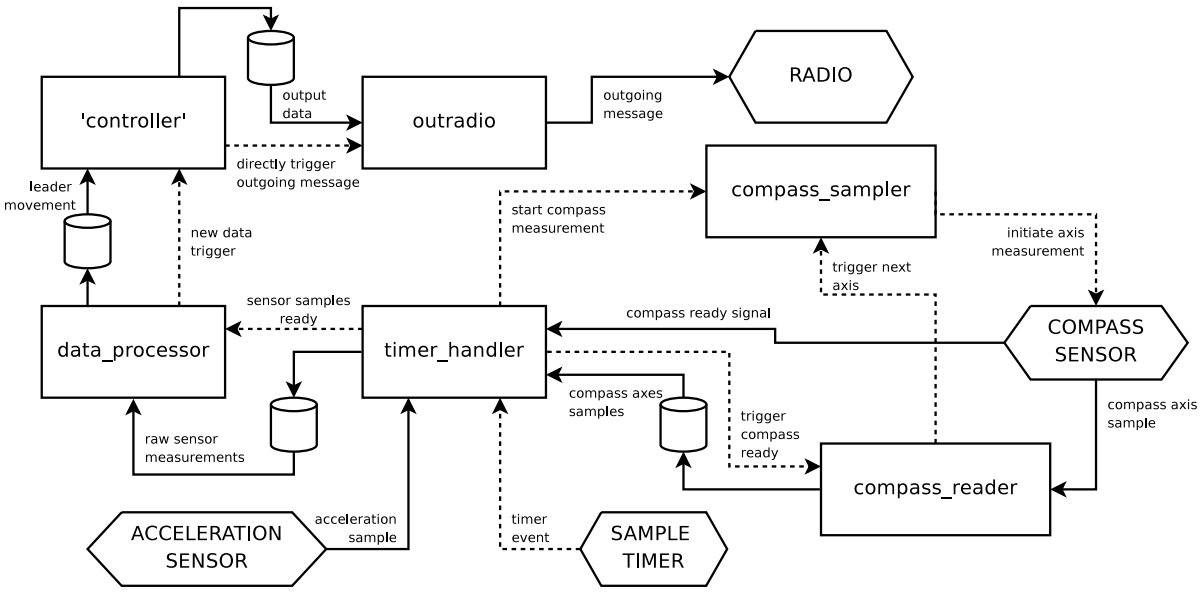


Figure 8.5: Task activation and information flow on the leader.

In terms of software complexity, the leader is relatively simple when compared to the follower. The leader runs no real controller and only needs to collect and broadcast measurements of its own movements. Figure 8.5 shows a a schematic representation of the tasks on the leader, how they are activated and what information

is exchanged. The dashed arrows are activation trigger relations and the solid arrows indicate the flow of information. The tasks are depicted as rectangular boxes and hardware or external components are indicated by the diamond-shaped rectangles. The diagram also shows cylindrical shapes that represent the temporary data storage between the moment the source task provides the data and the destination task picks up the data.

The beating heart of the system is the *sample timer*, which directly triggers the *timer_handler* task. This task is responsible for sampling the accelerometer, initiating compass measurements and triggering the data processing if a a set of new sensor measurements is available. As we describe in Section 3.1, ten accelerometer samples and two compass axis samples are collected each measurement period. Once these complete, the timer handler task triggers the *data_processor*, which calculates heading and velocity data using the incoming measurements.

The actual compass sampling activity is not performed by the *timer_handler* itself. That is performed by two separate tasks of which one, the *compass_sampler*, sets up the compass sensor for a new axis measurement and the other, the *compass_reader*, reads the resulting measurement when the sensor is ready. For the x axis compass measurement, both tasks are triggered by the *timer_handler*. The *compass_reader* is started by the *timer_handler* when the compass sensor has activated its ready signal. This ready signal is not an interrupt and therefore the *timer_handler* will not notice the ready signal until it itself is started by the sample timer. The *compass_reader* puts its measurements in a storage buffer to be read later by the *timer_handler*. When the *compass_reader* task finishes, it triggers the subsequent y axis measurement by signaling the *compass_sampler*. Thus, the *compass_sampler* is not directly triggered by the timer handler the second time, but rather indirectly trough the *compass_reader*.

Because the compass and the accelerometer are on the same SPI bus as we describe in Section 2.3.1, the *timer_handler*, *compass_sampler* and *compass_reader* are mutually exclusive: in the task specification of the AmbientRT operating system they share the same resource. This is also why reading the compass is split into two tasks: as we describe in Section 3.1 the compass can be set to measure in the background, leaving the SPI bus available for accelerometer measurements.

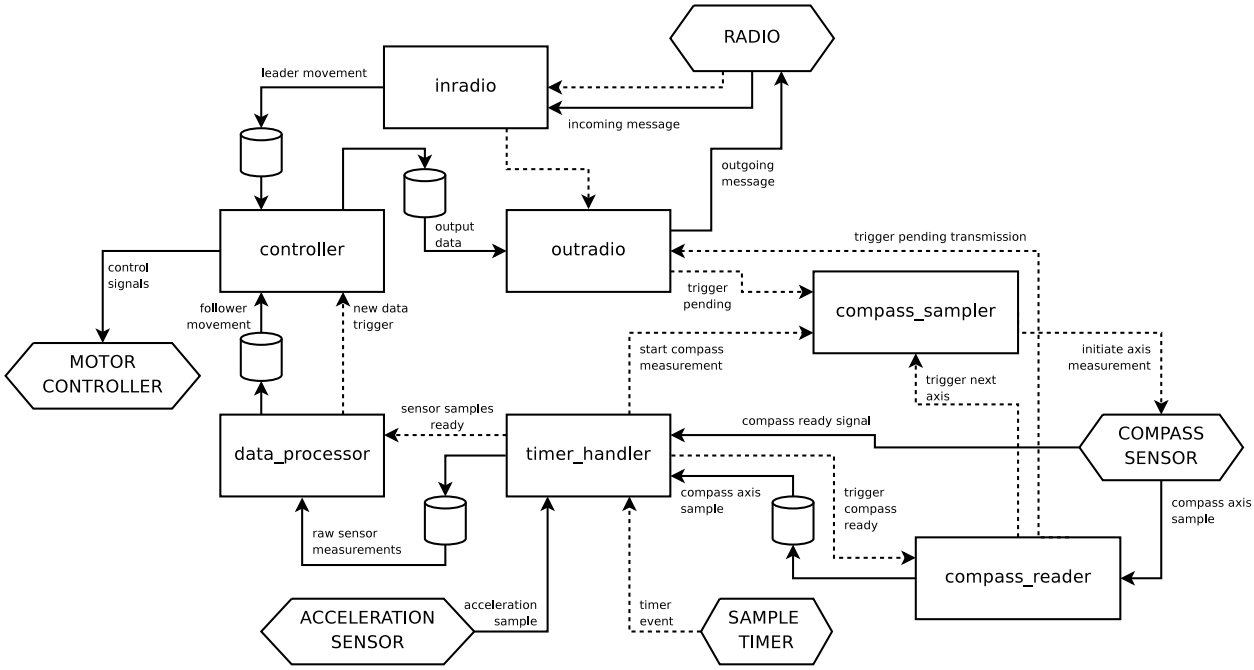


Figure 8.6: Task activation and information flow on the follower.

On the leader, the *data_processor* task triggers a dummy *controller* task which does nothing more than formulate an outgoing message for the *outradio* task that uses the radio to broadcast it over the radio channel. The implementation of the *timer_handler*, *compass_reader*, *compass_sampler*, *data_processor* and *outradio* tasks is common between leader and follower. To facilitate this, the leader provides a dummy *controller* task that implements the otherwise missing link between the data processor and the outradio

tasks.

We describe the protocol and the contents of the radio message in Chapter 5. In this section we are only interested in the fact that the transmitted message is picked up by the follower. Figure 8.6 shows the task diagram for the follower. The incoming radio message triggers the *inradio* task at the follower. First of all, this task stores the movement parameters coming from the leader for later use by the follower's *controller* task. The follower's controller is thus not activated by an incoming radio message. Other than that, it triggers the *outradio* task on the follower to transmit the follower's own message in the protocol. The *outradio* task uses the last output data from the controller that is stored after each controller execution.

However, as we outline in the next section, the *outradio* task must not be run if a compass measurement is active, i.e. when the *compass_sampler* was run but the *compass_reader* was not. If that is the case, the *outradio* task is not triggered and a transmission is marked as pending. When the *compass_reader* is finally triggered at the end of a compass measurement, it checks whether a transmission is pending and instantly triggers the *outradio* task if that is the case. If the *compass_reader* just finished the x axis compass measurement, the y axis measurement is not triggered in this case. In stead, the *compass_sampler* is triggered for the y axis measurement by the outradio task when it finishes the transmission. In the next section we show an example of how this is scheduled.

On the follower the controller is of course not a dummy. It is triggered by the arrival of new local measurements and, when activated, it collects the latest stored measurements from the leader. Using the locally measured data and the data from the leader, it formulates control signals for the follower vehicle's actuators. As we describe in Section 3.2, this is sent to the motor controller which performs the actual operations.

8.2.2 Scheduling

On sensor nodes, CPU time is a scarce resource. Therefore, it is likely that several of the software tasks we describe above need access to the CPU at the same time. The operating system that runs on the nodes assigns CPU time to each task. As we describe in Section 2.1, we use the AmbientRT Real-Time Operating System (RTOS) for our nodes. Being an RTOS, it assigns priority to tasks depending on their maximum allowed completion time; the deadline. This particular RTOS schedules the tasks with the earliest deadline first (EDF) and a running task is preempted when a new task with an even earlier deadline comes in [23].

In Section 3.1, we present a part of the software scheduling already because that is where we discuss the sensor sampling strategy. It is important that the sensor sampling schedule is not interrupted or inhibited by other tasks, because this can cause accelerometer samples to be missed. This is not a problem for the compass sensor, but the velocity integration algorithm will produce even less reliable results if the accelerometer is not sampled at regular intervals.

Primarily, the scheduling of the communication protocol that we discuss in Section 5.1 is liable to interfere with this sensor sample schedule. This is particularly true at the follower, where the protocol messages can be received at any instant of a sensor sampling period. This is because the follower does not synchronize its sampling timer with the incoming messages from the leader. At the leader, the problem is less profound, because messages are always sent right after the completion of a sampling period, since that is when a new set of measurements becomes available.

Although the compass sensor sampling intervals are not very critical, the compass sensor is affected by the protocol scheduling in another way. A radio transmission interferes with the compass sensor causing significant spikes in its output signal. Therefore, it is very important that no radio activities occur during a compass measurement. Because, at the follower, the transmission of a packet must be performed right after the reception of a packet from the leader, it is not a good idea to delay the radio transmission until all compass measurements are complete. This could cause the protocol to fail, since the next message from the leader may be transmitted at the same time as the follower finally gets a chance to transmit its message.

The radio is identified as the cause of the interference, because the interference disappears if the measurements are transmitted directly through a serial connection with the radio disabled. The exact mechanism of this compass interference is uncertain however. It could be caused by the significant magnetic field emanating from the radio antenna. On the other hand, it may be caused by the significant power drain from the radio chip; up to 30 mA at full transmission power (+10 dBm). This can cause noise on the power supply.

Experiments suggest the latter, because the use of a stabilized power supply in stead of batteries reduces the spikes. Also, when the technique described above is employed on the leader it exhibits almost no spikes, whereas the follower still has some remaining noise, even when its actuators are deactivated. This is possibly caused by the radio in receive mode, during which it still consumes up to 12 mA. If this were caused by the magnetic field from the radio, the interference should diminish if the transmitting leader is placed at greater distance. This is not observed.

Instead of delaying the radio transmission, we decide to delay compass measurements when the transmission of a message is due on the follower. This means that the sequence of two compass axis measurements can be interrupted by a radio transmission. However, if a compass axis measurement is currently active, the radio transmission will have to wait until that measurement finishes. Thus, this is not a preemptive process. On the leader, this process is simpler, because we know when the radio transmission is scheduled: right at the beginning of a new sensor sampling period. To avoid conflict between the compass and the radio on the leader we start the compass measurements a few accelerometer samples later.

Figures 8.7 and 8.8 provide examples of how scheduling is performed on the leader and follower node respectively. On both leader and follower the *timer_handler* task is the beating heart of the system. As we explain in Section 8.2.1, it is responsible for sampling the accelerometer, triggering the compass measurements and initiating data processing when a new measurement period completes. Both leader and follower have two separate tasks to initiate and end a compass measurement: *compass_sampler* and *compass_reader* respectively. In Chapter 3 we provide the basic explanation of the sensor sampling for leader and follower.

The figures do not show the deadlines for each task. The deadline for the sensor sampling tasks is such that they must complete before the next sample is due. The *timer_handler* task has a particularly harsh deadline to make sure it gets the highest priority during in the EDF scheduling algorithm. This prevents missing accelerometer samples. The deadline of the compass sampling tasks is longer, but still shorter than the radio and controller tasks to make sure these are executed closely after being triggered.

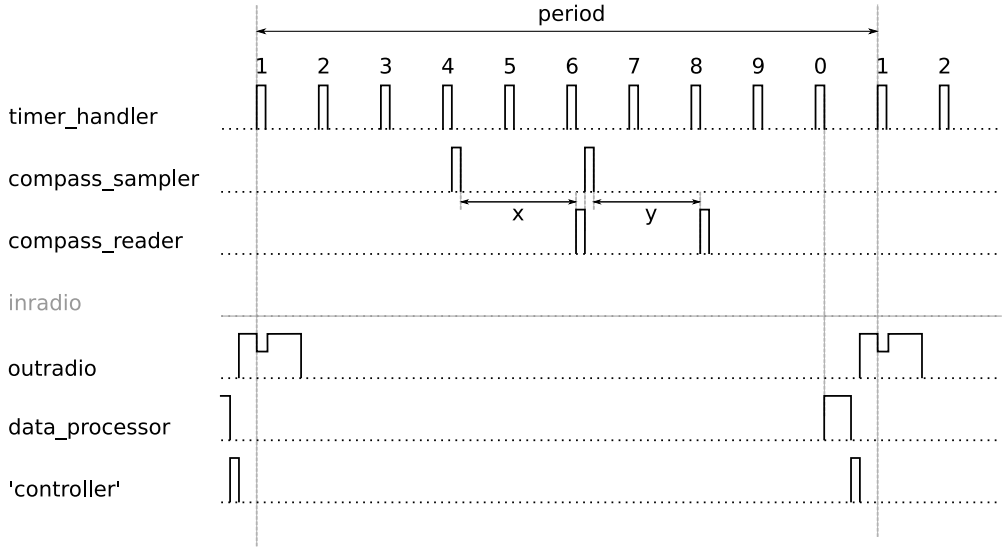


Figure 8.7: An example of task scheduling on the leader.

On the leader, the scheduling of the sensor sampling along with the other tasks is very straightforward. To prevent radio interference on the compass, the compass measurements are not started until after the fourth accelerometer sample. The radio transmission in the *outradio* task is performed closely after the previous measurement period finishes and new data becomes available when the sample counter returns to zero. But first data processing needs to be performed in the *data_preprocessor* task. This task triggers the frivolous *controller* task that performs nothing more than the composition of the outgoing message that is subsequently transmitted by the *outradio* task. Note that the leader does not listen on its radio, so the *inradio* task is not present in the scheduling.

In contrast, the scheduling on the follower is complicated by the asynchronous incoming messages from the leader. Figure 8.8 shows the most dire situation imaginable: the leader's message is received by the

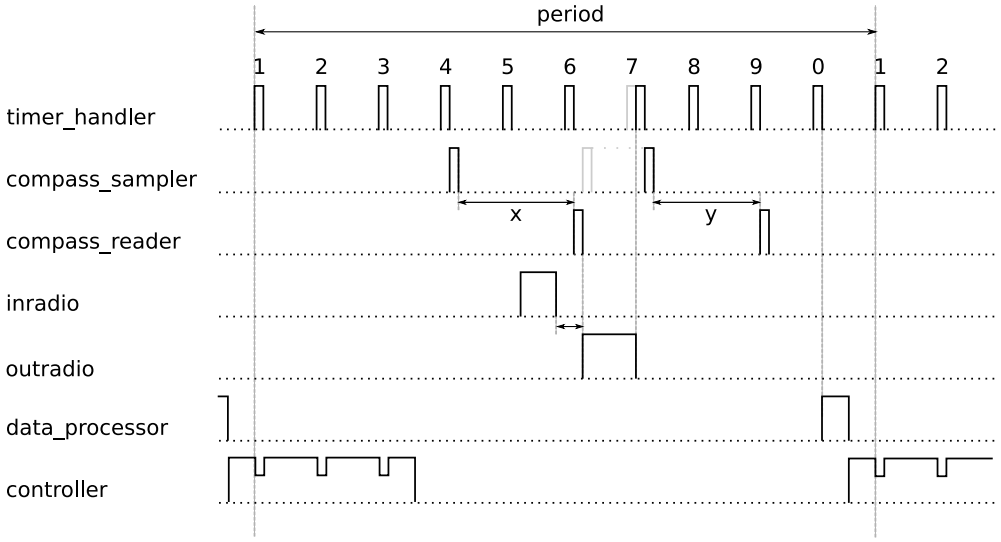


Figure 8.8: An example of task scheduling on the follower.

inradio task during the x-axis compass measurement. The active compass measurement is not interrupted to transmit the follower's own message. In stead, when the compass measurement finishes, the *compass_reader* task triggers the pending radio transmission. The start of the y-axis compass measurement is delayed until after the radio transmission is performed by the *outradio* task. Note that the *outradio* task includes a final delay after the radio transmission to make sure that the message is actually transmitted by the radio chip before the next compass measurement starts.

As on the leader, the *controller* task is triggered by the finished data processing. On the follower, however, this task is not trivial and takes a relatively long time to complete. It's deadline is the start of the next measurement period, which gives other more time-critical tasks ample opportunity to preempt it. In the figure, these preemptions are shown as dips in the task's activation schedule. Note that the execution times presented in the figures are not accurately represented.

We verified the above scheduling using a logic analyzer connected to the unused generic I/O ports of the node to visualize the scheduling of the tasks. Upon the activation of a task, the associated output port is set to high and upon deactivation its reset again, yielding a corresponding plot on the logic analyzer screen. This visualization method produces a view that is very similar to the figures above. It works in real-time and produces an intuitive representation of what is going on inside the node. We also use this technique to asses the relative CPU utilization of each task in the system.

8.2.3 Fixed-Point Arithmetic

The sensor nodes we use have no hardware support for floating point arithmetic. This means that we either need to use software floating point arithmetic or fixed point fractional arithmetic when we need to work with fractional data. Software floating point is relatively slow and large in terms of the size of the included library. Therefore, fixed point arithmetic [53] is often the better choice.

Fixed-point calculations work using the processor's integer operations by choosing a fixed bit position for the decimal separator, i.e. the radix point, in the values. The position of the decimal separator is chosen in accordance to the needs of the represented value, i.e. what the integer range of the value is and what kind of precision is needed for the fractional part of the value. This means that a fractional value is represented as $M.F$, where M represents the magnitude or integer part of the value and F represents the fractional part of the value. The word width of the processor is a given limitation, meaning that the sum of widths M and F is smaller than or equal to the word width, but this can be extended using multi-word integer representations. Note that signed values have one bit less available for the magnitude and fractional parts.

For fixed-point calculations, multiplication and division needs special attention because the radix position

changes. Just like for normal integer values, the result of a multiplication is twice as wide in terms bits as the operands. In fixed point arithmetic a multiplication looks like: $M_1.F_1 + M_2.F_2 = (M_1 + M_2).(F_1 + F_2)$. For example, this means that a 9.7 value multiplied by a 4.12 value yields a 13.19 result. If values with different radix positions need to be added or subtracted, the radix position of one of the values needs to be changed using bit shift operations. Note that precision is lost during these operations. Also, for example if the 32 bit multiplication result above needs to be put into a 16 bit storage position, the radix position needs to be shifted. In summary, the programmer is responsible for keeping track of the radix position between the calculations and the results are bounded by the fixed radix position, making the choice for a radix position an important consideration.

For multiplications with 16 bit values it is important to keep in mind that the sensor node only has word width of 16 bits, meaning that the default multiplication that the compiler generates yields a 16 bit result, even when the result is assigned to a 32 bit variable. For fractional representations, this almost always leads to truncations of the integer part of the value. It is a common bug to forget that and such problems produce very interesting and puzzling results. We trace such issues using the procedure described in Section 8.1.2.

We use a fractional representation for the vehicle's heading and, internally, for the square root and arctangent functions we describe below. We currently use a range of $\langle -180, 180]$ degrees in a 8.7 fractional format for representing headings . This is not ideal in terms of implementation. We may just as well use a simpler and more accurate integer range of $\langle -2^{15}, 2^{15} - 1]$ to represent the heading around the full circle, just like $\langle -\pi, \pi]$ radians is used by mathematical convention. All such ranges are valid as long as the trigonometric functions are implemented accordingly. For this initial work we use degrees to be able to work with an intuitive heading representation.

Square Root and Arctangent

The data processing equations that we outline in Chapter 4 need an implementation of the square root and arctangent functions. A straightforward and fast method to approximate arbitrary (injective) mathematical functions on small embedded systems is the use of a look-up table. Such a table contains a piecewise linear representation of the function, represented as a large list of $(\text{input}, \text{output})$ pairs. Output values for inputs that are not directly represented in the table are obtained through simple linear interpolation of the surrounding table values. Therefore, number of points that the table has for the desired input range determines the accuracy of the function result.

On the other hand, a larger table occupies more space in the device's (read-only) memory and it takes longer to find the values in a larger table. The table functions use a simple binary search algorithm to find the table values that surround the input value, meaning that the look-up time increases proportional to ${}^2\log n$ for which n is the number of table entries. Our $\arctan(x)$ table currently uses 1024 entries and our $\sqrt{x}$ table uses 420 entries (the $\sqrt{x}$ table is optimized using Matlab).

The $\arctan(x)$ function is directly approximated using the method described above. To fully implement Equation 4.3 on page 42, extra operations are performed for the calculation of $\frac{y}{x}$ and the handling of the special cases where $x = 0$ for instance. This yields a direct mapping from the x and y compass vectors to the heading angle.

The vehicle's velocity is represented as a 32 bit value, but the $\sqrt{x}$ table only contains a mapping of 16.0 integer input values to 8.8 fractional output values. To make this table useful for the full 32 bit input range, the 32 bits are dynamically scaled down. This means that the most significant bit of the value is determined in the 32 bit input. The input value is bit-shifted to the right by an even number of bits to make the most significant bit just fit into the lower 16 bits. For this value the $\sqrt{x}$ output is estimated. The result is scaled up by bit-shifting it left by half the amount of bit positions performed for the input value. This procedure is valid since $\sqrt{x} = 2^n\sqrt{\frac{x}{2^{2n}}}$. The advantages of this procedure are that the table entries only need to have 16 bit values and that the approximation resolution scales with the input value.

8.2.4 Controller Benchmark

The method that we describe in Section 8.1.2 for debugging the controller by running it on the follower node can also be used for benchmarking the controller if more information is sent to the PC than only the control signals. This way, we determine how fast the controller and its sub-components execute. Upon the start of execution, the OS system time is recorded and subtracted from the new system time after the controller (component) finishes.

Table 8.1 shows the typical execution times for the full controller, its sub-controllers and the stages involved in the fuzzy inference (refer to Chapter 6). The time spent in the *fuzzification*, *inference* and *defuzzification* stages for the heading and velocity controllers is identical. Notably, the execution of the whole heading controller takes longer than the velocity controller because a modulo function is used to map the heading error inputs to the range $[-180; 180]$ degrees (refer to Section 6.5.4).

Controller Component	Avg. Execution Time
Full Controller	2.1 ms
Velocity Controller	0.9 ms
Heading Controller	1.2 ms
Fuzzification	0.1 ms
Inference	0.5 ms
Defuzzification	0.2 ms

Table 8.1: Controller benchmark.

The results presented in Table 8.1 show that fuzzy controller executes relatively fast. In comparison, acquiring a single accelerometer sample is scheduled to take 0.7 ms. Sampling takes a relatively long time because the SPI bus of the sensors is implemented in software.

Note that the results presented above only indicate the execution time of the fuzzy controller itself, not including the time needed to convert the control signals for the motor controller and send them over the I^2C bus. This takes a relatively long time as well, since the I^2C implementation is built in software. Using SPI between node and motor controller in the future should significantly reduce this time, because the motor controller can provide hardware SPI support. That opens the possibility to increase the protocol frequency and thus reduce the time spent in transmitting the commands to the motor controller.

8.3 Summary

The implementation of the system faces practical challenges. In this chapter, we describe the practical problems and their respective solutions for the hardware and software. We describe hardware issues relating to the position of the sensors on the toy cars and debugging programs that somehow only fail on the nodes. For the software part of the implementation, we describe how all tasks we described in the previous chapters interact and how these are scheduled on the node's CPU. Additionally, we describe how the relevant equations of Chapter 4 are implemented on the node.

Hardware Issues

The main hardware problem is the placement of the compass sensor on the toy cars, since this sensor is very sensitive to interference from the car's actuators. The accelerometer is not affected by the car itself, other than the vibrations caused by its movement. For both sensors, the orientation must be identical on both vehicles. The compass interference problem is solved differently on leader and follower:

- Leader: The compass sensor is placed on top of the car, well away from any of the vehicle's actuators. However, this location is less stable than mounting it directly on the chassis of the car, increasing the potential for inclination changes during driving. The alternative mounting directly on the chassis of the car causes significant interference on the compass sensor from the rear drive motor.

- Follower: The compass sensor is placed on a sensor board along with the accelerometer at the front of the car. This is located directly above the steering actuator, causing significant magnetic interference. This is solved by introducing plate of metal between the sensor and the actuator.

In addition to the actuators, we find that a compass sensor can also be affected by other components of the vehicles:

1. On the leader, the node batteries prove to have a very significant influence on the validity of the sensor's calibration. The batteries are located relatively close to the sensor on the leader and if the batteries are replaced or just swap position in the battery holder, the observed offsets on the axes changes significantly. This is solved by recalibrating the compass each time something changes with the respect to the nearby batteries.
2. The follower includes an odometric sensor with small permanent magnets. The compass sensor is located close enough to the wheel that includes these magnets to make rotation of that wheel cause a significant sinusoidal signal on the compass' axes. This is solved by removing the magnets from the wheel, as the odometer is not currently used anyway.
3. Somehow the radio interferes with the compass sensor causing spikes in the sensor signal. This is avoided by scheduling radio communication and compass sampling mutually exclusively.

Another hardware problem is providing a means to debug the software while it is running on a sensor node. This is important to find software bugs that only present on the nodes and not while running on the PC. In order to find these platform-specific problems, we run the software on a gateway sensor node with a serial connection to the PC. The PC provides the input values for the software function under test and reads back the results. In Chapter 7, we describe how simulation can be used for node debugging the controller. By running the controller on the node using the technique described here, the simulation can be used to find bugs that only surface during field test using the data recorded from the leader.

Software Issues

The nodes run AmbientRT, a real-time multitasking operating system (RTOS). This RTOS uses the earliest deadline first (EDF) scheduling with preemption, meaning that tasks that need to finish earlier are scheduled first.

The software tasks running on leader and follower are identical regarding data acquisition. The primary difference between leader and follower is the controller running on the follower and the fact that the follower must receive messages from the leader. The leader only measures and transmits movement information, making the task set of the leader simpler. We describe the following important task scheduling issues:

- Asynchronous message reception on the follower: The radio protocol we describe in Chapter 5 dictates that the follower must be able to receive incoming messages asynchronously, meaning that its sensor measurements and controller execution are not synchronized to the leader. This means that messages can be received at any time in the sensor sampling process, triggering the potential for interference. Particularly the accelerometer needs to be read in time to prevent missing samples. We solve this by giving the sensor sampling tasks relatively close deadlines when compared to the radio reception handler.
- Interaction between compass and radio: Somehow, the radio causes spikes in the compass sensor output when it is activated. The radio is identified as the cause, because the interference disappears when radio communication is substituted by a wired serial link. We solve this by scheduling such that simultaneous radio and compass activation is avoided. However, this is difficult on the follower, since receiving messages is performed asynchronously. For proper protocol operation, radio transmission takes precedence over pending compass measurements. However, an active compass measurement is not interrupted.

The calculations presented in Chapter 4 require the implementation of square root and arctangent functions. These are implemented using a simple piece-wise linear representation in a look-up table. The interpolation

of values between table entries requires fractional number calculations. Since the MSP430 does not provide hardware floating point support, we use fixed point arithmetic exclusively. Fixed point arithmetic uses normal integer variables for which the radix point is defined at a fixed position to represent fractional values.

Benchmarks show that the fuzzy controller executes very fast, particularly when compared to the time spent sampling the sensors and sending commands to the motor controller. Probably, the greatest performance gain can be achieved by using hardware in stead of software support for the SPI and I^2C protocols.

Chapter 9

Evaluation

In chapters 2, 3, 4, 5, 6 and 8, we provide a thorough description of the design and implementation of our leader-follower system. In Chapter 7 we verify the implementation using simulations, whereas in this chapter we go one step further by evaluating our system using the results of tests with the actual vehicles.

First, we outline our test approach in Section 9.1 with a description of all performed tests. In Section 9.2, we present the results of all these tests. And finally, in Section 9.3, to assess whether the simulations of Chapter 7 are realistic enough to predict the behavior of the follower in real conditions, we compare our test results with simulations using the movement data that was recorded from the leader.

9.1 Tests

This section outlines the tests we performed to assess the performance of our leader-follower system. We first test the heading controller separately on our university's running track and subsequently the full system is tested at the university's hockey field. We present the results of all these tests in Section 9.2.

9.1.1 Heading Controller

To isolate problems specific to the compass and heading controller, we first test the follower's ability to maintain a specific heading without velocity control or post-processing. In these tests, the leader is omitted and thus the follower does not actually follow anything; it is pre-programmed with a desired heading. This has the advantage that the desired heading is very constant: keeping an actual leader vehicle on a perfectly straight line is very difficult. The pre-programmed desired heading also means that the test is only influenced by compass noise from the follower and not from the leader.

However, the desired heading is not kept entirely constant. To make the tests more interesting and last longer, the desired heading is changed by 180 degrees at fixed intervals to make the follower return. Consequently, the follower should drive a linear stretch back and forth all the time. Because velocity control is omitted, the throttle is fixed at maximum.

In Chapter 7, we establish that the heading controller works in simulation. A simulation of the desired behavior for this test is shown in Figure 9.1. Note that, although the heading is fixed, the actual path taken is not firmly defined. At the end of the stretch, the follower can choose whether it turns left or right to make the 180 degree turn. This determines whether the follower's current path will shift to the left or right respectively.

The tests of the heading controller are performed at our university's running track. Figure 9.2 shows a schematic representation of the test situation. The orientation of the stretch on the running track is about

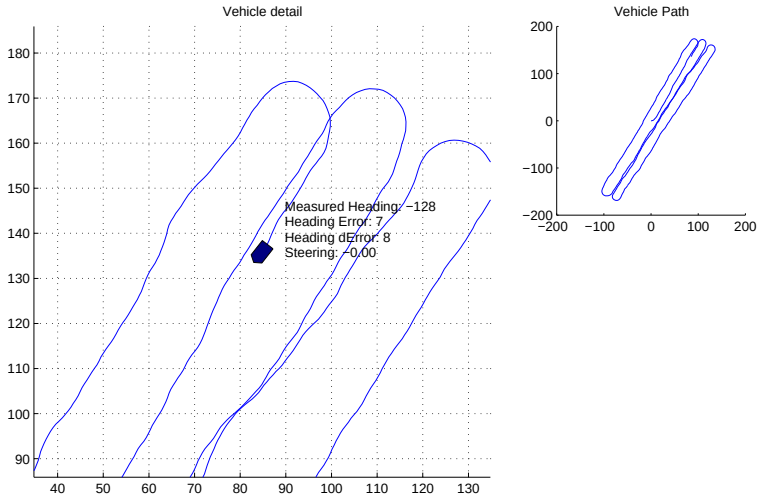


Figure 9.1: The simulation of the follower driving a linear stretch back and forth.

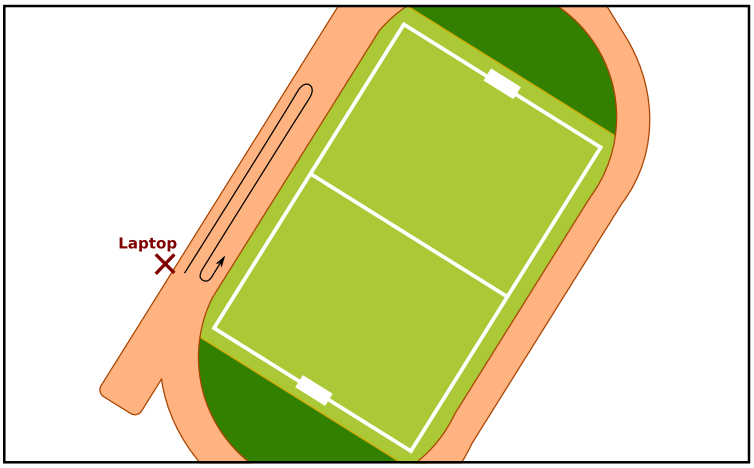


Figure 9.2: Tests on the running track.

32° relative to absolute north[1]. The sides of the track are lined with light masts and a metal fence. That is why it is important to keep the follower close to the middle of the track to prevent compass interference. The heading change time interval is tuned such that the follower drives back and forth over approximately half the track's straight section at maximum throttle. Note that this test cannot run indefinitely, because the follower's path will eventually shift to one side of the track, making it crash.

9.1.2 Full Tests

The velocity controller is not easily tested on the follower without heading control, because the vehicle will not drive a straight line when the steering is left unconnected. Therefore, we decide not to perform tests on the velocity controller separately. In this section, we describe how the full implementation is tested on the actual vehicles.

Location

All the full tests with leader and follower are performed on the university's hockey field. The advantage of this location is that it is a large flat surface with no metal anywhere on the field itself. The field is lined

[1]In the simulation of Figure 9.1 the 0° heading points to the right with the positive directions upwards, but to make the picture match up with a map of the running track, the simulated headings were adjusted accordingly.

with a metal fence and light masts however. But, when the cars are kept sufficiently far (about 2 m) away from the sides, their compasses will incur no significant interference from these metal objects. An advantage of the fence is that the cars can never escape the field.

The main disadvantage of this location is that it is outside. Consequently, the availability of our test ground depends largely on the dynamics of the Dutch weather. This problem is especially prohibitive during the winter months. The surface of the hockey field is artificial grass covered with large sand grains. This surface remains moist for a long time after it has rained and the sand grains have a tendency to stick to the car's wheels when they are wet. This pollutes the cars with wet sand on the inside. Other disadvantages of this location include the hockey players. Unfortunately, no other sufficiently large and preferably indoor location is available for this research.

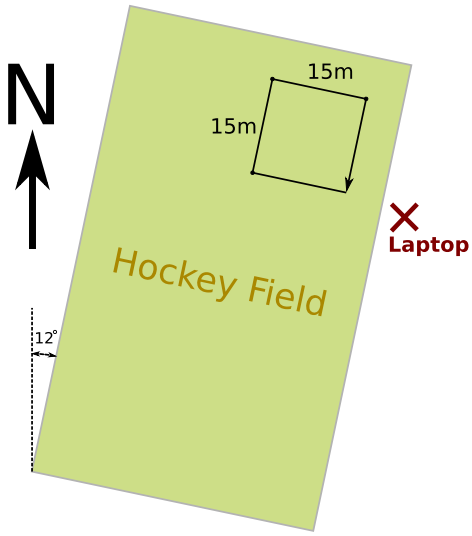


Figure 9.3: The geographical orientation of our test range.

The hockey field is shown schematically in Figure 9.3. It is not aligned with the direction towards the geographical north pole, it deviates from that direction by an amount of 12° . The square and linear tests we describe in this section are performed relative to the field itself in terms of orientation and not relative to north. To log the data from the leader and follower, a laptop connected to the gateway node is located at a fixed location at the side of the field.

Measurement Strategy

In order to reliably assess the performance of our leader-follower implementation, we need to measure how far the follower deviated from the leader's velocity and heading during each test. However, we do not have an absolute reference for the distance between leader and follower for the time they are driving. This also true for their relative heading. We can only acquire absolute measurements when the cars are standing still. This makes it hard to obtain useful test results.

Arguably, a continuous absolute record of the inter car distance and their relative headings could be recorded using a camera, but it is very hard to measure accurate distances on the camera's view if it is not mounted high above the field pointing in a direction normal to the fields surface. First of all this setup is not practical, because there is no practical way to mount a camera that high. Secondly, this would require elaborate image processing, because measuring the distance by hand on the camera frames is not a feasible option. Designing such intricate solutions to attain proper measurements is outside the scope of this work.

Because it is not feasible to assess the performance of the follower using continuous absolute distance and heading measurements, we are forced to use the measurements collected by the vehicles themselves. These measurements are unreliable and noisy, but these can still provide valuable information. The difference between the heading measured by the leader and the heading measured by the follower provides a useful continuous indication of how well the follower manages to attain the leader's heading. Performance problems

like over-steering are evident from these results, because the follower will correctly measure a deviation from the leader's heading in that situation. This is also true for the velocity.

Of course, the fact that the follower's movement measurements perfectly match the leader's measurements is by no means a guarantee that that is the reality of the test. For example, if at least one of the compasses is badly calibrated or influenced by other means, the follower's controller will work on unreliable data. Thus, in the follower's view on things, it may perfectly follow the leader, but in reality it would quickly lose track of it.

However, the situation can partly be assessed after a test drive is completed. If the follower followed the leader correctly, it should have maintained an approximately constant distance during the test while pointing in approximately the same direction as the leader at all times. If the test ends with the follower far away from leader pointing in a different direction, something went wrong during the test. If, however, leader and follower end the test at a distance close to the distance they started with while pointing in the same direction, the follower would have performed well.

Note that this assessment technique is not entirely reliable. The measurement at the end of the test is only a snapshot of the follower's performance. Subsequent errors may cancel each other out yielding pristine results at the end. On the other hand, a visually almost perfect test run can still be assessed as poor when the follower performs erroneous movements just before the test finishes.

In summary, we log the heading and velocity measurements from leader and follower during the tests and afterwards we measure the final heading the vehicles and the distance between them. As we explain hereafter, the distance to the finish line is also measured for both vehicles if a finish line is defined.

The heading of a vehicle is measured using a regular compass that is put on the ground at a distance of about 10 cm from the vehicle. Using a ruler that is laid in line with the vehicle's wheels, the vehicle's heading is read from the compass. The particular (odd) compass model we use can measure the heading in 64 increments, but all measurements hereafter are represented in degrees.

The distance between the vehicles is measured using a 50 m tape measure. That distance is defined as the distance between the rear ends of the vehicles, measured from the middle. The distances to the finish line are also measured from the vehicle's rear and to the centre of the finish line. Because this measurement method is not very accurate we read the distances with a precision of 5 cm.

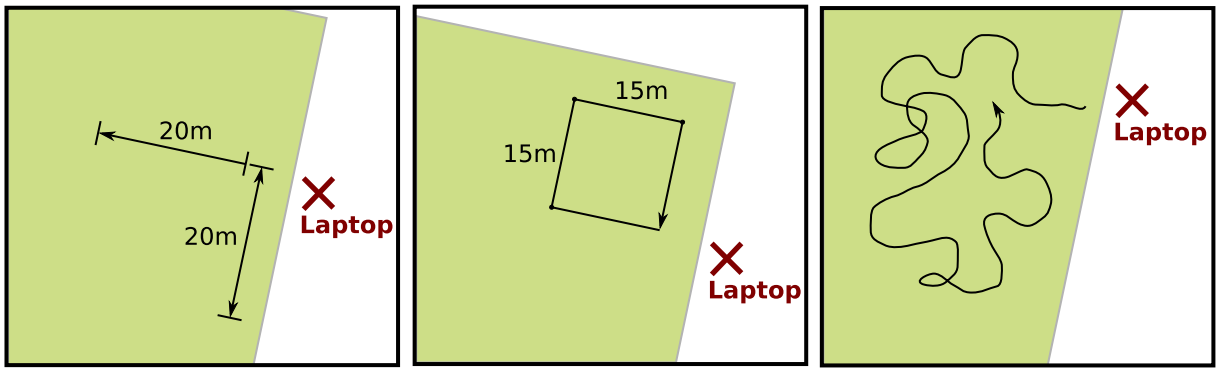
Tests

To test the performance of the follower in different situations, we devised tests with different drive patterns for the leader. We perform three types of tests on the hockey field:

- Linear tests: The leader drives in a straight line over a particular distance. The finish line is located 20 meters from the start line.
- Square tests: The leader drives an approximate square pattern. The edges of the square are 15 meters long and the corners are marked. The leader starts in one corner and finishes at the same position. Thus, the finish line is perpendicular to the start line.
- Random tests: The leader is driven along a random path during 30 seconds with varying velocity. It drives forward most of the time. The cars start at a fixed distance of about 2 meters. Their end location is random and therefore there is no finish line defined for this test.

Figure 9.4 represents leader's path during these tests schematically. The figure consists of a series of close-ups of the hockey field, one for each type of test. Twenty individual tests are performed to gain sufficiently reliable results for each type of test. The linear and square tests all begin at the same location with the initial distance between leader and follower starting at 1.8 meters. The random tests start where the previous test left of with the follower at an approximate distance of 2 meters. All tests are performed close to the laptop location to prevent high levels of packet loss.

As shown in Figure 9.4(a), the linear tests are not all performed in the same direction. The first 10 are performed east to west relative to the hockey field, whereas the final 10 tests are performed driving back



(a) Linear Tests (b) Square Tests (c) Random Tests

Figure 9.4: Schematic view of how the tests are performed on the hockey field.

and forth from north to south and from south to north respectively. The square tests in Figure 9.4(b) are all performed along the same square path.

Note that the paths given in Figure 9.4 are not an absolute. It is very difficult to drive the leader car along the designated path, because it is very fast and agile. Small mistakes in the steering will make it veer off quickly. Also, pulling the throttle too hard causes the car to spin in the corners. The linear and square tests during which the leader veers off the path for more than 3 meters are discarded and performed again.

9.2 Results

This section describes the results of the tests we performed with the cars on the running track and on the hockey field. The results of the running track tests are not outlined in detail, but primarily in terms of the goals achieved and lessons learned. The hockey field tests are performed several months later and provide the final results of this work. Those tests we do describe in great detail.

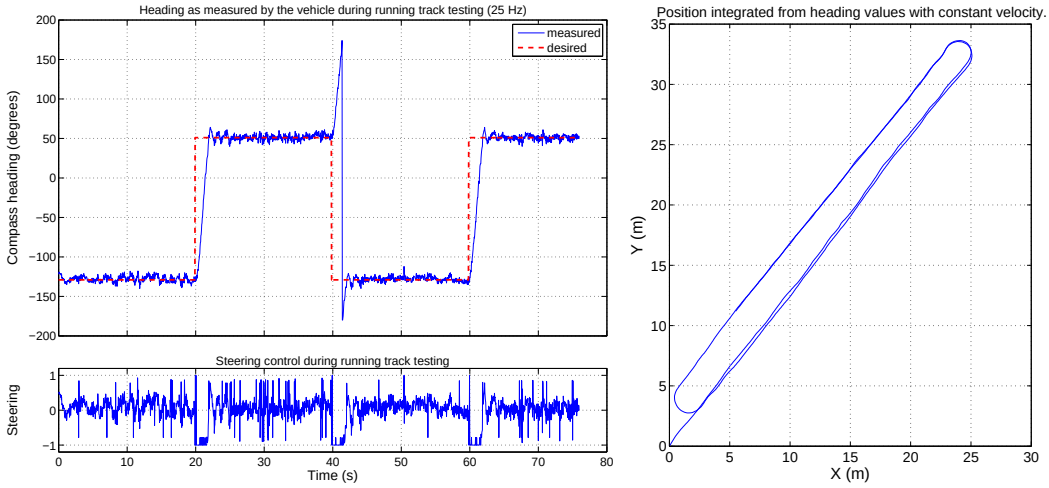
9.2.1 Running Track

In Section 9.1.1, we describe testing the heading controller on the running track. In this section, we present a short summary of the lessons learnt during these tests. The running track test results are particularly interesting because they demonstrate the problems that arise with the compass sensor and a sub-optimal fuzzy controller.

During testing it became apparent that the compass sensor is very sensitive to its position on the vehicle. During these tests the mounting of the sensor on the follower vehicle was similar to what is currently used on the leader (refer to Section 8.1.1). It is therefore imperative that the compass sensor is re-calibrated each time the sensor node module is removed. Also, the heading of the running track that is programmed into the node needs to be adjusted if the node is placed in a slightly different orientation on the car. Otherwise, the vehicle quickly finds itself crashed to the side of the track.

A bad calibration of the compass sensor is directly evident during a test run: the car will then not return along a path parallel to the initial. This deviation can be as large as 30° , making the car find the side of the track almost instantly after the turn that should have been 180° . Metal interference also has a very visible effect on the performance of the follower car. The large light masts along the side of the track cause it to sway heavily if approached too closely.

The running track is not as smooth as the hockey field: its surface is riddled with small ridges, deep footprints and acorns from a nearby oak tree. Because the node does not do dynamic tilt compensation for the compass readings, driving over ridges and debris causes significant noise on the measured heading.



(a) Heading control (b) Integrated Position

Figure 9.5: Example of a relatively short, but very clean test on the running track.

We present a simulated example of a test on the running track in Section 9.1.1. Figure 9.5 shows the results of a relatively short real test. The compass was mounted with its X axis pointing to the right side of the vehicle and the Y axis pointing backwards.

The 180° turns are clearly visible in Figure 9.5(a). After maintaining a relatively constant heading for about twenty seconds, the car took a very fast turn to right and came back. With twenty second intervals it performed more 180° turns, each time choosing to turn to the right. This way, it approximately remained on the same path, which is pure coincidence. The figure clearly shows that the measured heading is very close to the desired heading, except when it steers towards a new heading, which takes about two seconds, meaning that the follower can steer at a rate of about $\frac{180}{2} = 90°/s$ at full throttle.

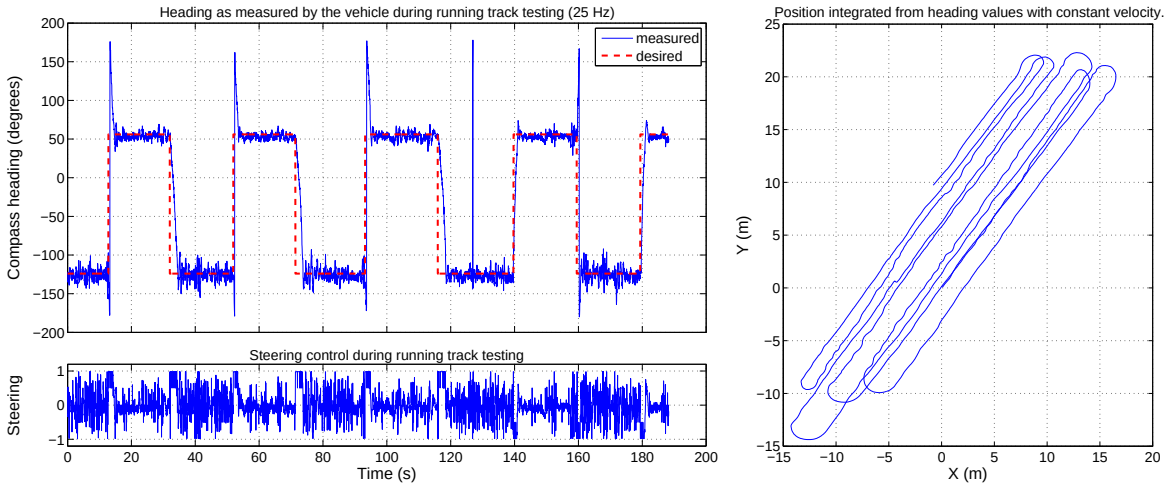
Looking closely at the heading right after a 180° turn, the car's heading controller had some difficulty stabilizing towards the new heading. This was observed as a significant swaying movement right after steering, which quickly disappeared when the vehicle gained speed again. For later tests, the fuzzy controller was tuned to be less aggressive to prevent this over-steering behavior. As we explain in Section 6.5.4, this is achieved by making the membership functions of the fuzzy inference system wider. This only partly compensated for the swaying effect, as we show in the next section.

Apart from the errors caused by the fuzzy controller, the heading readings are also influenced by the inclination changes caused by the relatively rough surface of the running track. Interestingly, when the car hit a large acorn it would get launched into the air a little, making it sway heavily afterwards. The heading of the car did not visually change due the impact itself however. In such situations, the controller tries to compensate for the virtual heading error that results form the sudden inclination change. This compensation is seen as aggressive, but mid-air, steering. When the vehicle returns to the ground, it is steering significantly, making it veer off course quickly.

To get an idea how the heading measurements would relate to the vehicle's position, Figure 9.5(b) shows the position integrated using the heading measurements of the car with the velocity assumed to be constant at 2 m/s. In reality, the velocity decreases if the car makes a turn or when it encounters debris. Although this figure is entirely based on the measurements of the vehicle itself and assumptions about its velocity, it does show how well the controller managed to maintain its heading measurements constant.

When the data of Figure 9.5(a) is filtered with a 5 tap running average filter to remove high frequency compass noise, the measured heading deviates at most 11 degrees from the desired heading when the car is not actively turning towards a new desired heading. The absolute heading error is then a little more than 2 degrees on average. This explains the very straight lines that the position integration of Figure 9.5(b) shows.

Figure 9.6 shows the same information for the longest successful run performed on the running track. The car made nine successful turns until it finally hit the side of the running track. However, this run is shown to



(a) Heading control

(b) Integrated Position

Figure 9.6: Results of the longest test on the running track.

be less accurate when we look at the maximum and average absolute error in the heading: 29 and 4 degrees respectively. Statistics extracted from a total of 24 minutes of testing in the course of 29 tests yields an average deviation of 3° , with a maximum of 39° .

The running track experiments show the following:

- The fuzzy heading controller performs adequate at fixed throttle.
- The compass is a very important factor in its performance however. If the compass' position, orientation or inclination relative to the vehicle changes even slightly, the calibration becomes inadequate and the sensor needs to be re-calibrated.
- Oscillating changes in the sensor's placement on the vehicle, i.e. vibrations, cause noise on the compass sensor readings accordingly.
- Large nearby metal objects cause significant measurement errors. In practice, the car would suddenly steer when it passed one of the large light masts closely.
- Irregular surface features can cause significant measurement errors as well, because the inclination of the sensor suddenly changes. For example, hitting one of the acorns on the track would make the car steer and sway heavily.

The tests on the running track show how the follower performs when it needs to maintain a heading at fixed throttle. This provides a test of the heading controller only and the interaction between leader and follower is not tested. The hockey field tests we describe in the next section show what happens if both velocity and heading are actively controlled in an effort to follow the actual leader vehicle.

9.2.2 Hockey Field

The hockey field is much smoother than the running track and it is free from debris. Also, the test range is wider, making it easier to avoid the metal fence and light masts at the side. Therefore, we would expect the compass readings to be much cleaner on this testing location.

The simulations of Chapter 7 give an indication of how well the controller on the follower can manage to mimic the leader's movements. The simulations show that steering friction at the follower can have a profound impact on its performance regarding mimicking the leader's velocity. The simulations are based on initial tests with the cars where this was observed already. Thus, if the leader drives with too high velocity, the follower will not be able to keep up during steering. Therefore, these tests are mostly performed at a reasonable velocity, which turns out to be the velocity achieved by the leader at minimum manual throttle. It is still relatively easy to drive or steer too fast with the leader car as we show in the course of this section.

	Leader	Follower
Linear tests	1.3%	1.1%
Square tests	0.6%	1.4%
Random tests	0.2%	0.1%

Table 9.1: Packet loss.

In the following subsections we present the results of the linear, square and random tests we performed on the hockey field. For each type of test, 20 individual runs were performed. We provide an analysis of the interesting statistics calculated from the test results.

In addition, we show examples of logs collected for several of these tests. These logs are shown as plots presenting the heading and velocity that the two cars measured for themselves and the control signals the follower issued. The heading plots sometimes show discontinuities. These are caused by the heading crossing the lower boundary and wrapping around. Internally, the headings are represented in the range $[-180 \ldots 180]$, meaning a heading of 200° is represented as -160° . To provide a clean picture, this range is sometimes changed for the construction of the plots to prevent a large amount of discontinuities clouding the picture.

During testing, it is not uncommon that packet loss occurs, so no log is entirely complete. In the plots of the logs the packet loss is visible as gaps in the graph. The logs provide the means to calculate the packet loss as identified by missing sequence numbers. The results listed in Table 9.1 show the average percentage of packets lost during the tests. The random tests show less packet loss, possibly due to the fact that these tests were performed closer to the gateway node. In general, the packet loss ratio is acceptable, meaning that our protocol works adequately.

Linear Tests

The linear tests are particularly valuable to assess whether the follower can maintain a approximately constant heading while following the velocity of the leader. However, the leader's steering is very sensitive and it is often biased towards either direction. That is why it is not that easy to drive a straight line with the manually controlled leader. For the tests presented here, we accept a best-effort level of heading error for the leader, allowing for it to sway around its designated course. Before we analyze the results of the linear tests in terms of statistics, we present the most interesting logs.

Figure 9.7 shows one of the linear tests performed at the hockey field. It yielded a final distance of 1.30 m between leader and follower and a difference in heading of less than 6 degrees. The follower traveled 20 cm farther than the leader relative to the finish line. Considering that leader and follower started at a distance of 1.80 m, their distance changed by 50 cm. These results seem very promising.

However, the figure shows that, although it managed to adequately trace the leader's velocity in the beginning, the follower fell behind significantly when the leader reached full velocity. The follower's throttle is active at its maximum most of the time, meaning that it is not a problem caused by the velocity controller in this case. The simulation of Section 7.3 hints to the causes of the failure of the follower to keep up: (1) the leader was driving too fast in general and (2) if the follower is steering, it incurs much more friction and if the leader is driving relatively fast, the follower will fall behind. This is exactly what happened here as shown in the lower graph of Figure 9.7.

Obviously, during this linear driving test the cars should not have been steering at all. As seen in the figure, the leader swayed moderately with a maximum deviation of about 8° . The follower, on the other hand, presented excessive steering with the deviation peaking up to 25° . This is clearly an instability in the steering controller. The period of the largest steering operations, starting after 5 seconds and ending after 7 seconds, is exactly when the follower fell behind significantly; the velocity even decreases while throttle is maximal.

Considering the follower's large swaying movement and failure to attain the leader's velocity, one would expect the final results of this test to be far from promising. However, this test is an example of successive errors canceling each other out, yielding a good final result. Because the follower was slower than the leader,

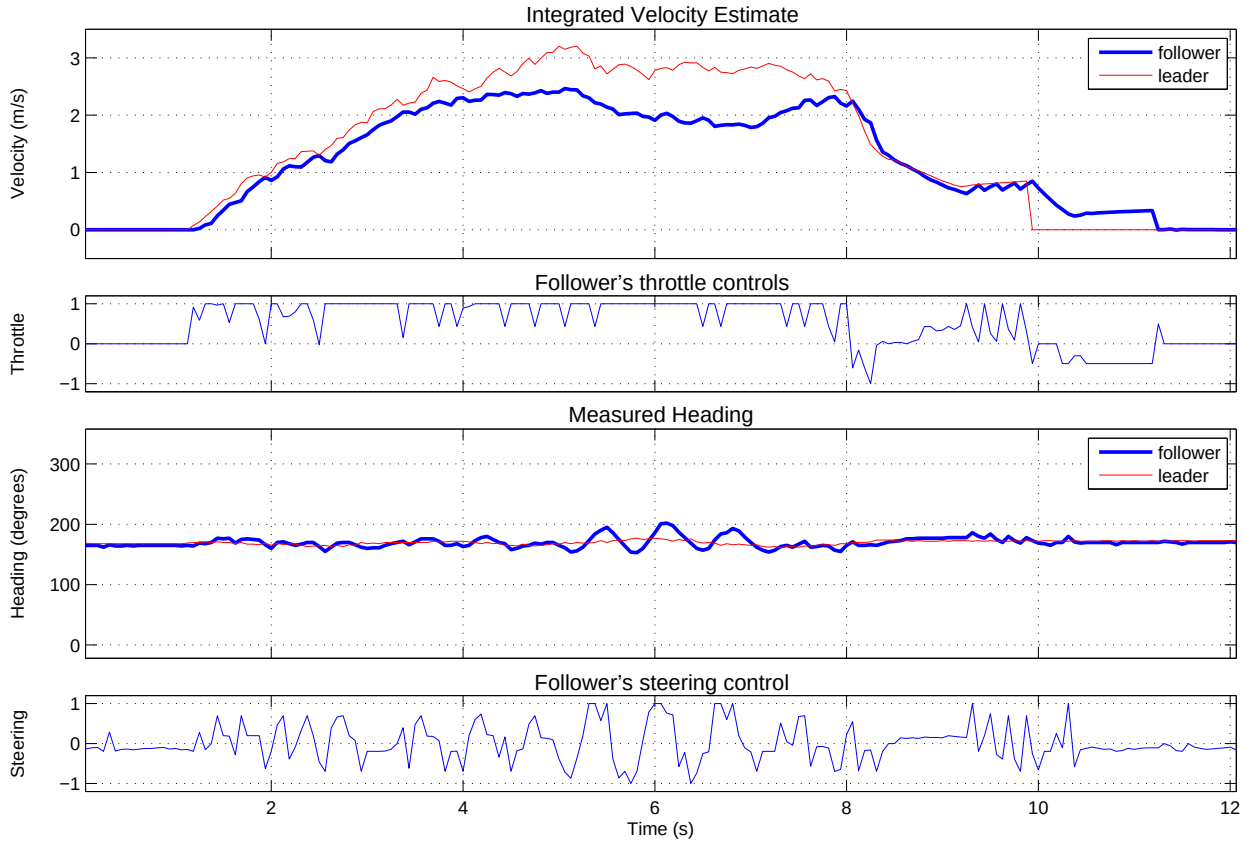


Figure 9.7: Velocity and heading logs of the tenth linear test.

it initially traveled less distance. However, it stopped about a second later, giving it ample time to catch up. Something similar holds for the final heading: after the leader stops, its heading remains constant, which gives the follower's heading controller the ability to stabilize to that heading.

The follower failed to stop instantly after the leader stopped, because the leader's velocity integration accumulated error. At the instant the leader actually stopped, after a little more than 9 seconds since the beginning of the test, its velocity estimate was not zero. The motion detection technique we describe in Section 4.3.2 reset the velocity to zero 750 ms later. That is when the follower first tries to actually stop: first by applying bursts of negative throttle and then by activating the brake. The brake control is not shown in this figure, but as soon it is activated the throttle signal is preempted. This means that the negative continuous throttle visible in the final stages of the follower's stop was not actually applied on the vehicle's drive motor.

According to Figure 9.7, the follower stopped at about 10.5 seconds into the test. Much like the leader, the velocity integration does not present a zero velocity at that time. Again, velocity does not become zero until it is reset by the motion detector a little less than a second later. However, the error accumulated by the follower is less significant than the leader's error. Interestingly, that is observed in most tests and, if the follower does accumulate error, it is negative most of the time.

Figure 9.8 shows another linear test run which performed better when only the correlation between the velocity measurements of the cars is considered. Also, the velocity integration of both vehicles returned to a value very close to zero when the vehicles stopped. This caused leader and follower to stop at almost the same instant. However, the follower did not maintain the leader's velocity all the time due to the same problems as for the test we presented before: the leader was driving a little too fast and the follower was swaying heavily in response to relatively small deviations in the leader's heading.

Due to the failure of the follower to mimic the leader's velocity continuously, the follower fell behind. And for this test, the follower did not make up for this increase in distance by driving further for a while after the leader stopped already. That is why the leader and follower were 5 m further apart at the end of this

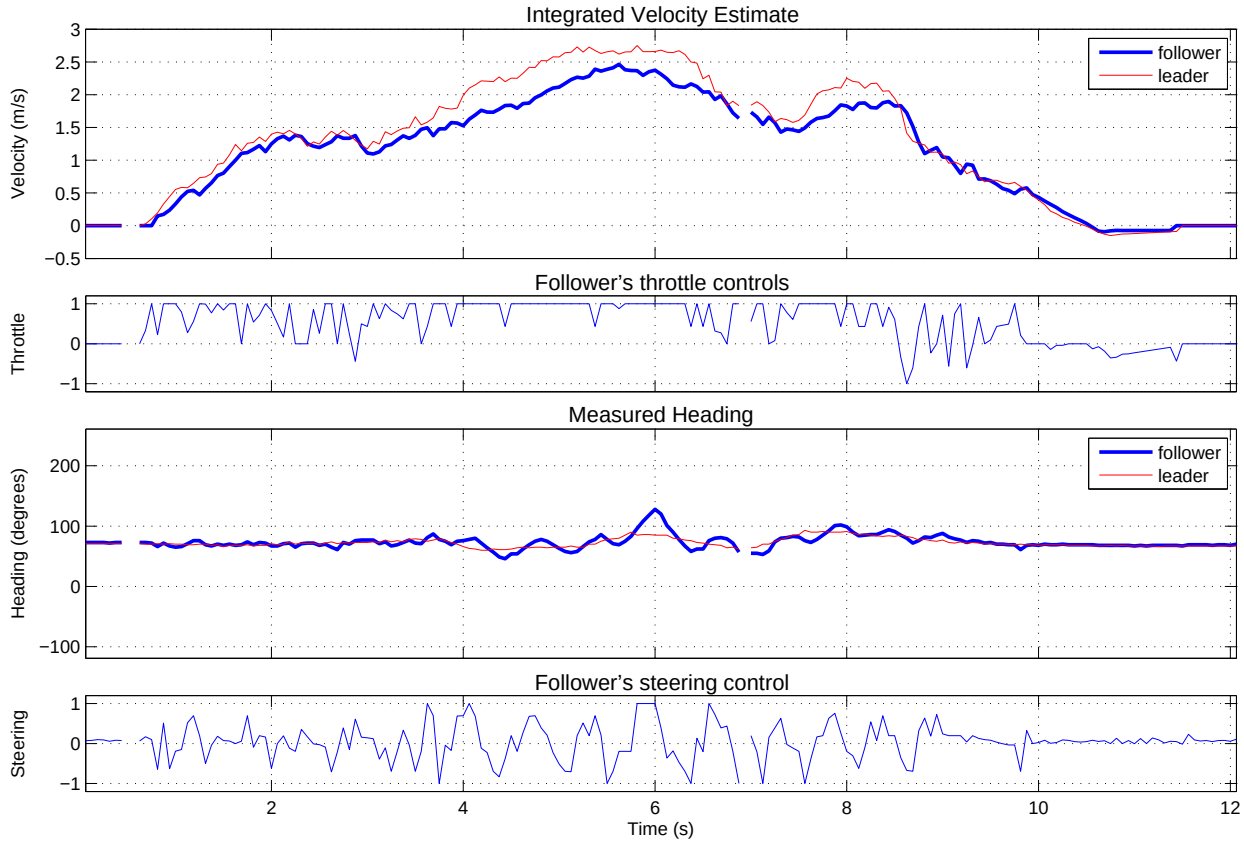


Figure 9.8: Velocity and heading logs of the thirteenth linear test.

test. The leader crossed the finish line by 2.10 m, while the follower did not even reach it for another 3.60 m. In contrast, this test performed better than the previously described test for the final difference in heading: less than 3° .

A full list of all measurements performed after the linear tests is shown in Table 9.2. The *Distance* column shows the distance between the vehicles at the end of each test. This distance varies between 30 cm and more than 9 m with an average of 3.7 meters. Since the vehicles start at a distance of about 1.8 m, they deviate on average by 1.9 m over the traveled distance. Although the leader was supposed to drive 20 m, it drove 1.7 m further on average, meaning that the follower deviates by an amount of $\frac{1.9}{20+1.7} \approx 9$ cm per traveled meter on average.

According to the table, the follower deviates 8° on average when traveling that 22 m distance. Most tests result in a final heading difference below 9° , but test 11 is a notable exception. During this test, the leader made a large swaying movement with a deviation of close to 40° from the designated heading. After the swaying movement performed by the leader, the follower continued making large swaying movements for another 7 seconds until the leader stopped. The follower's heading deviated from the leader's heading with peaks of up to 25° .

The log of this test is shown in Figure 9.9. The figure suggests that the leader's initial sway caused a self-sustaining oscillation in the follower's steering control, because the follower continues the swaying motion until it is forced to stop. During that time the leader's heading changed only gradually and was relatively smooth, thus that is not likely to be the cause of the control instability.

The swaying motion ended when the leader stopped. This happened when the controller was steering back from a large deviation. The follower's controller activated the brake of the follower at the exact moment the follower's heading reached the opposite peak. Consequently, the test ended with a large heading difference. Notice the steering control after both vehicles have stopped: it is at an extreme value because the steering controller tries to compensate for the final heading difference. But, since the vehicle is not driving anymore, this has no effect.

Test	Distance (m)	Heading (°)			Distance past Finish (m)	
		Leader	Follower	Diff.	Leader	Follower
1	5.4	295	292	3	2.5	8.3
2	0.3	301	290	11	2.8	2.8
3	1.0	290	281	8	0.8	1.4
4	9.2	292	309	17	1.0	10.2
5	0.4	309	307	3	3.2	3.4
6	4.7	292	284	8	1.4	-3.1
7	3.6	292	281	11	-0.1	-3.5
8	3.5	298	292	6	0.8	4.2
9	1.5	332	343	11	2.6	3.4
10	1.3	281	276	6	0.1	0.3
11	4.4	186	152	34	3.4	7.3
12	6.1	217	219	3	1.4	6.3
13	6.3	194	197	3	2.1	-3.6
14	5.0	219	214	6	0.2	4.2
15	1.6	231	225	6	1.4	2.8
16	3.5	225	225	0	1.8	-1.1
17	1.9	200	202	3	2.6	2.8
18	6.8	169	169	0	2.4	4.3
19	5.6	152	160	8	3.3	4.8
20	1.1	194	186	8	1.0	0.5
Avg.	**3.7**			**8**	**1.7**	**2.8**

Table 9.2: Test results from driving a linear path over a distance of 20m.

Figure 9.9 shows how quick a combination of friction, negative throttle and the inductive brake can stop the follower car when it is driving about 1.4 m/s. At the instant the leader notices that it has stopped, the follower starts to slow down. The throttle control shows no output most of the time, meaning that the follower is stopped solely by friction and the inductive brake. Braking takes almost one second in total, making the deceleration 1.4 m/s^2. This yields a delay between the leader stopping and the follower stopping of almost two seconds total.

Our braking strategy assures that the follower stops quickly when it needs to, because the controller is only allowed to control the vehicle within a limited interval after the leader has stopped. After that, no throttle is applied anymore, making sure that the car stops on friction if controlled methods like negative throttle and the brake fail. During testing, the application of the brake can clearly be heard and always results in the follower stopping within seconds when leader remains stationary. However, if the follower accumulates very significant error in the velocity integration it sometimes drives in the opposite direction over a short distance, but it always stops within a few seconds.

Interestingly, the final heading difference in Figure 9.9 is shown to be about 33°. This correctly matches our final measurement in Table 9.2 using the mechanical compass. This suggests that the difference that the vehicles measure during driving would correctly match the difference measured using our mechanical compass, assuming the electronic compasses are properly calibrated. However, as we show in the next section for the square tests, this is not certain.

To provide a more pictorial idea of how the cars ended up after each test, Figure 9.10 shows a schematic view of the vehicles in their final position for each test. The y axis of the plot shows the position of the vehicles relative to the finish line. The finish line is depicted as a thick line at distance zero. The individual tests are lined up horizontally. Keep in mind that the plot only shows the perpendicular distance of the respective cars to the finish line. This does not relate to the distance between the cars directly, because the cars were not directly behind each other most of the tests. Rather, they moved side to side to prevent the follower from clashing into the back of the leader if it would accelerate too enthusiastically. The initial distance of 1.8 m is also the distance between the cars side to side, which means that the initial difference in their distance to the finish line is approximately zero.

The plot also shows the heading of the vehicles at the end of the tests. The lines emanating from the boxes that represent the cars indicate the heading of the vehicles. In Section 9.1.2, where we describe the test strategy, we explain that the linear tests are not all performed in the same direction. However, the plot shows all headings relative to the finish line of the respective tests.

Clearly, in the final position, the leader and follower point in the same general direction for most tests. The distance between leader and follower seems to be a more profound issue. This is not surprising: the compass provides an absolute heading reference to which the steering controller can always find its way. In contrast, the velocity integration can quickly lose track of the real velocity of the vehicles giving the controller invalid information to work with. Also, the influence the steering has on the friction causes the follower to fall

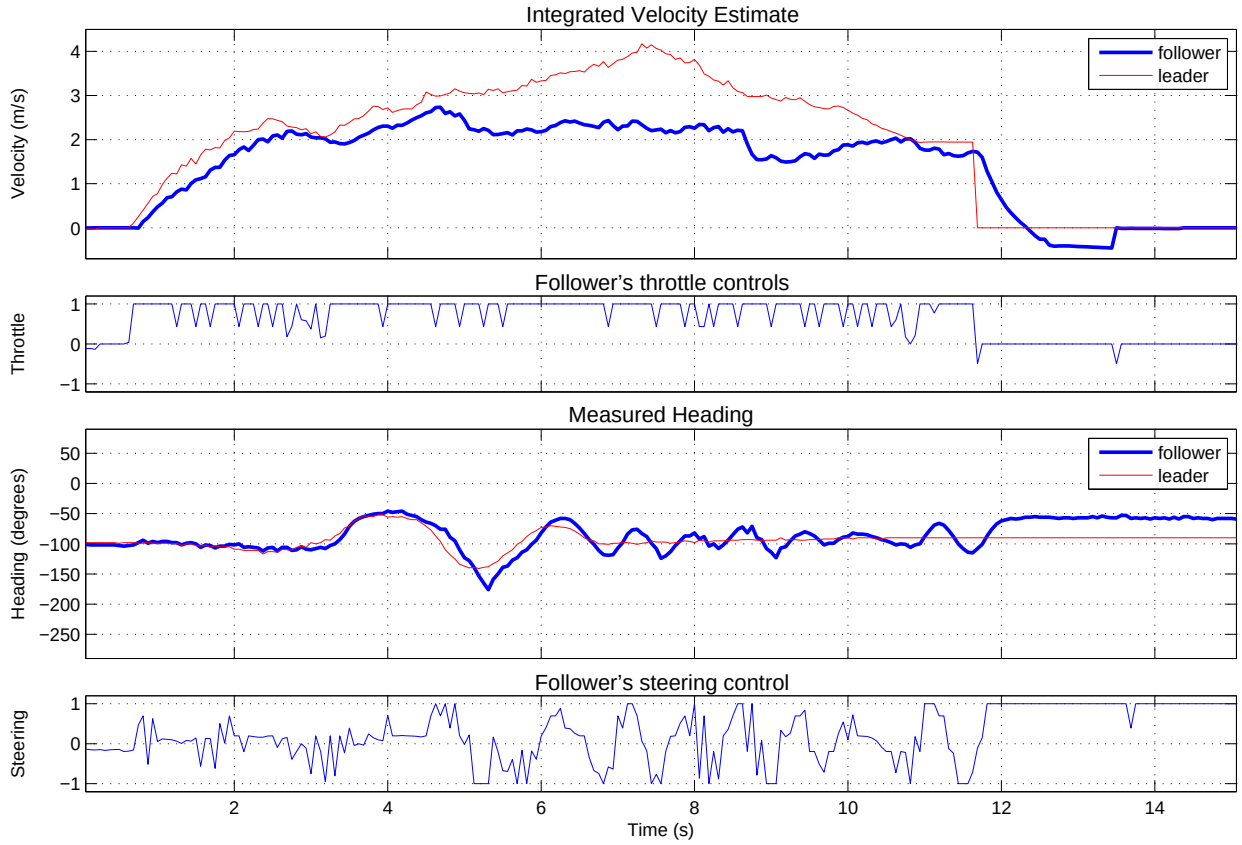


Figure 9.9: Velocity and heading logs of the eleventh linear test.

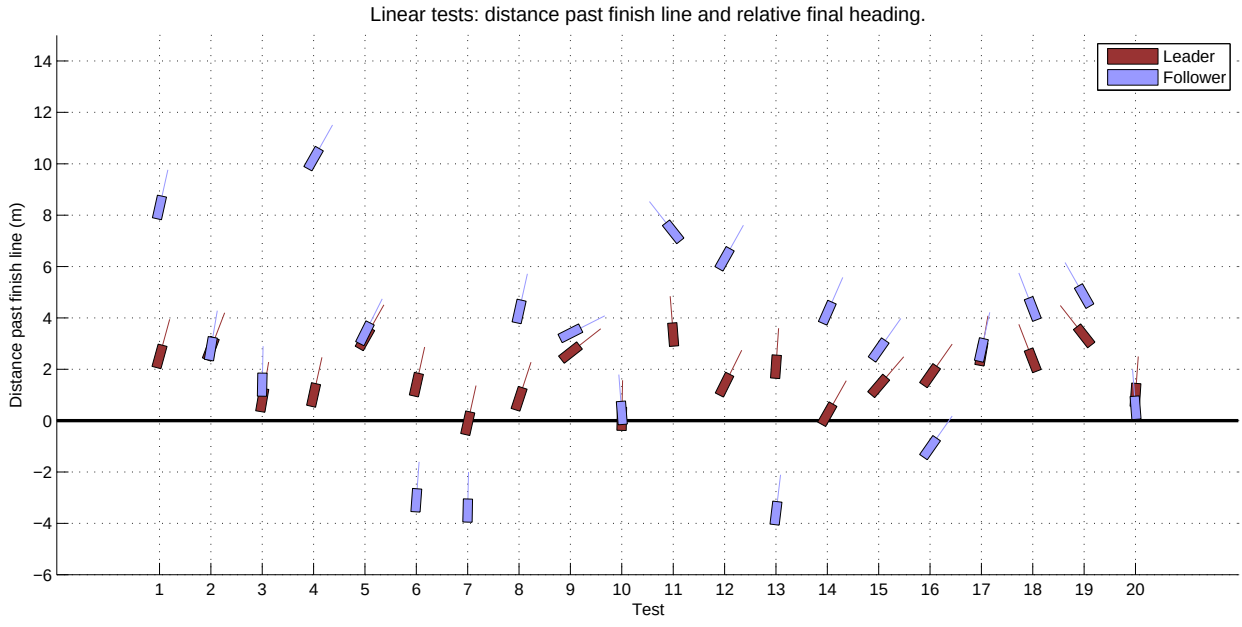


Figure 9.10: Graphical representation of the test results from driving a 20m linear path.

behind when the velocity of the leader is too high.

Reviewing the logs suggests that the velocity integration of the leader is more significantly affected by inclination changes than the follower. In Figures 9.8 and 9.9 for example, the leader ends up with a much larger integration error than the follower. After the leader has stopped, it still measures a significant constant forward acceleration yielding a linearly increasing velocity. This is a clear indication of an inclination change causing a gravity influence on the integration. This is observed in many more logs (also see subsequent sections on square and random tests).

Whenever the leader integrates its velocity to an erroneously high value due to gravity influence, the follower will try to attain that speed when possible. As we show above for Figure 9.8, this causes the follower to drive further for a while, even when the leader stopped seconds ago. This is unfortunately a very common phenomenon as is visible in Figure 9.10: all tests for which the follower traveled further than the leader are affected. Simulations and tests with correct velocity integration suggest that the follower's velocity controller never tries to exceed the leader's velocity (see also Section 7.2.3). Therefore, the follower can only end up having traveled farther than the leader when integration problems have occurred.

The linear tests are used to assess how well the follower can mimic a constant heading at varying velocity. In summary, we make the following observations for the linear tests:

- Final heading difference and distance are low on average: The final heading difference is 8° on average and the final distance is 3.7 m on average for the linear tests. Considering that the compass resolution is at best 1° and that the follower's steering is not very accurate, a final deviation of 8° is satisfying. Regarding distance, the follower deviates from the leader 9cm per traveled meter, meaning that it will deviate only 9 m over a stretch of 100 m.

- Logs show large swaying movements by the follower: Especially when the leader is driving at high velocity, the straight line tests show a follower that sways in a sinusoidal motion around the desired heading. On average, the desired heading is achieved, but the standard deviation is very high.

- Velocity of the leader easily exceeds the follower's maximum: Even though the leader is driven at very low throttle, it still exceeds the follower's capability at several occasions. This effect is worse when the follower is steering. This means that the swaying motion of the follower also worsens the velocity mimicking performance at high velocity.

- Velocity integration performance of the leader is often worse than the follower's: When the vehicles stop, their velocity must be zero, which is often not observed. That is when the velocity integration error becomes apparent. Somehow, the leader often shows a positive integration error which is more significant than the negative integration error that the follower presents in most cases.

- Velocity integration error at the leader causes a delayed follower stop: Because the leader's velocity estimate is not zero when it physically stops, the follower is not notified of the leader's stop until the motion detection at the leader resets the velocity estimate to zero. This happens less than a second later.

- The follower's brake manages to stop the vehicle reliably: When the follower is finally notified about the leader zero velocity, it starts to slow down. The brake provides a reliable means to definitively stop the vehicle. Forward-backward driving oscillations are not observed. However, if the follower has a significant positive velocity integration error, it will sometimes first try to drive backwards until the brake is actually applied. In any case, the follower stops reliably within a few seconds when it needs to.

- Errors incurred during driving can still yield pristine final results: Errors incurred during driving have a tendency to cancel each other out, potentially yielding very good distance and heading results at the end of the test. A very common situation is that the follower falls behind during the test, but makes up for the distance by driving up to a second longer. On the other hand, good performance during driving is sometimes negated by driving further past the stopped leader's position or a steering error just before the follower is forced to stop.

Square Tests

The linear tests we describe in the previous section are very simple as these do not include explicit steering activities by the leader. In this section we describe the results of the more complex square tests in which the leader is directed to drive a square with edges of approximately 15 m long. All tests involve the square pattern being driven in a clockwise manner.

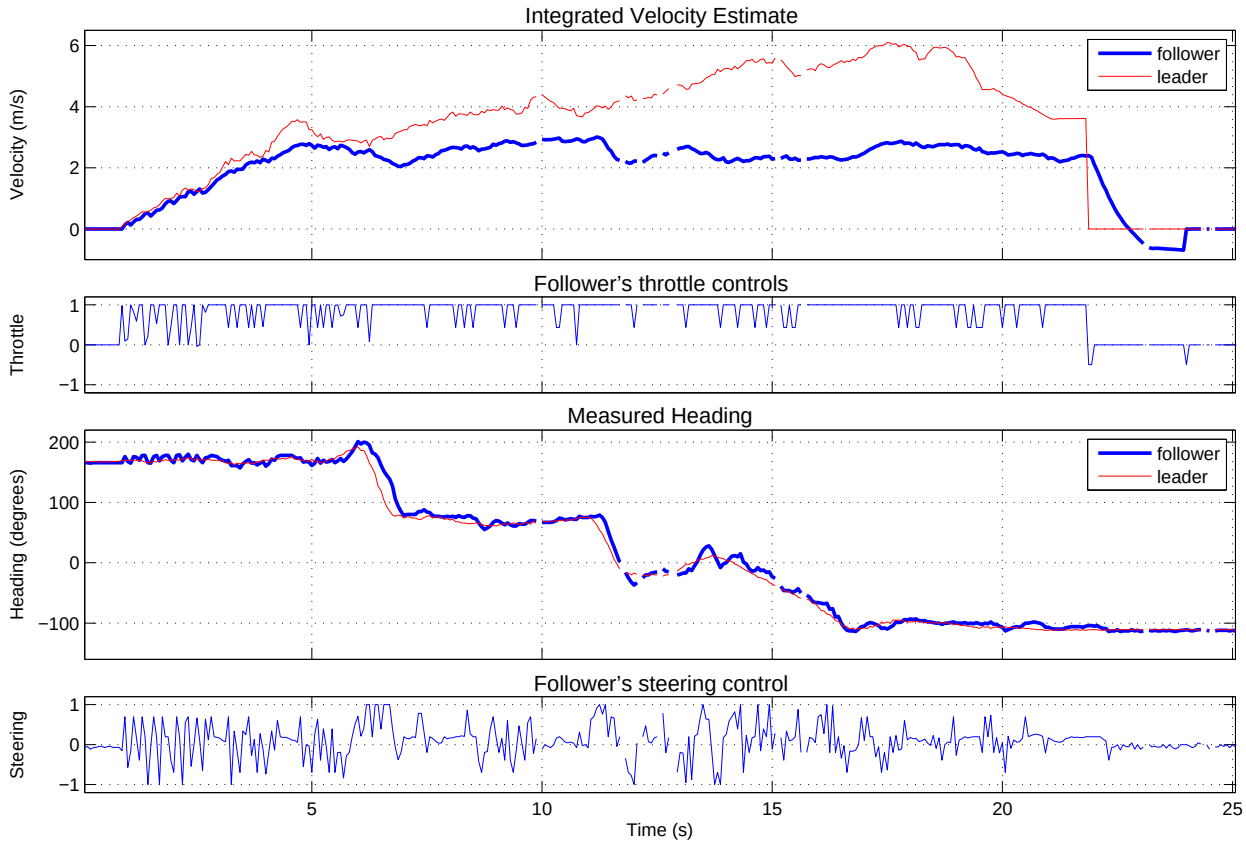


Figure 9.11: Velocity and heading logs of the sixth square test.

First, we show a relatively clean test log in Figure 9.11. The steering behavior of the leader is clearly visible as a heading change with somewhat regular intervals. The leader's heading increments at the turns range form 80° to 100° and the straight sections do not have a very constant heading, showing that it is not trivial to drive a proper square with our leader vehicle.

However, unlike all the linear tests we present in the previous section, no continuous swaying movement is observed for the follower most of the time. Notably, at 13 seconds after the start of the test the follower needs some time to stabilize towards the desired heading, but the deviation is relatively small when compared to the linear test we presented earlier. Overall, this is a good test with respect to the steering controller's behavior.

Judging by its final value and much like the linear test results we present in the previous section, the leader's velocity increments to an erroneous high value. In contrast, the follower returns to a velocity much closer to zero. As before, the large velocity offset at the leader causes the follower to accelerate towards its maximum velocity and, more annoyingly, it stops much later.

When compared to the linear tests of the previous section, the addition of regular heading change is observed to make the velocity integration incur more inclination effects. Table 9.3 shows the measurements performed right after each square test. The average distance between the vehicles becomes 8.3 m over a traveled distance of about 60 m. The vehicles still started at a distance of 1.8 m, meaning that the cars deviate by $\frac{8.3-1.8}{60} \approx 11$ cm per traveled meter. Interestingly, this is not the very significant increase that is to be expected with the high velocity errors observed at the leader.

Test	Distance (m)	Heading (°)			Distance past Finish (m)	
		Leader	Follower	Diff.	Leader	Follower
1	6.3	202	191	11	5.9	0.3
2	4.7	208	205	3	-2.2	2.1
3	2.4	200	186	14	-0.5	0.4
4	5.1	205	194	11	-0.6	4.0
5	5.2	194	191	3	0.7	5.2
6	10.7	211	200	11	-1.0	9.3
7	6.3	194	197	3	0.1	6.4
8	9.4	211	194	17	0.6	9.2
9	10.5	233	219	14	-1.6	7.4
10	10.9	188	191	3	0.2	11.0
11	6.0	191	197	6	-0.1	5.8
12	10.2	222	214	8	0.1	10.3
13	10.5	267	267	0	0.8	2.7
14	9.2	194	197	3	-0.3	8.4
15	10.7	211	191	20	-1.8	8.5
16	11.1	180	169	11	0.2	11.2
17	6.8	281	264	17	-0.3	0.0
18	9.3	259	281	22	2.6	6.3
19	9.7	211	222	11	2.5	10.6
20	12.0	186	231	45	3.1	15.1
Avg.	**8.3**			**12**	**0.4**	**6.7**

Table 9.3: Test results from driving a square path with edges of 15m.

The leader is driven at its lowest continuous velocity. At this velocity the follower can just keep up without problems. If the leader goes any faster, the follower will not be able to increase its velocity much more, especially in the corners. This has the added effect that the follower will not be able to speed up to the very high erroneous velocities that the leader's velocity integration sometimes produces. This fortunate effect limits the impact of the integration error on these test results.

Only at the end of a test the effect really becomes evident, because that is when the cars must stop: if the leader has accumulated much error in its velocity integration, the follower will not slow down until the motion detector at the leader resets the velocity to zero. This makes the follower drive further at full throttle for a while, causing it to deviate significantly from the leader. That is why almost all results of Table 9.3 show the follower much farther over the finish line than the leader. On average the distance between the vehicles with respect to the finish line has increased from 1.1 m to 6.3 m. In contrast, the error per traveled distance does not change much.

The discussion above suggests that the follower driving further after the leader has stopped is the predominant cause of error in the measurements performed after the tests. This was observed during testing. Often, the cars would visually follow each other within less than two meters, but when the leader stopped the follower could drive on for as much as 10 m. Actually, this 10 m is probably the maximum bound for this error, because the follower will stop closely after the motion detector resets the velocity of the leader to zero. As we explain for the linear tests, this happens with a bounded delay after the leader has stopped completely. To obtain a better idea of how the error increases with distance it is therefore important to perform tests in which the traveled distance is larger, making the bounded stopping error for the follower less significant.

Figure 9.12 shows the schematic overview of the vehicle positions and orientations at the end of the square test. Evidently, the follower was almost always much farther over the finish line than the leader. Like for the linear tests, the final heading is visually very similar. However, on average, the absolute heading error has increased from 8° to 12° .

Test 20 is the most significant outlier with a final deviation of 45° . This deviation was caused at the very end of the test: for an unobvious reason the follower applied full left steering and steered far beyond the desired heading and stopped closely after. Unfortunately, the associated log for test 20 is not very clear. That is why we present the log for test 17 in Figure 9.13 as an example for a test with much less promising results than the test presented in Figure 9.11.

As shown, the follower sways heavily in this test again with deviations ranging up to about 30° from the leader's heading. This swaying motion is also the cause for the final deviation of 17° listed in Table 9.3. And although this difference matched the log for the linear test we describe in the previous section, it is very much different in the log for square test 17: the log reports an angle difference of 42° . This can be attributed to a measurement error with the mechanical compass, but more likely something peculiar is going on with the electronic compass sensor. Note that the initial heading in the log is correctly close to identical as both vehicles have the same initial heading.

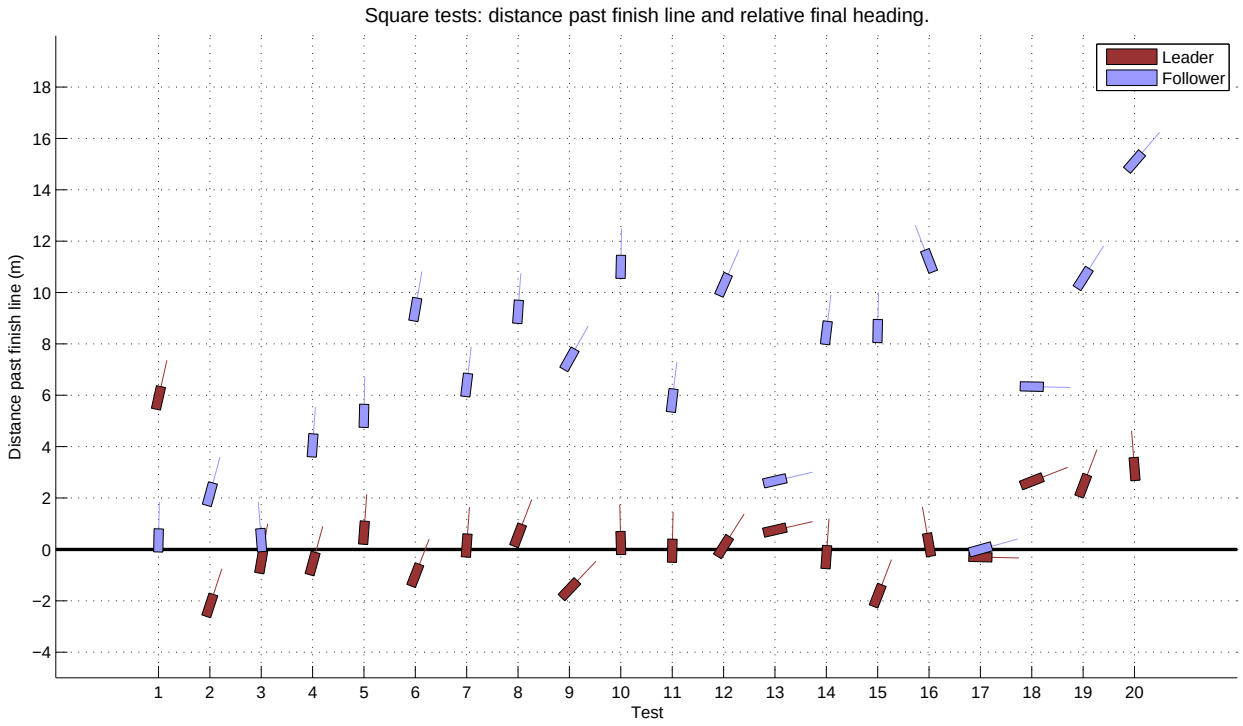


Figure 9.12: Tests results from driving a square with 15m long sides.

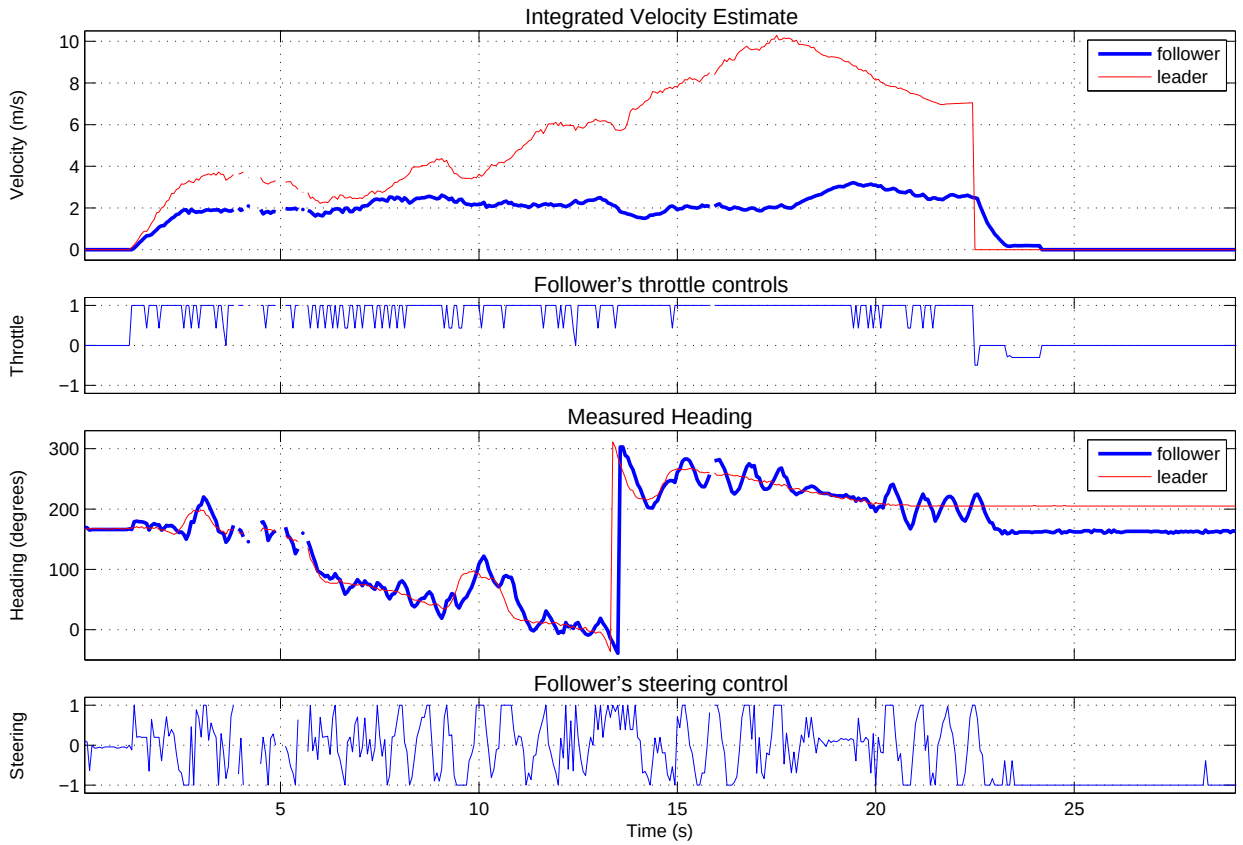


Figure 9.13: Velocity and heading logs of the seventeenth square test.

The trigonometric discontinuity after 13 s provides a clear indication of the delay between the leader action and the follower response. The plot shows a delay of about 200 ms and the heading of the follower lags behind about 25° . This is substantially more than the 63 ms (16 Hz) and 5° we attributed to the controller's response time in Section 6.2. This probably means that the leader steered too quickly for the follower to follow.

The velocity integration of the leader is severely compromised again for test 17. In contrast, the integration of the follower's velocity returns to a value very close to zero and the velocity during driving is shown to be very constant. Just like in Figure 9.18, the increasing error in the leader's velocity in Figure 9.13 seems to start at its first steering activity and progresses linearly from thereon.

Many of the observations from the square tests are identical to what we have seen for the linear tests. The square tests yield the following additional observations:

- The follower mimics the leader's heading and velocity: The follower manages to stay close to the leader within a few meters during most of the tests. However, the leader's velocity integration problems make it stop later at the end causing the finish distance result to be rather large: at the end the average distance between the nodes is 8.3 m. Relative to the linear tests, the final heading error increases from 8° to 12° .
- The addition of sudden aggressive steering makes velocity integration deteriorate: Especially the leader more often yields velocity integration results that range far beyond its true capability. In many cases, the error is seen to become significant just after the car steered significantly. The follower seems less affected by steering in this respect.
- The leader's large velocity integration errors do not necessarily cause distance errors: The follower can not attain the erroneously high velocity often reported by the leader, meaning that it will only accelerate to its maximum velocity, which is in many cases close to what the leader is actually driving at. Therefore, the follower will not overhaul the leader and follow it at close distance, even though the leader's reported velocity is completely wrong. However, if the leader's velocity error were negative, the follower would quickly fall behind. But, the leader almost always presents a very high positive velocity error when driving forward.
- The square tests also show much swaying by the follower: Especially at the linear sections of the square pattern, the follower often sways heavily around the desired heading. Close to the steering operations, the swaying temporarily subsides to return again when the leader's heading stabilizes.
- It is hard to keep the leader on a designated drive pattern: Although this section describes driving square patterns, the real path driven by the leader during these tests is not exactly square with relatively much over-steering and corrections. The leader did however travel past the corner markers laid out for this test, meaning that the overall pattern can be considered square.

Random Tests

The square and linear tests are very structured, as we try to achieve a series of designated headings with the cars without stopping. The test results we present here, in contrast, are random. The first ten random tests involved steering in random directions, while the last ten tests also involved large variations in the leader's velocity. During all tests, the leader was stopped within 30 seconds.

Figure 9.14 shows the log of a random test in which the leader was driven at a very constant speed with a randomly changing heading. The log initially suffers from a few instances of packet loss, but after 10 seconds the log is clean.

Considering the final state, the velocity integration performed very well for the follower as it returned to a value very close to zero. The leader, on the other hand, accumulated an inclination error in its velocity again. The leader seems to start accumulating error at the moment it accelerates to its constant velocity. In comparison to the linear and square tests we present in the previous sections, this random test seems more stable regarding the steering performance. Only when the follower needs to stabilize towards a constant heading, it makes swaying movements that disappear quickly. At 12 and 17 seconds after the start of the test the sway is still significant though with a deviation of close to 20° .

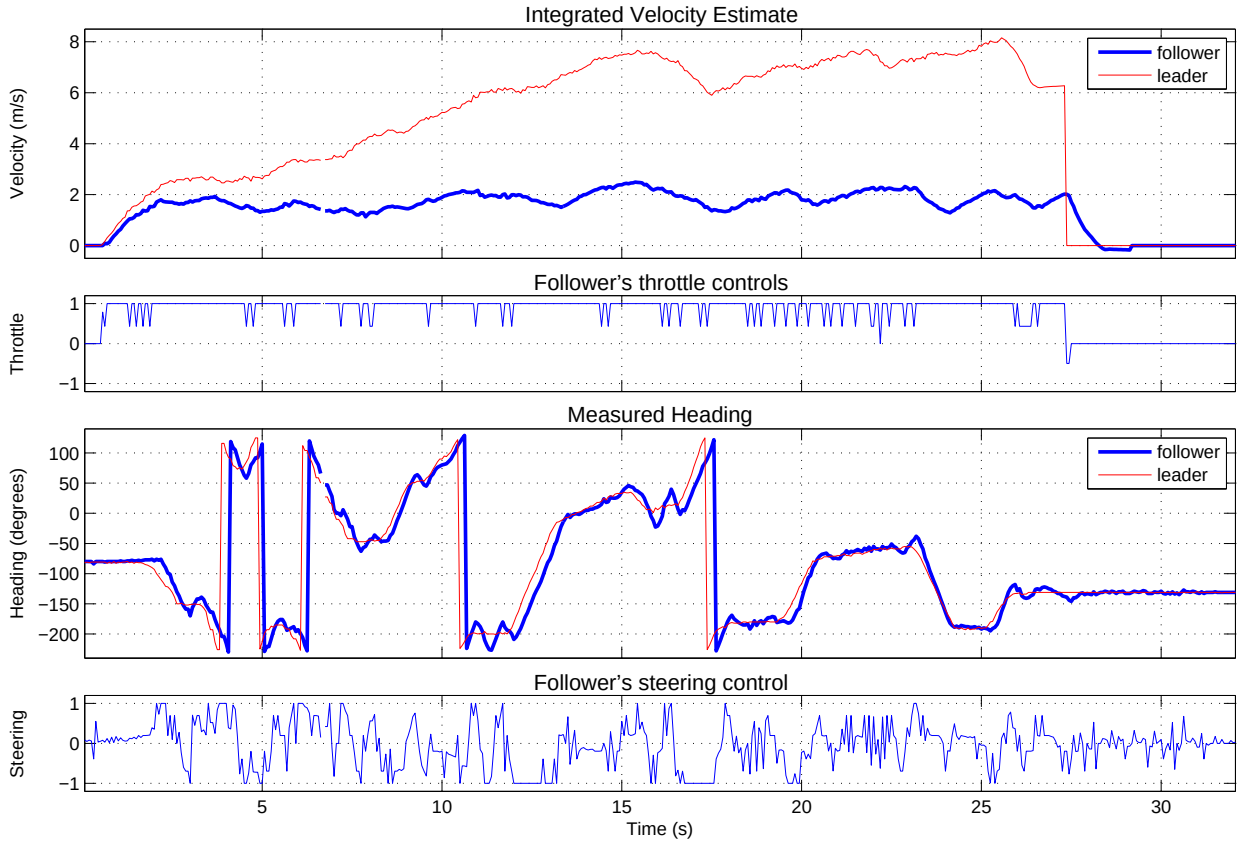


Figure 9.14: Velocity and heading logs of the seventh random test.

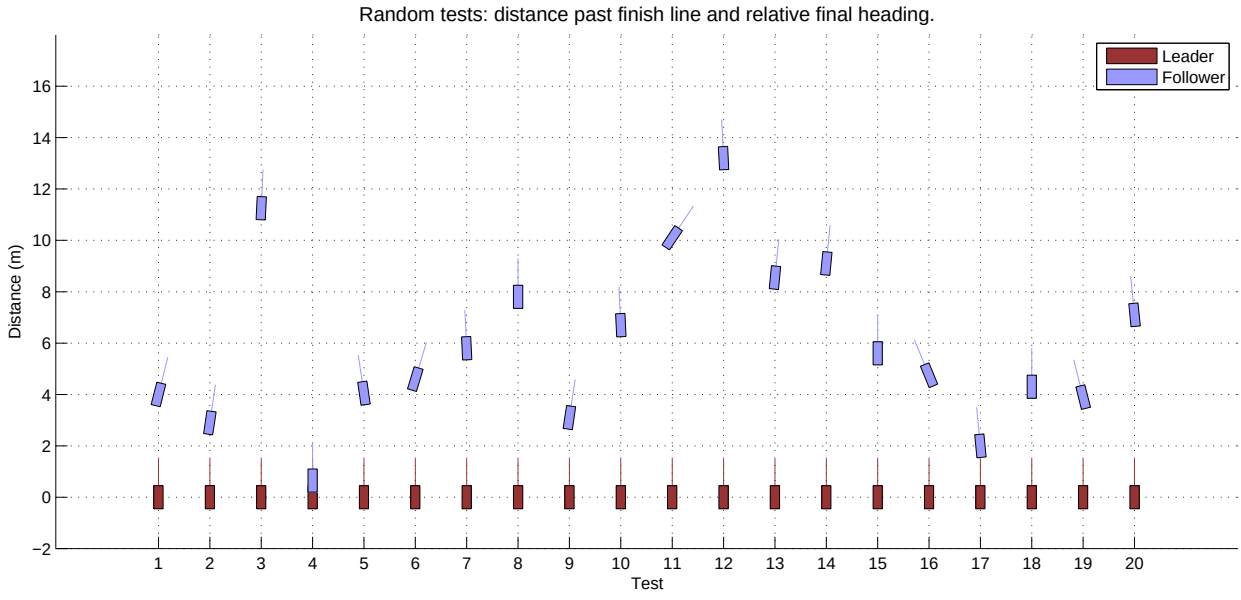


Figure 9.15: Graphical representation of the test results from driving a random path for 30 seconds.

Test	Distance (m)	Heading (°)		
		Leader	Follower	Diff.
1	4.0	112	127	14
2	2.9	158	166	8
3	11.2	202	205	3
4	0.7	273	273	0
5	4.0	112	104	8
6	4.6	163	180	17
7	5.8	219	217	3
8	7.8	217	217	0
9	3.1	338	346	8
10	6.7	284	281	3
11	10.1	79	112	34
12	13.2	166	163	3
13	8.6	354	0	6
14	9.1	152	158	6
15	5.6	124	124	0
16	4.8	236	214	22
17	2.0	214	208	6
18	4.3	143	143	0
19	3.9	177	163	14
20	7.1	298	292	6
Avg.	**6.0**			**8**

Table 9.4: Test results from driving random path for a duration of 30 seconds.

Table 9.4 shows the measurements performed after each random test. The seventh test is the one we show in Figure 9.14. The final heading result for these tests is good, with a deviation of only 3° . Interestingly, the final heading was indistinguishable for the cars on various occasions and the average deviation is relatively low at 8° when compared to the square tests which ended at a deviation of 12° on average. The average final heading performance of the random tests is identical to the linear tests. The final distance between the cars is also better on average for the random tests than the square tests, even though the distance traveled during most random tests is estimated to be longer than the square tests: most square tests took only 20 seconds at similar velocities. Figure 9.15 provides a quick view of the final situations as observed when each test ended. Unlike the linear and square tests, the random tests do not involve driving towards a finish line. That is why no finish line is plotted in the figure. The headings and distances are all plotted relative to the leader, meaning that the leader is always plotted at distance zero with a straight forward heading.

The last ten random tests involved the leader making significant velocity changes. Figure 9.16 provides the resulting log of one of these tests. The integration of the leader's velocity does not show a large offset at the end in this case. Interestingly, the follower's velocity offset is larger and negative. The negative offset suggests that the follower was driving faster than it thought it was. The follower's velocity plot approaches the velocity logged from the leader, with the peaks in the leader's velocity as exceptions. Apparently, the follower could not accelerate that quickly to achieve those velocity peaks. This would result in the follower falling behind. However, keep in mind that the follower's velocity probably exceeded the leader's velocity at the end of the test, considering the negative offset. Also, the follower stopped later than the leader again. These factors, when combined, result in a final distance between the cars of 2 m, which is approximately the same distance as the cars started at.

Figure 9.17 shows the leader intermittently driving and stopping. It drives four times and turns each time. As shown, the leader accelerates very quickly and the follower fails to keep up when the leader's velocity spikes. Interestingly, the leader's velocity integration often returns to zero, while the follower always ends up with a negative offset. The second and third time that the leader stopped the both nodes did not manage to reset their velocity estimate to zero. This happened because the leader started driving again before the motion detectors could determine that the cars were standing still.

Apart from the fact that the test results we present in Figure 9.16 and 9.17 show that the follower can mimic the leader's velocity changes, it is also seems as though the changing velocity has a positive effect on the stability of the steering control. The first random test we present in this section was performed at a more or less constant speed and still has some stability problems. The test we present in Figure 9.17 is much cleaner in that respect.

Surprisingly, the random tests seem to show better performance for the follower car than the linear and square test. The net performance is predominantly better, because the follower swayed much less. When the follower does not sway as much it can attain a higher velocity, due to the reduced friction. Consequently, it can keep up better with the very agile leader car. Of course, if the follower's steering quickly stabilizes towards the desired heading, the heading deviation is smaller overall as well.

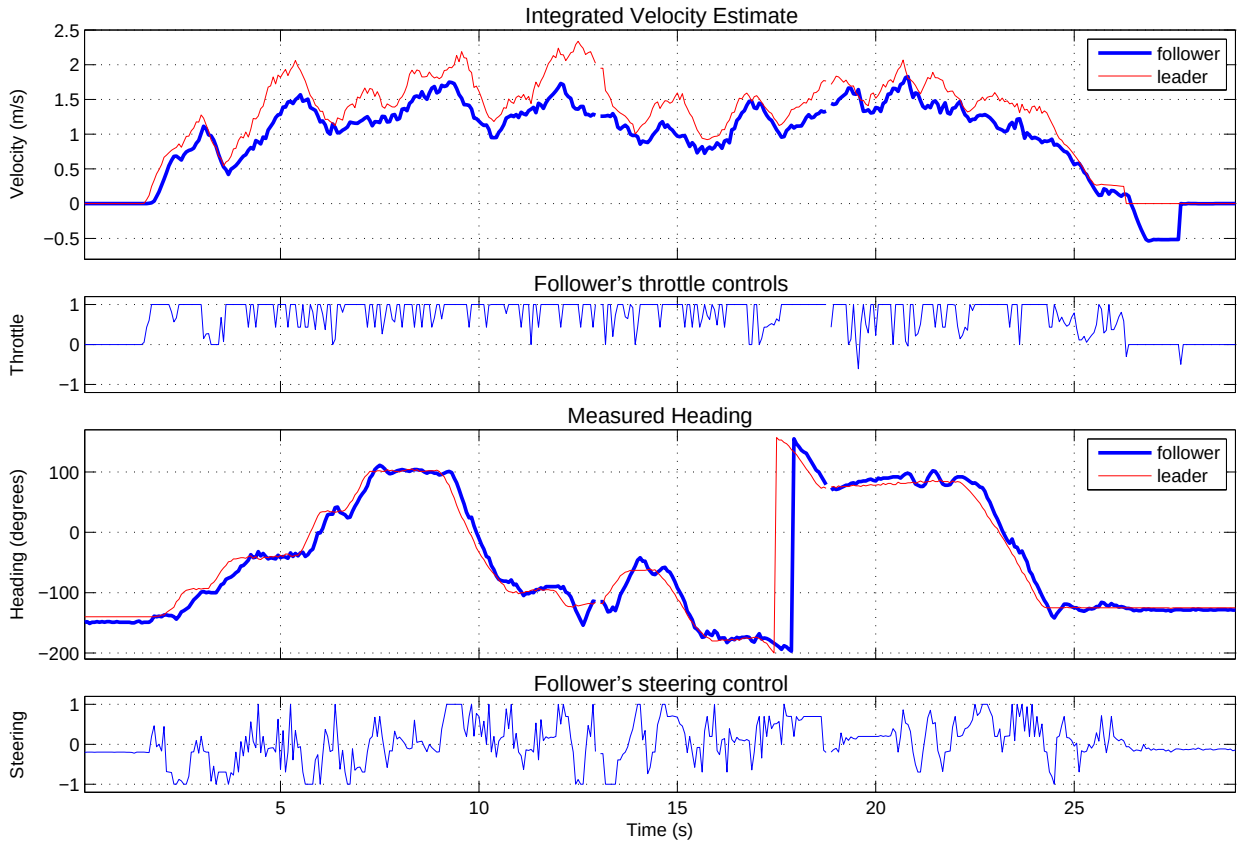


Figure 9.16: Velocity and heading logs of the seventeenth random test.

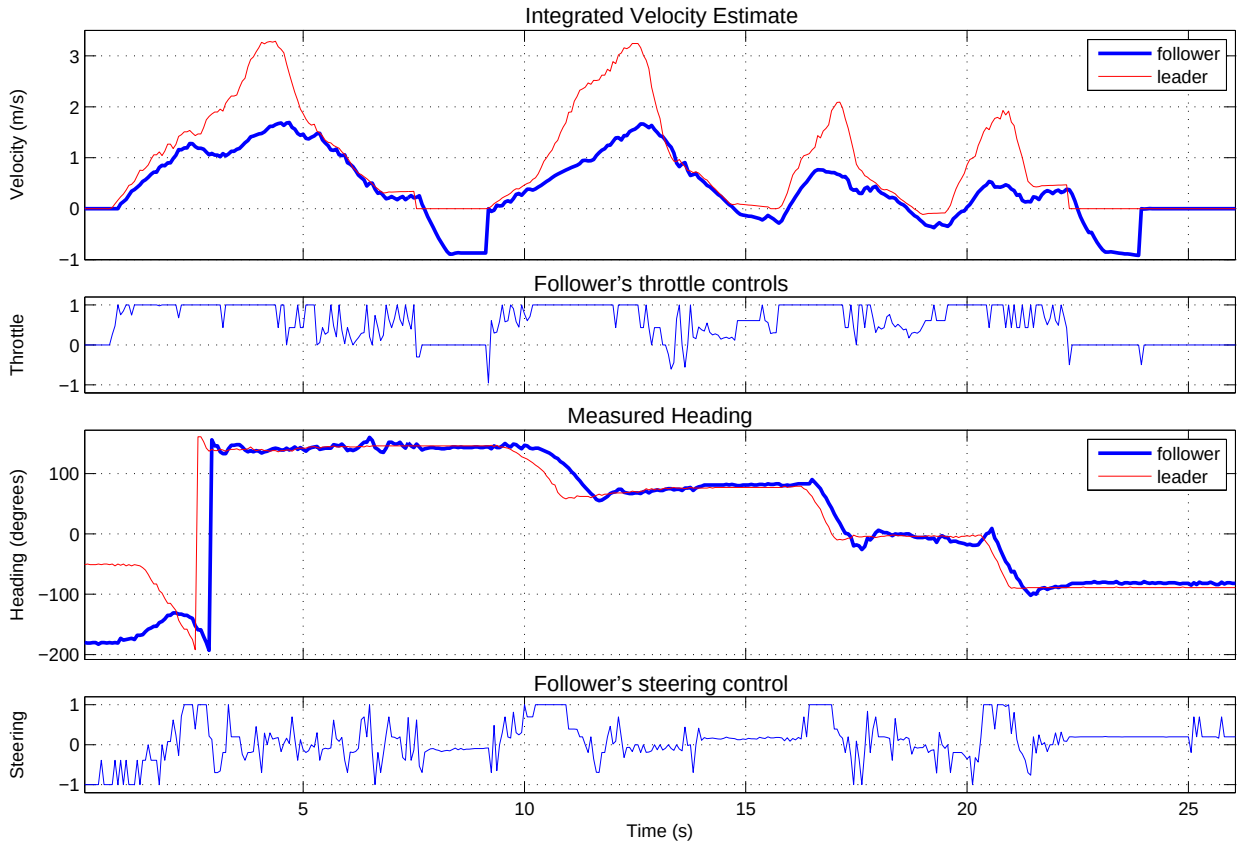


Figure 9.17: Velocity and heading logs of the nineteenth random test.

Additionally, the frequent heading changes during the random tests did not result in a noticeable decline in the performance of the velocity integration as one might expect. Visually, the random tests even perform better than the linear and square tests. One unrecorded test was performed for 10 minutes during which the follower managed to continuously maintain a distance of no more than a few meters. Yet, much like the linear and square tests, the leader suffers from a very significant inclination influence and the follower returns to a value close to zero. The significance of these errors did increase much when compared to the square tests. However, as for the other tests before, the limited follower velocity reduces the effect of this error on the distances between the vehicles. Interestingly, the tests with changing velocity show that the follower incurs an increase in the velocity offset when it needs to change its velocity frequently. Unlike the leader, this offset is almost always negative.

In summary, we make the following observations for the random tests:

- Visually, the follower mimics the leader's movement surprisingly well: Contrary to what is to be expected for the unstructured movements performed during the random tests, the follower often tracks the leader's path really well at a distance of only a few meters.
- Judging by the end deviations, the follower performs better than in the square tests: Surprisingly, the average final heading deviation is only 8° and the average final distance is only 6 m, both better than the square test performance. Also, the final heading performance is mostly comparable to the linear tests. Thus, considering the final deviations, the very frequent heading and velocity changes do not result in a performance decrease.
- The follower sways much less than in the linear and square tests: The swaying steering behavior seen for the linear and square tests is mostly absent for the random test. The frequent heading changes by the leader incite frequent steering operations in the follower that somehow counter the instabilities it otherwise shows. Furthermore, changing velocity also seems to have a positive effect on the stability of the heading controller: initial tests at very constant velocity still show limited swaying while the final tests that include large velocity changes are mostly clean.
- The follower's velocity integration seems more sensitive to velocity changes than the leader's: Tests that involve a sequence of alternating acceleration and deceleration in many cases show better velocity integration performance for the leader than for the follower. This is contrary to the tests where the velocity is constant.

Velocity Error Accumulation

For many of the hockey field tests we observe velocity integration errors that become explicitly evident at the end of the test when vehicles have stopped and their respective motion detectors reset the velocity estimate to zero. The cause of this error is primarily the accumulation of gravity influence on the acceleration measurements. Especially the leader seems prone to regular inclination changes.

As an example, we consider the 6^{th} square test we presented earlier in Figure 9.11. If the offset in the velocity integration is estimated to be a linearly increasing value that is added to the true velocity integration, one could imagine subtracting that value from the plot of Figure 9.11. This assumes that the gravity influence is incurred at a single point in time and remains constant after that. Using this correction, we could obtain a better idea of how the cars fared in reality.

Figure 9.18 shows this procedure graphically. The top plot is identical to what is displayed for the velocity in Figure 9.11. Additionally, this plot includes loose estimations for the accumulating velocity integration error. The error is assumed to emerge first when the leader enters its first corner. This is intuitively a logical instant at which the inclination of the vehicles can change and it is also the moment in the original velocity graph where the integrated velocity of the two vehicles starts to change significantly.

Even though the error estimations are largely based on assumptions, subtracting these leads to very intuitive results as shown in the bottom part of Figure 9.18. During this particular test, the cars were observed to have matching velocities, meaning that the leader was not driving much faster than the follower. The corrected velocity graphs show velocities that match much better than the raw logs. Also, the velocity of the leader is much more constant in stead of increasing. The fact that the follower stopped much later than the leader is clearer in the corrected plot as well.

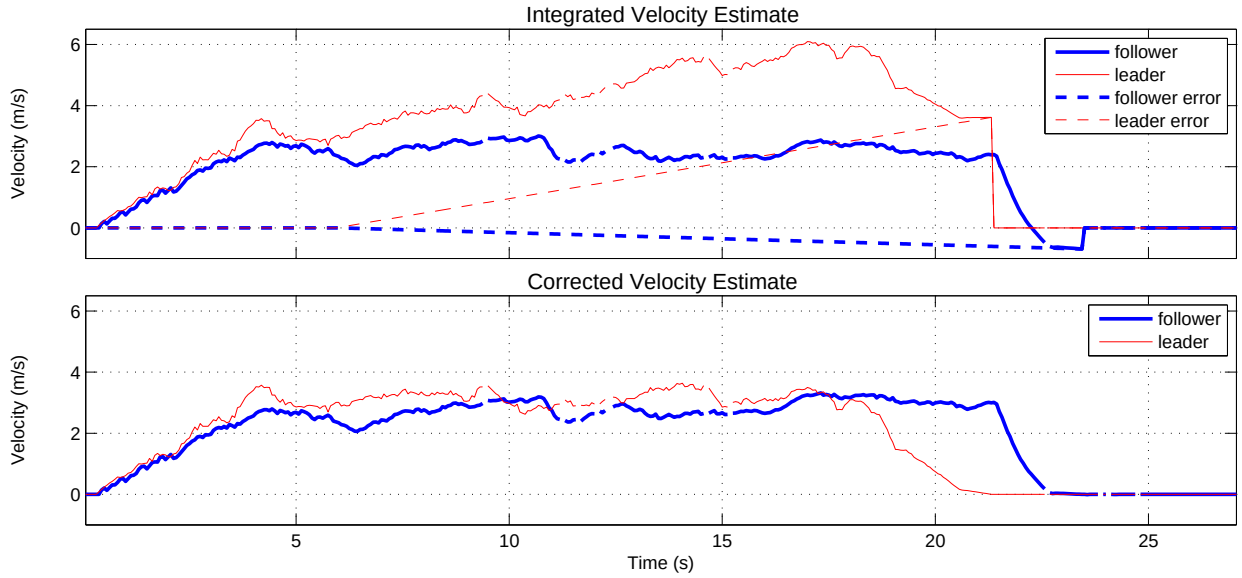


Figure 9.18: Uncorrected and corrected velocity logs of the sixth square test.

This result raises the question whether this correction can be applied nodes to in the field while they are driving. This would be a very difficult procedure. First, estimations of the error are mostly based on the final velocity difference when the cars are known to stand still. When driving, that information is not available. Second, we assume the velocity error progression to be linear and, although that seems to be a valid assumption for the presented test, this is not going to be valid in general.

It is possible to notice inclination changes by keeping track of the average acceleration on all three axes of the accelerometer. However, it is not clear how the calibration of the accelerometer should be adjusted based on the averages when a change is noticed, possibly using a heuristic method. And, since the average is also influenced by actual acceleration by the vehicle, the use of the average is not a very reliable means to check and compensate for inclination changes. For example, as we show in Section 4.3.2, a significant continuous centripetal acceleration results from circular motion. Additionally, this averaging technique will notice changes with a considerable delay, meaning that inclination errors still have ample opportunity to distort the velocity estimate.

Overview

To provide an overview of the hockey field results, we first provide comparative histograms in Figure 9.19. Histograms are provided for the final distance between the nodes and the difference between their respective headings. The figure shows a separate histogram for each drive pattern performed on the hockey field. The initial relative distance of 2m between cars is not subtracted from the presented distance values, therefore a distance of 2 m is the optimum.

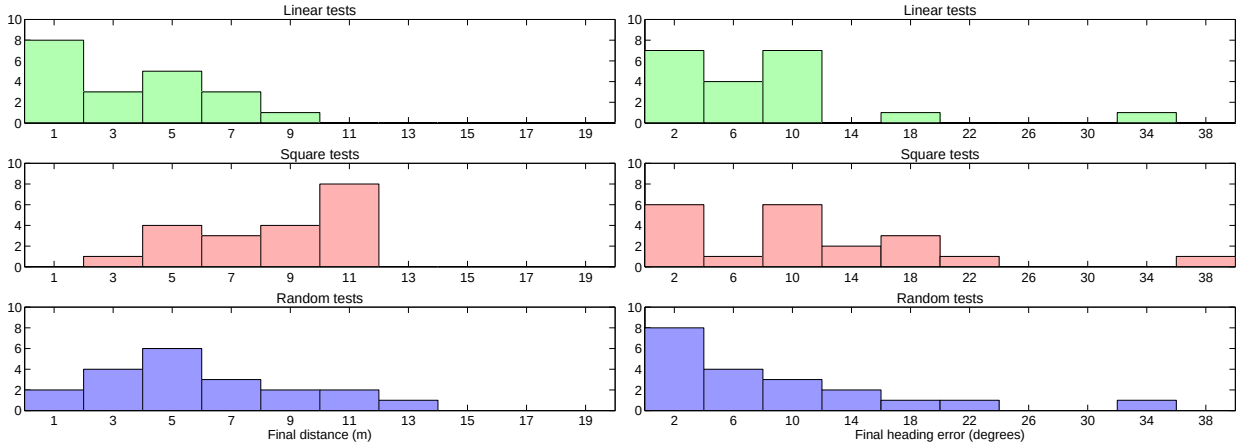
Regarding the final distance between the nodes, we see in Figure 9.19(a) that the distribution is very different for the linear, square and random tests. The linear tests clearly peak around 1 m, while the square tests peak around 11 m and the random tests peak around 5 m. Clearly, the random tests perform better than the square tests when only the final results are considered. And, while the distribution of distances smaller than 9 m is comparable for random and square tests, the distribution for the larger distances is very different: the square tests show many final distances close to 10 m, while these are not observed very often for the random tests.

The heading error histograms of Figure 9.19(b) show the contrast between the performance of the square and linear tests and the random tests. On the one hand, the error distribution for the random tests clearly shows a preference for low heading errors, on the other hand, the square and linear tests either have errors close to 2° or close to 10° . In this respect the linear and square tests are very similar. This is probably a direct result of the swaying behavior that the follower shows. All tests show significant outliers around 35

	Linear	Square	Random
Velocity (m/s):			
Mean Absolute Error	0.53	1.97	2.75
Standard Deviation	0.55	1.42	1.93
Correlation Coefficient	0.93	0.63	0.54
Heading (°):			
Mean Absolute Error	7.38	10.23	14.99
Standard Deviation	9.79	14.23	21.06
Correlation Coefficient	0.64	0.99	0.99

Table 9.5: Hockey field log error and correlation statistics.

degrees.



(a) Distribution of final distance

(b) Distribution of final heading error

Figure 9.19: Histograms for the hockey field test results.

Figure 9.20 summarizes these results into a single plot. The error bars are plotted with 5 and 95 percentiles, removing the outliers. Clearly, when only the final measurements are considered, the random tests perform better than the square tests at both fronts.

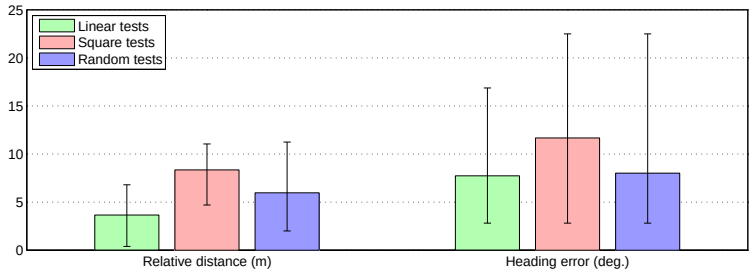


Figure 9.20: Summary of hockey field results.

All previous discussion only deals with the measurements performed after the tests. As we describe in Section 9.1.2, we do not perform absolute measurements during the tests. In stead, we record logs of the node's own measurements on their movements. In the previous sections, we described these logs only graphically. Here, we extend our discussion to statistics extracted from these logs.

Table 9.5 summarizes these statistics. For both velocity and heading, we compute the mean and standard deviation of the absolute error between the two vehicles at any moment in time, as well as the correlation coefficient of the signals recorded in each test. The (sample) correlation coefficient is a commonly used

measure of similarity between two signals. It is a value between -1 and 1: a correlation coefficient close to 1 indicates well matching signals, a value close to zero indicates absolutely no relationship between the signals and a value close to -1 indicates very good but inverse correlation between the signals. The sample correlation coefficient R on input vectors X and Y with equal length n is commonly defined as follows [29] (with $\bar{X}$ and $\bar{Y}$ being the average of these signals)[2]:

$$R = \frac{\frac{1}{n}\sum_{i=1}^{n} X_i Y_i - \bar{X}\bar{Y}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{9.1}$$

We make the following observations:

- The log statistics only reflect the perception of the nodes: In contrast to the results of the measurements performed at the end of each test (from Figure 9.20), the values in Table 9.5 offer just the perception that the nodes had about their velocity and heading. More specifically, these values include the inherent sensor and integration errors and therefore these measurements cannot be considered absolute or very reliable.
- The velocity errors are very significant when compared to the finish distance results: When the velocity errors are integrated over the course of the average test duration, distance errors ranging up to 60 m are to be expected. This is not observed for the measurements obtained at the end of each test. This is caused by the very significant velocity integration errors accumulated by the leader. The follower is less affected, causing a significant difference in the velocity perceived by the nodes. Because the follower cannot fully act upon the erroneous information, i.e. it cannot achieve the desired erroneous leader velocity, the errors are not fully propagated into the end results.
- The heading correlation coefficient in linear tests is smaller than in square and random tests: The linear tests show a correlation coefficient of only 0.64 compared to 0.99 for the square and random tests. The reason is that the heading varies much less in linear tests, as there are no turns, and therefore any small difference in the compass readings of the two nodes has a strong impact on the correlation. Recall that the linear tests show significant swaying. That is why the correlation coefficient for the linear tests is relatively low compared to the square tests, that also show swaying but much more variation, and the random tests, that show much less swaying.
- The performance in random tests is slightly worse than indicated by the ending results: A detailed analysis of the logs reveals two main reasons. Firstly, during random tests, the leader is subjected to steep acceleration and steering at times. Due to constructive limitations, the follower can not match this behavior, but later on he recovers the difference. Secondly, the vehicles drive longer distances in random tests than in other tests and thus accumulate higher errors through acceleration integration. However, their velocity is physically limited, which explains why the actual ending results are better than what the sensor nodes compute.

9.2.3 Summary

Throughout this section, we describe the results of tests at our university's running track and hockey field. In summary, these tests yield the following observations:

- The follower mimics the leader's heading and velocity: On the running track we show that the heading controller adequately achieves the desired heading when the compass is calibrated and mounted correctly. During the hockey field tests, the follower manages to stay close to the leader within a few meters most of the time. The follower clearly tries to mimic the leader's heading and velocity. The overall performance of the random tests on the hockey field is better than the square tests and comparable to the linear tests, with the added benefit that the follower sways less in the random tests.
- The communication protocol works adequately: Statistics of the packet loss observed at the gateway node shows that almost 99% of the messages is received from both nodes, meaning that our communication protocol works adequately and that the follower manages to properly schedule the transmission of its message among its other software tasks (refer to Section 8.2).

[2]We use the MatLab implementation of this function (corr2).

- The follower sways heavily on straight sections: Especially when the leader is driving at high velocity, the tests on the hockey field that include driving straight lines show a follower that sways in a sinusoidal motion around the desired heading. On average, the desired heading is achieved, but the standard deviation is very high. On the running track this effect was also observed just after a 180° turn, but there it quickly subsided to a less noticeable swaying motion.

- The compass is very sensitive to external influence and inclination changes: If the compass' position, orientation or inclination relative to the vehicle changes even slightly, the calibration becomes inadequate and the sensor needs to be re-calibrated. Inclination changes of the whole vehicle cause noise on the compass sensor readings and large nearby metal objects cause significant measurement errors. Both these effects can cause the follower to make sudden inadequate steering motions.

- Errors can cancel each other out: Despite the integration errors at the leader and the swaying behavior of the follower, the latter still manages to keep on the right trajectory, especially when the leader is driven with frequent changes in heading. This effect is shown both during the random tests and in an additional test that lasted for more than 10 minutes. This is partly explained by the fact that errors tend to compensate for one another. In addition, the limited velocity of the vehicles improves the overall performance. When the leader does not drive faster than what the follower can manage the latter follows correctly, even when the estimate of the leader velocity rises beyond its constructive possibilities.

- The velocity integration is very sensitive to inclination changes: When the vehicles stop, the end velocity is often not zero, showing the presence of integration errors. When the raw acceleration data is reviewed an often very constant acceleration offset is observed that emerges at some point during the test. This causes an often linear increase in the velocity integration error. This is a clear indication of inclination changes causing gravity influence on the velocity integration. The leader performs worse in this respect during most tests, especially when it is steering frequently.

- The follower often stops seconds later: We notice that the follower generally ends up further relatively to the finish line than the leader (the random tests are irrelevant in this case, as there is no finish line). This is caused by the errors accumulating in the acceleration integration process and the inherent delay of the motion detection technique (see Section 4.3.2).

- The follower's brake works reliably: When the follower needs to stop, it can do so within one second from high velocities. Also, a failing velocity controller cannot make the follower accelerate or drive backwards indefinitely, because there is a time limit within which the controller is allowed to stop the vehicle.

- The leader's relatively high agility influences the tests: First, it is hard to keep our very agile leader at a velocity that the follower can match and to steer it at a angular rate the follower can handle. This often causes the follower to fall behind or steer slower. In those cases, it could never have achieved the desired movement, even with perfect control. This effect has both positive and negative effects on the results: it counters the effect of large velocity integration errors on the leader, but steep accelerations and quick steering operations cannot be followed yielding suboptimal test results. Second, the leader is hard to control manually due to its agility: the real path driven during the linear and square tests is not exactly what was intended, with relatively much over-steering and corrections.

9.2.4 Discussion

The test results from the hockey field suggest that the follower performs better when the leader regularly changes its heading. The resulting continuous steering behavior of the follower seems to counter the steering instability observed for the square and linear tests. However, swaying or not, the follower does follow the heading of the leader in general. In the logs we present in the preceding sections this is seen as the follower's heading graph oscillating around the header's heading graph. Something similar, albeit less significant, is observed at the running track: the follower frequently steers too much, but overall it maintains the designated heading.

One possible cause for the swaying behavior of the follower is the fact that nearby metal objects can cause a significant influence on the compasses. Although the hockey field itself is void of any metal objects, the cars could influence each other when they approach each other closely. Our current movement mimicking strategy causes our vehicles to change their relative orientation when the leader changes heading. If the

inter-car influence is significant, this could cause both leader and follower to measure an erroneous heading, triggering an inadequate controller response. On the running track no leader was present and there it would be able to drive straight lines without much swaying, other than right after steering or when hitting debris or surface irregularities. However, the swaying behavior of the follower is not only observed when leader and follower are close together. In the tests where the cars accumulated significant distance (> 5 m) during driving, the swaying behavior was still present.

The other important difference between the hockey field and running track tests is that a velocity controller is included and that, consequently, the velocity of the follower varies with that of the leader. This can be indicative of an interaction between the heading and velocity behavior of the follower car. Possibly the model we define for the steering behavior in Section 6.4 is too simplistic to build a stable heading controller. Particularly, the steering response is assumed to be proportional and close to instant, which is an optimistic assumption at best. Also, the angular velocity of the vehicle resulting from a specific steering intensity depends partly on the vehicle's current forward velocity. Obviously, when the vehicle is standing still, it will not change its heading, no matter what the angle of the front wheels is. This dependence is only used in our models when the vehicle's velocity is close to zero.

Regarding the velocity integration, the tests on the hockey field show that this technique can perform well as long as the inclination of the vehicles does not change during driving. Particularly the follower frequently manages to end up with a velocity estimate that is correctly close to zero when it stops. Even after the complex movements performed during the random tests and the quick swaying movements the follower sometimes performs, the follower's velocity integration performs surprisingly well.

In contrast, the leader performs much worse on several occasions. We attribute this to the very soft suspension that the leader car has and the weaker mounting of the sensor on the leader. Each individual wheel is equipped with spring dampeners, giving the vehicle's hull the freedom to tilt a little in any direction. As the sensor node with the accelerometer is attached to this movable hull, it can incur significant inclination changes as the car moves. For example, when the vehicle steers left at a significant velocity, its inertia will make it sink a little into the suspension springs of its right wheels. This causes a short change in the inclination of the vehicle's hull. Also, because the sensor's attachment is not very rigid it can maintain this change in inclination by leaning to the side a little.

The follower vehicle is much sturdier in this respect. It only has a spring suspension on its front wheels and the sensors are mounted rigidly on the vehicle. Additionally, the construction of the suspension does not allow for the wheels to move individually with respect to the vehicle's chassis. This means that the follower can predominantly only change its inclination among its front and rear and not sideways. We think that is why the follower's velocity integration performance only declines significantly when it needs to accelerate and decelerate frequently as performed during the last ten random tests: when accelerating it the front of the car will lift a little and when decelerating it will sink into the front suspension a little. Steering movements are much less problematic for the follower.

Another interesting fact to note is that the follower's integration error is negative most of the time, whereas that error for the leader is large and positive for most tests. This suggests a significant dependence on the construction of the vehicle and the mounting of the sensors: in most random tests the follower performs very similar movements as the leader and still the leader's integration error ends up positive and the follower's integration error ends up negative. Also, the sometimes very large swaying movements the follower performs in the linear tests show no significant effect on the performance of the velocity integration: the leader performs worse in most cases. The relatively weak mounting of the sensors on the leader may very well be the main problem for the integration performance and this should be mended for future versions.

In Section 9.2.2 we describe interesting properties of the integration error caused by changing the vehicle's inclination. Since the inclination influence is caused by gravity, its influence is constant as long as the inclination does not change further. When the vehicle finally stops, the integration error is visible as an offset from the zero velocity at the end of the test. Also, when the vehicle drives at a relatively constant velocity, the integration error is often seen to progress linearly, suggesting that its influence on the measured acceleration can remain constant for long periods of time during driving. Whether this type of integration error can be identified and compensated for at runtime remains an open problem.

Apart from the integration-related problems, the follower's velocity controller works good enough to keep up with the leader's velocity changes as long as the leader is not driving or steering too fast. However, an interesting thing to note from the log plots shown in this section is that the follower never tries to exceed

the leader's velocity even a little. For a really good velocity match, the follower must produce a velocity response that is equal to the velocity of the leader on average. This means that the follower is allowed to have a velocity that lies below the leader's velocity every once in a while, as long as it exceeds the leader's velocity just as much on other occasions.

As we show for our velocity simulations in Section 7.2.3, we think this behavior is directly caused by the fuzzy controller. If the velocity gets closer to the desired value, its throttle output will approach zero. As we describe in Section 6.5.5, the friction the car incurs makes the control problem asymmetric. If the controller outputs a very small throttle control value when the velocity approaches the desired value, the friction will cause it to be ineffective to accelerate further to remove the small final velocity difference. The margin between the desired and the achieved velocity is greatly determined by the width of the fuzzy input membership functions, since that determines what velocity difference is necessary to yield a significant throttle output.

An issue resulting from velocity integration errors is the often relatively large interval between the leader's stop and the follower's stop. Due to the integration errors on the leader, the leader's velocity estimate often does not reach zero after a stop until it is reset by the motion detection. This makes the follower maintain a high velocity during that motion detection interval. Luckily, the inductive brake on the follower performs adequately, making it stop within 0.5 m at full velocity when the leader's stop is finally detected. Consequently, the maximum distance traveled by the follower after the leader has stopped is determined by the time needed to detect the stand-still. Tuning the motion detection to make this interval smaller is not a real solution to the problem however; the velocity integration needs to be improved to make it less prone to inclination effects.

The measurements collected at the end of the tests are arguably not fully indicative of good or bad performance. Errors have a tendency to cancel each other out and just before the end of a test the follower can make significant errors, yielding results that do not reflect the reality of the test. Visual observation during the test is a means to provide only a subjective assessment of the follower's performance. We increase the value of the measurements collected at the end of the tests by amending these with the measurements collected by the vehicles themselves. This, however, only shows how well the follower itself thought it was following the unreliably measured movements of the leader. Using correlation and error statistics on this data, we can show how well the controller manages to achieve a desired value during the tests. But, whether or not the follower truly matched the leader's heading and velocity during driving is not clear from these logged measurements.

Visually, the follower can perform surprisingly well, especially during the random tests where follower's swaying behavior subsides. The measurements collected at the end of each test partially show a different story with relatively large final distances. This is primarily caused by the velocity integration errors making the follower drive further for a few seconds. Additionally, the swaying behavior of the follower causes large final heading errors in multiple linear and square tests on the hockey field. The statistics extracted from the collected logs show that the measured movements of leader and follower correlate well in general, although the results are worse for the linear tests due to the large amount of swaying when compared to controlled heading changes. To obtain truly conclusive results, some means is necessary to keep track of the real distance and heading difference during a test, possibly using a camera.

Contrary to the measurements collected after each test, the statistics extracted for the logs of the random tests show that the average error is higher than the linear tests. This is caused primarily by the longer test duration and the fact that the leader was driven more erratically and aggressively during the random tests. This shows how the agility of the leader car can influence the test results. For future work, it is a good choice to use identical cars to make a more valid evaluation of the follower's performance: if the leader cannot exceed the follower's capability, this cannot influence the results. Also, it is advisable to find a set of cars that are less agile in general, because it is very difficult to follow a designated pattern using a toy car that is obviously designed for racing. Alternatively, the size of the drive pattern can be increased to make the significance of driving errors smaller. Keep in mind, however, that this system should be applicable to any wheeled set of vehicles: using identical cars serves only as a means to isolate the true controller-related issues.

The swaying steering behavior of the follower on straight sections needs attention, probably by defining additional or different input values to account for the influence of velocity on the rate of steering. Something similar is necessary to improve the velocity controller. Both in simulation and during actual tests the follower does not attempt to achieve the full velocity of the leader. This is caused by the fact that the

current velocity controller design does not properly account for the asymmetric nature of the velocity control problem. Designing a fuzzy controller with an additional input that indicates whether the vehicle needs to accelerate or slow down provides the proper means to make acceleration more aggressive and deceleration use the benefit of friction. The current velocity controller only uses the velocity error which gives no indication of this state and the asymmetric nature of the velocity control is mended by inelegantly subtracting a constant value from the throttle control when the vehicle is trying to slow down.

Summarizing, our measurement method provides the means to assess the performance of the system after a period of driving. This shows to what extent the controller manages to achieve its goal of staying close to the leader by mimicking its movements. It does however not provide an absolute reference on how the cars performed during a test, giving error cancellation a chance to distort the results. With the inclusion of statistics gathered from measurements performed by the cars themselves during driving, we feel confident however that we can conclude that our system presents the desired behavior, albeit with some significant problems at leader and follower pertaining to data collection and (manual) control. First of all, the compass sensor is a very sensitive device that fails to achieve the desired accuracy if improperly calibrated, when inclination changes significantly or in the presence of large metal objects in the vicinity. Second, integrating acceleration towards a velocity estimate is very sensitive to inclination changes, causing significant velocity errors and a large delay between the leader stopping and the follower doing the same. Third, instability of the controller at the follower causes it to sway when driving straight lines. And finally, the agility of our leader vehicle makes testing more difficult, as driving a designated pattern is hard and the follower cannot always keep up. To isolate problems directly relating to the controller, two identical cars should be used.

9.3 Simulation Validation

In Chapter 7 we present a set of simulations for the vehicle's controller. To asses whether these simulations are realistic enough to test the performance of the fuzzy controller, it is necessary to compare simulated results with the results of actual runs with the two toy cars.

We present the results of our field tests in Section 9.2. That discussion makes use of the data that was recorded at the gateway during each test. In this section, this logged data is used as input for the full simulation of Chapter 7. Thus, effectively, the controller has the same input data from the leader in the simulation as during the actual tests. This makes comparison between the simulated and the actual response of the controller possible.

In summary, we use our field test results to validate our simulation model. However, we only provide a relatively limited comparison; we describe only the most interesting logs described in Section 9.2. To provide a more concrete idea of how well the simulation and the real follower match up, we provide error and correlation statistics similar to those presented in Table 9.5 in Section 9.2.2 (page 123). Although the simulation can be tuned freely, its configuration is kept constant for all simulations presented here, as not to bias our results.

The first test we compare with the simulation is the test we show in Figure 9.8 on page 110. Figure 9.21 shows the results of the simulation. Clearly, the simulated velocity approaches the real follower's response. Only at about 5 seconds the simulated velocity deviates significantly from what was recorded from the real follower. Interestingly, at that moment the largest sway in the follower's steering behavior occurs. This sway does not occur in the simulation. Therefore, we would expect the real follower to attain a lower velocity at that moment, but that is not the case. Either the influence of the steering friction is more complex than what we modeled or the simulation parameter for the steering friction has a too high value.

The much less significant heading deviations in our simulation can confirm that our simulation model is not adequate to simulate the vehicle accurate enough to reproduce this erroneous controller behavior. However, the simple fact that the simulation also yields sinusoidal deviations rather suggests that the simulation can be tuned towards a more realistic response. The amplitude of the simulated deviations is smaller but the deviations occur with a higher frequency.

The simulation results presented in Figure 9.22 seem to confirm this suggestion. Again, the simulated velocity is very close to the real response and the simulated heading is much more stable. Interestingly, the sinusoidal deviations in the simulated and real heading often occur at the same time, indicating that these

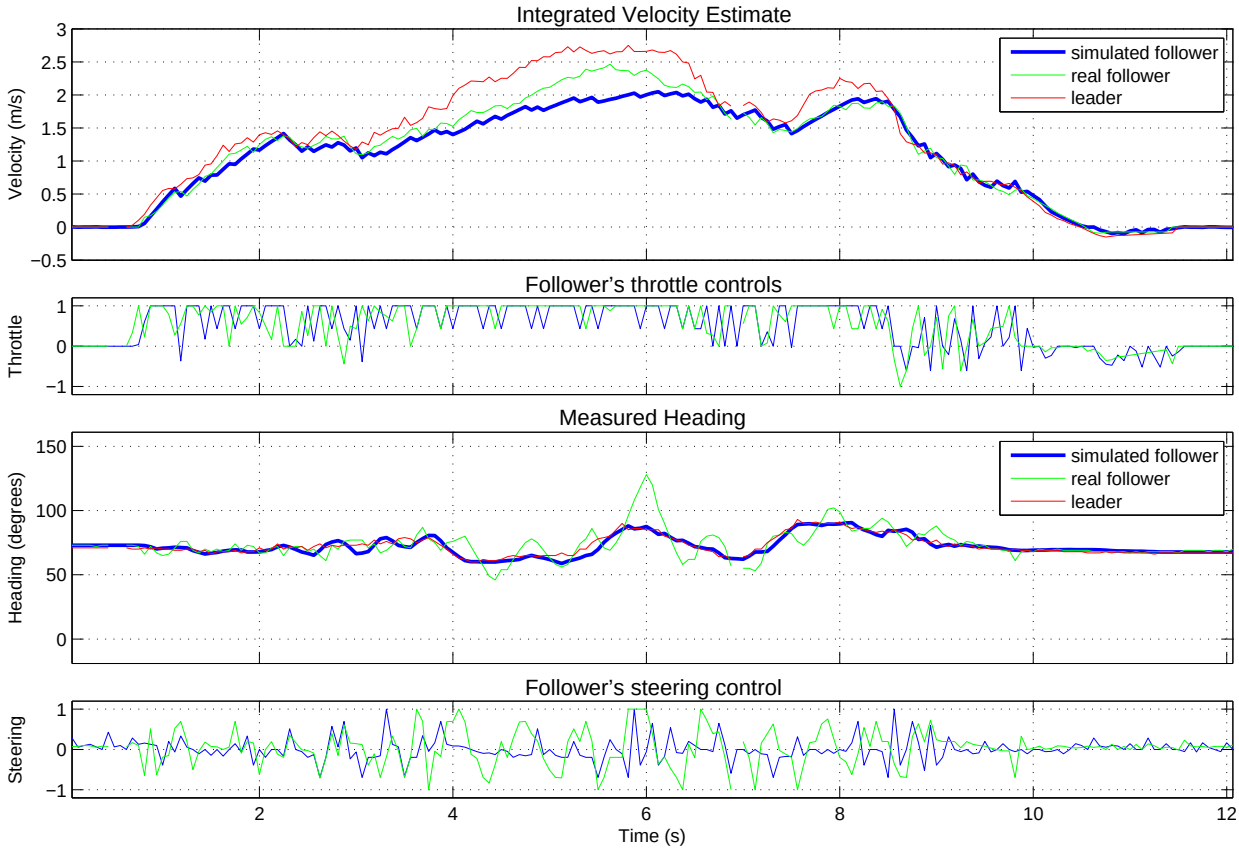


Figure 9.21: Velocity and heading simulation of the thirteenth linear test.

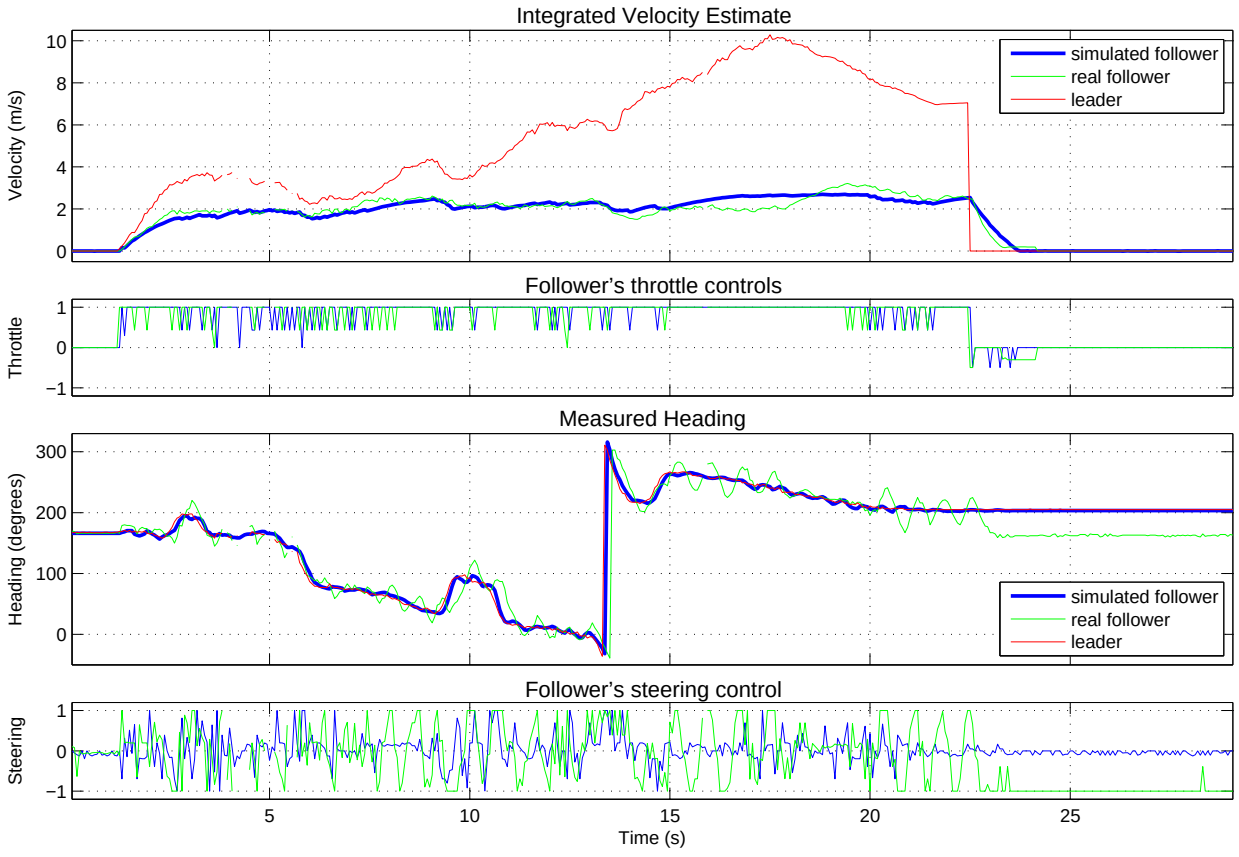


Figure 9.22: Velocity and heading simulation of the seventeenth square test.

emerge from a common cause.

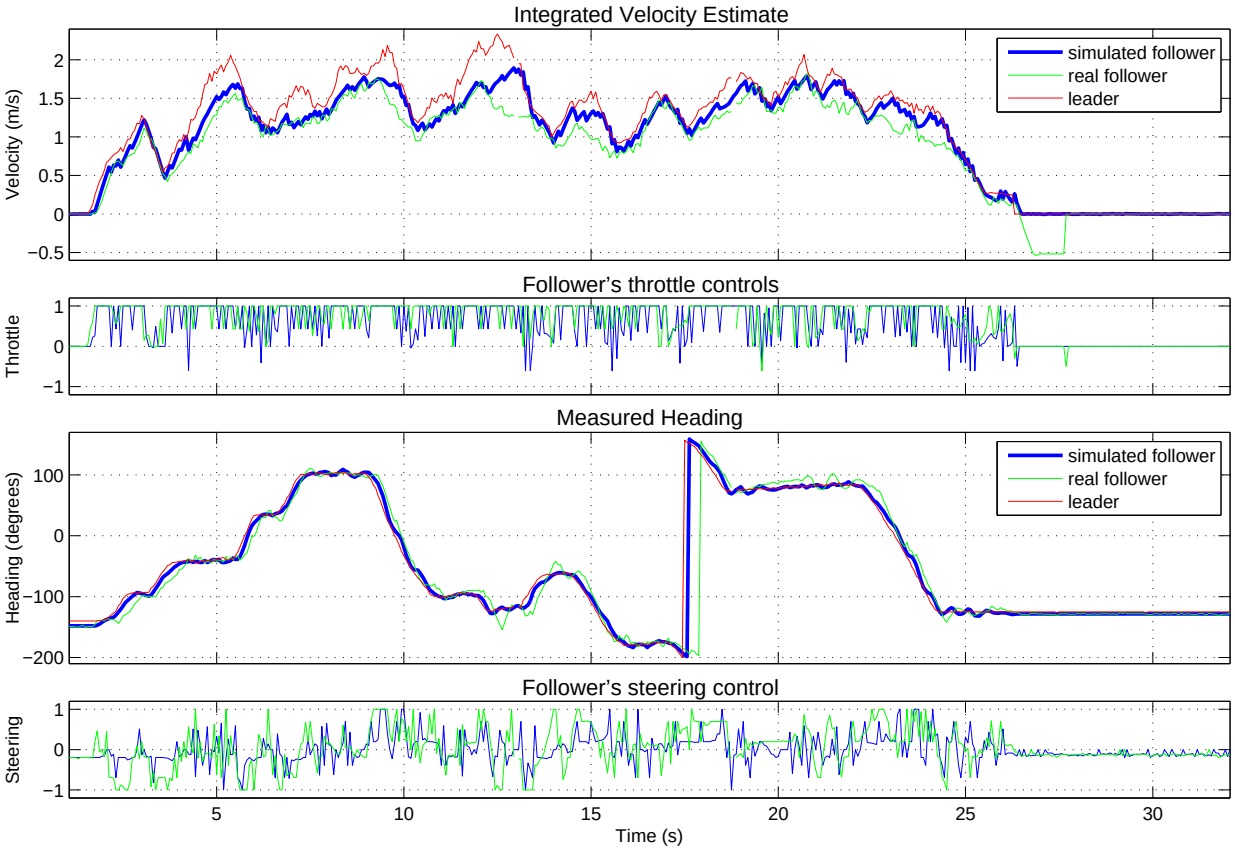


Figure 9.23: Velocity and heading simulation of the seventeenth random test.

Figure 9.23 shows the simulated results of one of the random tests we present in the previous section. The velocity simulation is very close to reality again, but what makes this figure particularly interesting is that the real follower steered significantly slower to follow the heading changes than the simulation. This happened even though the steering commands occur at the same time or earlier than in the simulation.

This indicates that the steering of the real car is slower than what we currently simulate. This does not pertain as much to the maximum steering angle the vehicle can achieve, but rather to the delay between a change in the steering signal from the controller and an actual change in the heading of the vehicle. This may very well be a cause of the fact that the follower shows steering instability in reality and much less significantly in the simulation. In Section 6.5.1 we explain that the controller needs to take this delay into account to provide stable control and that this delay partly determines how aggressive the controller can be without instability.

This is however not the only factor. When the simulation is modified to make the vehicle respond to the steering signal with a larger delay, i.e. by adjusting the steering filter of the simulation accordingly, the simulated follower indeed shows more significant instability. However, the sometimes very large heading deviations by the follower could not be reproduced this way. This means that other factors are at play as well.

Although we use simulations to tune our controllers for an optimum configuration, this optimum is not verified extensively with field tests yet. This requires testing the performance several times at various configurations, which is very laborious. That is why we omitted verifying the optimality of the configuration, but, considering the test results, e.g. the unstable steering, such tests are very valuable to further verify the validity of the simulation.

The results we present in this section show that our simulation can provide realistic results, particularly for the velocity response of the vehicle. The heading simulation in general produces a correct response

	Linear	Square	Random
Velocity (m/s):			
Mean Absolute Error	0.19	0.47	0.57
Standard Deviation	0.23	0.38	0.34
Correlation Coefficient	0.97	0.84	0.84
Heading (°):			
Mean Absolute Error	7.44	9.19	11.34
Standard Deviation	9.95	12.03	15.39
Correlation Coefficient	0.62	0.99	0.99

Table 9.6: Simulation versus real follower error and correlation statistics.

for the steering activities of the follower. However, it currently fails to provide a means to reproduce the full magnitude of the steering instability that the follower sometimes incurs in reality. Making the steering response of the simulated follower slower improves the simulation significantly in this respect, but the the largest deviations that the follower incurs in reality are never shown in that magnitude by a simulation.

Table 9.6 gives a quantitative view on how well the simulation matches the real behavior. Similar to Table 9.5 on page 123, we provide three statistical indicators for each type of test: the mean and standard deviation of the absolute error, and the correlation coefficient of the signals recorded in each test. In general, the real and simulated behaviors match well. The correlation coefficient in linear tests is relatively small for similar reasons as explained in Section 9.2.2. A detailed analysis of individual tests shows that a more accurate friction model is needed to further enhance our simulations. This is seen in the figures presented above where the simulated velocity approaches, but not quite matches the real velocity: at many of those occasions the simulation manages to achieve the leader's velocity better than the real follower. Also, in many cases, the relatively large differences observed between the real and simulated velocity are caused by the fact that the simulations omit integration errors.

In summary, the simulations match reality quite well, but we make the following additional observations:

- The swaying behavior of the follower is much less significant in the simulation: Although the simulation also shows sinusoidal deviations that oscillate around the desired heading, these deviations are smaller in amplitude and often higher in frequency. A cause of this may be the way we simulate steering, specifically with regard to the delay between a change in the steering control signal and a corresponding change in the vehicle's actual heading. However, simply increasing this delay does not fully achieve the magnitude of the real follower's swaying steering behavior.

- The swaying behavior presents itself often at the same time in reality and simulation: Visual inspection of the logs shows that the swaying behavior of the follower mostly occurs at the same time in reality and in simulation. This is an indication for a cause common to both reality and simulation. However, in the simulations the net effect is less significant.

- The velocity of the simulation is surprisingly close to reality: Both visually and by correlation, the velocity estimates of the real and simulated follower match each other very closely. The match is never perfect however: often the simulated follower performs better than the real thing. Probably our friction model needs more work to match reality more closely. Statistics extracted from the logs show a less appealing picture, but that is primarily caused by the fact that the simulation omits integration errors.

In general, we think that the simulation is predominantly a useful tool to verify whether the follower will perform adequately in reality. Large software problems can are easily be spotted in simulation. The more subtle problems like the steering instability require an improvement of the simulation. However, as the results in this section show, the current follower implementation also does not perform problem-free in the simulation. Meaning that, if these problems are solved in simulation, they could diminish in reality as well. And, coarse tuning of the controllers can be performed using simulation, yet, for performance fine-tuning the vehicles still need to be taken into the field.

9.4 Summary

To evaluate the performance of the implemented leader-follower system, we perform field tests on our university's running track and hockey field:

- Running track: The tests on the running track focus specifically on the performance of the heading controller described in Section 6.5.4. The surface of the running track is rough and the track is lined with metal fences and light masts. These tests omit the use of a leader: the desired heading is pre-programmed and changed 180° at regular intervals. Omitting the leader for the running track tests provides the means to test the follower's ability to follow an absolutely constant heading.
- Hockey field: The tests on the hockey field involve the full controller described in Chapter 6, with both leader and follower actively driving around. The hockey field has a smooth surface with no debris and metal objects are only found on the edge of the field. Three different drive patterns are tested on the hockey field:
  - *Linear tests*: The leader drives a 20 m straight line. This tests the ability of the follower to maintain a constant heading while following the velocity of a real leader.
  - *Square tests*: The leader drives an approximate square pattern with 15 m edges. The finish line is perpendicular to the start line. This tests the ability of the follower to track regular heading changes by the leader.
  - *Random tests*: The leader drives a random pattern for 30 s. This involves steering in random directions and large variations in velocity.

  Twenty individual tests are performed for each drive pattern. All tests are performed close the laptop to limit the amount of packet loss. At the end of each test the final heading of the cars and their relative distance is measured using a normal compass and a tape measure respectively. For the linear and square tests a finish line is defined and the distance to it is then measured as well.

The tests on the running track and the hockey field show the following results:

- The follower mimics the leader's heading and velocity: On the running track we show that the heading controller adequately achieves the desired heading when the compass is calibrated and mounted correctly. During the hockey field tests, the follower manages to stay close to the leader within a few meters most of the time. The follower clearly tries to mimic the leader's heading and velocity, albeit with some delay and control stability problems.
- The communication protocol works adequately: Statistics of the packet loss observed at the gateway node shows that almost 99% of the messages is received from both nodes.
- On the hockey field the follower sways heavily on straight sections: Especially when the leader is driving at high velocity, the tests on the hockey field that include driving straight lines show a follower that sways in a sinusoidal motion around the desired heading.
- The compass is very sensitive to external influence and inclination changes: If the compass' position, orientation or inclination relative to the vehicle changes, the sensor needs to be re-calibrated. Inclination changes of the whole vehicle cause noise on the compass sensor readings and large nearby metal objects cause significant measurement errors.
- Errors can cancel each other out: Despite the large integration errors at the leader and the swaying behavior of the follower, the latter still manages to keep on the right trajectory, especially when the leader is driven with frequent changes in heading. This is partly explained by the fact that errors tend to compensate for one another. In addition, the limited velocity of the vehicles limits the impact of the integration error.
- The velocity integration is very sensitive to inclination changes: When the vehicles stop, the end velocity is often not zero, showing the presence of integration errors. The leader performs worse in this respect during most tests, especially when it is steering frequently.
- The follower often stops seconds later: We notice that the follower generally ends up further relatively to the finish line than the leader. This is caused by the velocity integration errors and the inherent delay of the motion detection technique.

- The follower's brake works reliably: When the follower needs to stop, it can do so within one second from high velocities.
- The leader's relatively high agility influences the tests: First, it is hard to drive our very agile leader vehicle at a velocity and rate of steering that the follower can match. Second, it is hard to manually let the leader drive a designated pattern due to its agility.

Our measurement method provides the means to assess the performance of the system after a period of driving, i.e. to assess to what extent the controller manages to achieve its goal of staying close to leader by mimicking its movements. It does however not provide an absolute reference on how the cars performed during a test, giving error cancellation a chance to distort the results. By including the measurements logged from the nodes themselves into our assessment, we feel confident however that we can conclude that our system presents the desired behavior, albeit with some significant problems at leader and follower pertaining to data collection and (manual) control.

The simulations presented in Chapter 7 are verified in this chapter by comparison of simulation results to the results collected from actual tests. To make proper comparison possible, the simulation is performed using the data communicated from the leader during the actual test. This means that if the simulation and the real test match closely, the resulting heading and velocity progression should be close to identical for reality and simulation.

The performed simulations match reality quite well, but we make the following additional observations:

- The swaying behavior of the follower much less significant in the simulation: Although the simulation also shows sinusoidal deviations that oscillate around the desired heading, these deviations are smaller in amplitude and often higher in frequency.
- The swaying behavior presents itself often at the same time in reality and simulation: Visual inspection of the logs shows that the swaying behavior of the follower mostly occurs at the same time in reality and in simulation.
- The velocity of the simulation is surprisingly close to reality: Both visually and by correlation, the velocity estimates of the real and simulated follower match each other very closely. The match is never perfect however: often the simulated follower performs better than the real thing.

The more significant swaying of the follower in reality may be caused by the way we simulate steering: results indicate that the simulated vehicle responds too quickly to steering commands. However, making this response slower never achieves the full magnitude of the deviations observed for the real follower. The fact that the swaying motion occurs at the same instances in reality and simulation points to a common cause. The differences observed between the real and simulated velocity are partly caused by the fact that the simulations omit integration errors, but probably our friction model needs more work as well to match reality more closely.

In general, we think that the simulation is predominantly a useful tool to verify whether the follower will perform adequately in reality. Large software problems can are easily be spotted in simulation. The more subtle problems like the steering instability require an improvement of the simulation. Tuning of the controllers can be performed using simulation, albeit with low accuracy: fine-tuning needs still needs to be done using field tests.

Chapter 10

Conclusion

In this work, we describe FollowMe, a distributed method for team trajectory coordination using regular toy cars augmented with wireless sensor nodes and inertial sensors. We implement this system to explore the potential of Wireless Sensor Network (WSN) technology in dynamic applications requiring a *sense-and-control* loop. The final goal is to have a self-organizing group or swarm of mobile nodes that maintain a formation by periodically exchanging their sensed movement information. We specifically investigate a formation that contains a *leader* whose trajectory needs to be copied by a set of *followers* in a convoy formation. The leader is controlled manually. Our research is currently limited to only one follower. We further simplify our design problem by not considering the presence of obstacles and assuming that the surface that the vehicles are driving on is smooth and level. Also, for this initial works, we do not aim to actually let the follower track the leader's path in a convoy-like manner, but we rather aim to copy its movements to attain movement synchronization. A solution that achieves the true convoy formation requires some means to determine the distance between the nodes, which is currently unavailable.

Concretely, our design problem involves obtaining sufficiently accurate movement measurements at both nodes, communicating these measurements wirelessly from leader to follower node and controlling the follower such that it successfully mimics the leader's movements. This control loop is to be designed, simulated, implemented and tested on real vehicles. Challenges include the resource limitations of the sensor nodes, attaining an appropriate accuracy and frequency for the movement measurements, limiting the required bandwidth for the communication between leader and follower and designing a controller with sufficiently rapid response time.

In our solution, both leader and follower sense their movements only using inertial sensors, meaning that the use of GPS is omitted. Because we focus our research on the use of WSN technology, we use only small, low-cost, low-power and low-resolution inertial sensors. We use an electronic compass sensor to measure the compass heading of the vehicle and we use an accelerometer to estimate the current velocity of the vehicle by integration. Using WSN technology, movement information is communicated from leader to follower. The leader node only has sensors attached, whereas the follower also has the means to control its vehicle. The follower includes a controller that aims to match the heading and velocity communicated from the leader by controlling the vehicle's actuators accordingly.

Both compass sensor and accelerometer are sensitive to inclination changes, meaning that if the vehicles drive over uneven terrain, the sensor measurements will be inaccurate. This effect can be compensated when the current inclination of the vehicles is known. Obtaining this information during driving is tricky and a common solution is to use a gyroscopic sensor. Due to its poor availability and the complex calculations involved to perform the compensations, we currently avoid using such sensor or any other kind of tilt compensation. Considering the assumed level driving terrain, this should not be problematic.

In light of the resource limitations of the sensor nodes and the relatively unreliable sensors, we choose to use fuzzy logic technology for the follower's controller due its appealing properties in this regard: fuzzy logic control can be employed on limited hardware and it is robust to imprecise and unreliable data. Additionally, using fuzzy logic control can significantly reduce the design and development time and it handles non-linear systems better when compared to conventional approaches.

In this final chapter we conclude our work. First, in Section 10.1, we summarize the most important design choices we made throughout this work and we outline the most important system properties. Then, in Section 10.2 we summarize the performance of the system and we reflect upon the design problem and design choices. That is also where we draw a definitive conclusion about the result of this work. Finally we provide a survey of recommendations and open research issues in sections 10.3 and 10.4 respectively.

10.1 Design Choices and System Properties

As we describe in Chapter 2, the hardware of our FollowMe system is composed of the sensor nodes, the inertial sensors and the toy car vehicles. The software running on the nodes performs data acquisition, data processing (navigation), communication and vehicle control. Apart from the communication and vehicle control tasks, the software running on leader and follower is mostly identical. The follower controller is first verified, tuned and evaluated in a simulation before the actual implementation is tested.

The vehicles are two different toy cars of which the leader is faster and more agile than the follower. The follower vehicle is modified with a custom motor controller to provide the means for the attached node to control its motors. The sensor nodes run a real-time multitasking operating system called AmbientRT. Other than leader and follower, a third gateway node is present to receive messages from both mobile nodes. These messages are recorded for later evaluation.

The mobile nodes are equipped with compass and acceleration sensors, connected to the node though a shared SPI bus. These are used to keep track of the node's heading and velocity respectively. Both these sensors are shown to be influenced by inclination changes. We currently do not compensate for inclination, because the means to continuously measure inclination are not available and we assume the driving range to be approximately level. The compass sensor needs to be calibrated to account for metal and magnetic influences from the vehicle itself. This also compensates for the static inclination. Calibration cannot compensate for external and dynamic magnetic and metal influences, meaning that those need to be avoided. We assume our driving range to be void of such objects. The inherent noise observed on the outputs of both sensors is insignificant when there is no movement. The compass additionally shows low hysteresis, meaning that measurements are very deterministic. The accelerometer measures much harmonic noise when the vehicle is driving.

The heading is inferred from the compass sensor measurements using a simple arctangent function. A node's current velocity is estimated by integrating the acceleration measurements. Practical tests show that a sample frequency of 16 Hz is adequate for the compass sensor and that a sample frequency of 160 Hz is adequate for the accelerometer. The compass frequency is, along with the follower control frequency, chosen to be sufficient to account for fast heading changes by the leader while still being feasible for the limited node. The accelerometer frequency is one of the frequencies supported by the sensor itself and experiments show that the resulting velocity integration is very smooth at that frequency. The accelerometer frequency is very high, but the velocity integration algorithm is performed at a ten times lower rate to reduce the CPU usage on the node. The sample frequencies of the two sensors differ by a ratio of ten, which is a potential problem if both sensors need to be sampled simultaneously over the shared SPI bus. Luckily, as we explain in Chapter 3, the slower compass sensor does not need to occupy the bus when measuring, meaning that sampling the two sensors can be interleaved.

In Chapter 4, we describe how raw sensor measurements are processed into velocity and heading measurements. Unlike the common inertial navigation techniques, our velocity integration technique only uses the accelerometer to obtain a velocity estimate. This is based on the assumption that our cars can only move forwards or backwards and need to be driving to change their heading. Tests with linear and circular motion show that our velocity integration strategy works as intended. Unfortunately, integration accumulates errors in sensor measurements until eventually the velocity estimate does not reflect reality anymore. This is solved by assuming that the vehicle stops every once in a while, which provides an opportunity to reset the velocity to zero. Such a standstill is detected by observing the amount of (harmonic) noise on the accelerometer signal as inferred from calculating the standard deviation on a history of samples. At each standstill, the accelerometer is re-calibrated for the current inclination.

Our heading calculation aims to achieve a resolution of 1°. The compass sensor can be configured at various angular sensitivity levels, but this configuration also negatively influences the attainable sample frequency.

Ideally, this would require an angular sensitivity of the sensor of 0.1° , but to achieve the 16 Hz sample frequency, we use a sensitivity of 0.5° . This means that the resolution of 1° is achieved with less reliability when noise is at play. The heading results from different, calibrated compasses correlate very well with deviations of little more than a degree on average, provided that the inclination remains constant.

We use a simple communication protocol to propagate radio messages from leader to follower and from both nodes to the gateway. As we describe in Chapter 5, the protocol is strictly one-way and reactive to avoid the complexity of medium access control. The gateway is connected to a PC through a serial connection. The leader controls the protocol and the follower synchronizes its transmissions to incoming messages from the leader. The follower internally does not synchronize to the incoming messages, meaning that these are asynchronous. This has the benefit that the follower's controller is executed even though no messages are received, making it follow the last known leader state. The exchanged messages not only contain the movement data of the transmitting vehicle, but also raw sensor readings and other values useful for debugging purposes. To detect packet loss at the gateway during evaluation, the exchanged messages contain a sequence number, which also serves to distinguish between messages from leader and follower. When logging data from all nodes to the gateway is not necessary, the protocol is inherently feasible for large numbers of followers, since the leader is the only node transmitting.

We use fuzzy logic to implement our follower controller. Because heading and velocity of the follower are influenced by two different motors, heading and velocity control are assumed to be largely uncoupled. Therefore, in Chapter 6, the controller is decomposed into a fuzzy heading controller and a fuzzy velocity controller. Control signals from both sub-controllers are compensated for mutual influence in the post-processing stage. This is also where the inductive brake of the vehicle is applied when the leader has stopped and the follower approaches a sufficiently low velocity. Both fuzzy controllers are structurally identical, with only differences in the range of the input values, the definition of the membership functions and post-processing operations.

In Chapter 8 we describe a series of practical issues for both hardware and software. The main hardware problem is the placement of the sensors. Theoretically, as long as the relative orientation is identical among the two vehicles, the actual position of the sensors does not matter. However, the compass sensor is easily influenced by metal and magnetic influences of the vehicle itself. On the leader this is not a very profound issue, as the sensors are located on top of the vehicle, far away from any actuators. To provide a more rigid mounting of the sensors to the leader, they could be mounted directly on the chassis, but this relatively small change in position is enough for the compass sensor to incur significant noise from the drive motor. The follower has the compass sensor directly above the steering actuator, causing a very constant influence if the actuation intensity remains constant. To remove this influence, we mount a 5 mm thick plate of metal between sensor and actuator.

The main software problem is scheduling sensor sampling, data processing, communication and control simultaneously on the nodes. Both nodes use the same sampling strategy and the associated scheduling is identical with the sampling tasks having the shortest deadlines to give them (usually) the highest priority. Overall, scheduling is much simpler on the leader, since communication only involves transmission and control is omitted entirely. The follower needs to receive messages asynchronously and needs to transmit its own message within a short period of time to uphold the communication protocol. Additionally, the radio transmission must not be performed when the compass is measuring to prevent interference. For the leader, this is trivial since the transmission is triggered at a deterministic instant in the scheduling. The follower potentially receives messages asynchronously in the middle of a compass measurement, meaning that the subsequent outgoing transmission has to wait until the transmission completes.

The data processing calculations require the implementation of an arctangent and square root function on the nodes. We choose to use a simple piecewise-linear representation of these functions in a look-up table. Intermittent values are linearly interpolated between table values. This requires fractional number calculations. Because the node processor has no hardware support for floating point calculations and software emulation is bound to be slow, we use fixed point fractional calculations exclusively. This means that the fractional operations are in fact normal integer calculations with the radix point defined at a fixed position.

We use simulations to test, evaluate, tune and debug the follower controller: simulations are used to generally test the validity of the implemented controller, to evaluate its (simulated) performance, to tune it for optimum (simulated) performance and to find software issues that surfaced during field tests. The simulations we describe in Chapter 7 use a simplistic vehicle model that includes a model for the friction that the vehicles incur during driving. The controller outputs are fed to the vehicle model and the model yields a new vehicle

state that is fed back to the controller as simulated sensor measurements. To make the simulation more realistic, uniformly distributed random noise is added to both control and sensor signals to emulate inaccurate control and sensor noise respectively. Debugging of the controller and validation of the simulation itself is performed by feeding movement data from the real leader as input to the controller inside the simulation. This provides a means to reproduce problems in the lab and to compare the simulated and real response of the follower. To isolate problems that only occur on the sensor nodes themselves, the controller is run on the actual node and connected to the simulation on the PC through a serial connection.

10.2 Discussion

In Chapter 9, we evaluate the performance of the system using tests on the university's running track and hockey field. The tests on the running track only involve the heading controller to verify its performance without the influence of the leader vehicle and the person operating it. The running track tests involve the follower driving one side of the track back and forth on a pre-programmed heading. For the hockey field tests, the follower is fully implemented and the leader is active as well. We perform linear tests in which the leader drives a 20 m straight line, we perform square tests in which the leader drives an approximate square with 15 m edges an finally we perform random tests in which the leader drives a random pattern with varying velocity during 30 s. Twenty individual tests are performed on the hockey field for each of these drive patterns. In Section 9.3, we compare the hockey field test results with simulated results from identical leader movement in order to validate the simulations.

In this section we use the test results and notions from Chapter 9 to reflect upon our implementation, our test methods and our simulations. Finally, in Section 10.2.5 we provide a summary of our conclusions.

10.2.1 Test Results

Tests on the running track show that the heading controller adequately achieves the desired heading with a standard deviation of only a few degrees, provided that the compass is mounted and calibrated correctly. The hockey field tests show that the follower presents the desired behavior: it visually manages to stay close to the leader within less than 5 m most of the time. It mimics the leader's movements, albeit with a delay and control stability problems on straight sections.

The follower is seen to sway significantly on the straight sections of the linear and square tests. During the random driving patterns, the swaying behavior subsides for the most part. Additionally, the follower's measured velocity is never seen to fully match the leader's velocity. This is caused by the asymmetric nature of the velocity control problem: acceleration is inhibited by friction, while deceleration is aided. This is also observed in simulation. Both these control issues indicate that the fuzzy controller needs more work to achieve better stability and better performance. This can be achieved by using new or different inputs for the two fuzzy inference systems and amending the fuzzy rule set accordingly. For the velocity controller, a possibility is to add an additional input that indicates whether the vehicle needs to speed up or slow down to reduce the velocity error. This gives the ability to actually compensate for the asymmetry in velocity control. To improve the steering stability, the exact cause of the swaying behavior needs to be found first.

The velocity integration technique we use works surprisingly well, considering that we make bold assumptions on the vehicle's movement capabilities and only use the accelerometer. However, it is sensitive to inclination changes, causing the velocity estimate to rise beyond what is possible for the cars. The leader vehicle is more affected, probably due to the weaker mounting of the sensor and the weak suspension of the vehicle. This problem also causes the follower to stop seconds later due to the fact that our motion detection technique needs time until it resets the erroneous final leader velocity to zero. Actually stopping the follower takes additional time. However, when the follower knows it needs to stop, it can do so reliably within a few seconds using negative throttle followed by applying its inductive brake. Solving the velocity integration issues requires some means to measure and compensate for a dynamically changing inclination, possibly using a gyroscope or heuristically using the accelerometer itself (refer to Section 9.2.2).

The heading calculated from the compass sensor readings is also sensitive to inclination changes, although less continuously: where the velocity integration retains the inclination errors until the velocity is reset

to zero by the motion detector, the compass provides a valid heading again when the inclination returns to the level the compass was calibrated for. Proper calibration is a very important prerequisite for the follower achieve any useful performance at all. Making even minor changes to the position and orientation of the sensor or the composition of the vehicle makes re-calibration a necessity. When calibrated, compass is still sensitive to metal and magnetic distortions. Even though interferences from the car itself can be compensated by calibration and shielding, external influences like driving close to large metal objects has a significant influence on the performance. There is no real solution for these interferences, other than avoiding their encounter. Combining the compass sensor with other methods to keep track of the vehicle's heading could provide the means detect and compensate for such interference. Methods again include the use of a gyroscopic sensor.

10.2.2 Test Method

To obtain an objective view on the performance of the system, we take manual heading and distance measurements after each test on the hockey field as to record the final state. The collected data serves as the means to assess the performance of the system after a period of time, i.e. whether the follower manages to stay close to the leader. This data only provides a snapshot of the test right after the cars finished driving and it does not provide information on how well the follower fared while the cars were still driving. Errors that emerge during driving can cancel each other out or deteriorate the results of an otherwise very good test just before it finishes. The results show acceptable ending distances and heading differences for most tests. However, the follower often drives further for several seconds due to velocity integration errors, distorting the results.

Continuously measuring the absolute distance and heading of the nodes during a test is difficult and requires more elaborate measurement methods, for example using a camera. We obtain an indication how the nodes performed during driving by using the heading and velocity measurements logged from the nodes themselves as a reference. This data is not very reliable and cannot be used as an absolute reference, because these measurements include the inherent sensor and integration errors. It does, however, show how well the follower controller manages to match its own measured heading and velocity to the heading and velocity reported by the leader. Statistics extracted from these logged measurements show that the measured heading and velocity correlate well. However, the measured velocities can differ significantly in their absolute values due to integration errors that have a different impact on the individual cars. Also, the significantly more agile leader easily exceeds the follower's capability, yielding suboptimal results in tests where the follower could never have achieved the desired velocity or rate of steering. Considering the effect of the agility difference on the results, it is important to use leader and follower vehicles that have matching velocity and steering capabilities for future testing in order to obtain more reliable test results.

10.2.3 Test Evaluation

The results discussed above show that the primary goals of this work are achieved: both mobile nodes obtain sufficiently accurate movement measurements and these measurements are correctly communicated from leader to follower in real-time such that the follower can mimic the leader's movements. During many of the (random) tests the follower manages to stay close to the leader for 30 seconds and beyond. However, major problems still exist: the influence of the vehicle's inclination on the sensor measurements, the sensitivity of the compass sensor to distortions and the instability of the follower controller. The designed and implemented fuzzy logic controller can deal with the noisy sensor measurements, but the accumulation of error in the inertial movement measurements is only dealt with properly when the vehicles stop on a regular basis, providing a reset opportunity. Our initial assumptions that the system is not subjected to external influences and that the mobile nodes are physically identical are not entirely justified for the performed tests: the compass is easily influenced by metal objects close to our testing locations and the toy cars we used are very different. Despite these issues, the system still manages to achieve the primary goals during most tests.

Our tests do not show what the effect is of using alternate design choices, regarding the control, sample and communication frequencies for instance. Choosing a higher control frequency would for example improve the response time of the system, but it also increases the CPU occupation, making scheduling even more difficult. Tests at a series of configurations could point us to the most optimal alternative. Also, our current protocol is very simple and the control frequency is equal to the communication frequency, meaning that a message

is sent from leader to follower irrespective of whether anything has changed in the current movement of the leader. For efficiency, we must investigate what the net effect is of reducing the communication frequency and how to alter the protocol and control strategy to reduce the message exchange between the nodes without reducing performance significantly.

10.2.4 Simulation Evaluation

Such alternate configurations can also be evaluated using simulation, which is an additional goal of this work. In such a simulation the control loop of the follower acts upon a simulated vehicle with a realistic response to the controller's actions. In Chapter 7, we define this simulation. Its validity, i.e. whether it is realistic enough to mimic reality, is assessed in Chapter 9. This validation is performed using movement measurements logged from the real leader as input to the simulation. The resulting simulated follower response is compared to what the follower did and measured in reality.

The results indicate that the simulated follower response matches the real follower quite well considering the correlation of the resulting heading and velocity signals. The absolute difference between the simulated velocity and the velocity as measured by the real follower is sometimes very significant however. This is due to the fact that the simulation does not consider the velocity integration problems like the effect of inclination changes. Also, the simulation shows a lower response time to heading changes. The most significant difference between reality and the simulation is that the follower sways much less significantly on the straight sections for the linear and square tests recorded at the hockey field. This indicates that the simulation improperly emulates the factor that causes this controller instability. Note however that, although the magnitude is smaller, the heading instability of the follower is often observed in simulation at the same time as in reality, indicating that this factor is not neglected entirely.

We think that the simulation is particularly useful to determine whether the follower will perform adequately in reality. Obvious software problems are easily spotted in simulation, but the more subtle problems like the steering instability require an improvement of the simulation. Tuning the controller using simulation is mostly a matter of coarsely finding a configuration that will be close to optimal in reality. Real tests are needed to perform fine-tuning, because the simulation is simply not accurate enough yet. In this work we only coarsely tuned the width of the fuzzy input membership functions of the heading controller using simulation, but as we describe above, this can be extended to other configuration alternatives as well. In general, a better simulation model, e.g. for the driving friction, would appreciably improve the simulation towards more realistic results.

Using the simulation for benchmarking as explained in Section 8.2.4 shows that the bare fuzzy controller has a relatively low impact on the CPU usage of the node. Equally significant are the time spent sampling the sensors and sending commands to the motor controller. This shows that choosing fuzzy control for its low computational complexity is a valid rationale. With efficiency improvements on the execution of the various tasks in the system, ample room remains for more elaborate navigation and control operations.

10.2.5 Summary

Summarizing, this initial work shows that it is feasible to implement a sense-and-react control loop for movement coordination using WSAN technology. Our implementation has the following benefits:

- The main benefit of the method we describe in this thesis is that it enables autonomous mobile team coordination based solely on compact, low-cost wireless sensors and actuators.
- Our solution is theoretically not restricted to vehicles on wheels, but supports *any moving entities capable of determining their velocity and heading.*
- Another advantage is the constructive robustness to errors: from any initial or intermediate faulty orientation, the follower is able to return to the correct heading, due to the usage of a magnetic compass as a reference.
- Furthermore, making use of a fuzzy controller facilitates the implementation on resource constrained sensor nodes and handles robustly the noisy sensor data as well as the rough mechanical capabilities

of the vehicles.

- The vehicle simulations we devised show realistic results and provide a means to test, evaluate, (coarsely) tune and debug the controller.
- With respect to wireless communication, the leader-follower model is inherently scalable and tolerates transient packet losses.

However, our implementation still faces numerous problems that eventually need to be solved:

- As any inertial navigation solution, FollowMe accumulates errors through the integration of acceleration measurements into a velocity estimate. Without stopping periods or external reference sources like GPS, the error can potentially grow to infinity.
- Additionally, our current implementation does not support dynamic tilt compensation, causing problems on inclined terrain.
- The compass utilization is restricted to environments without strong magnetic influences, typically outdoors.
- The follower controller does not always show stable behavior with respect to steering and it handles the asymmetric nature of the velocity control problem not entirely adequate.
- The simulations need to be improved for more realistic results. Although the simulations show results that are very similar to the actual tests, the follower consistently performs better in simulation than in reality: for example, the simulations fail to fully reproduce the steering instability of the follower.
- With respect to convoy-like applications, the main limitation is the lack of any information and subsequently control of the relative distance between the vehicles (see also future work in Section 10.4).

10.3 Recommendations

The compass sensor used for this work is a prominent reason for various sometimes very surprising and often very frustrating problems that surface during testing. Many of the compass-related problems are well-documented in the - not so very abundant - documentation, but others, like the strange radio and battery interferences, were encountered without prior notice. We learned that it is often best to focus on isolating and fixing these problems first, rather than continuing the generic tests on the outside testing range, as these otherwise mostly turn out to be a frustrating waste of time. However, especially the transient nature of these issues makes it hard to isolate them. For example, the radio interference on the compass was in some implementation versions solved by coincidence, depending on how the scheduling turned out for the task set of that particular version. Small changes to the software can then suddenly lead to the compass problems surfacing with no apparent reason.

Also regarding its placement on the vehicles, the compass sensor is reason for concern. The oddest influences were observed emerging from the vehicle's actuators and other components. A prominent example is the magnetic odometer we discovered on the follower car. Its revolving wheel-mounted magnets would cause a peculiar sinusoidal signal on the compass sensor output, especially on the axis pointing in the general direction of the wheel. And, even though actuators are located at considerable distance, they can still cause relatively large transient spikes on the compass output signal. We learned that changing the location or orientation of the compass sensor does not only influence its calibration, but also how much noise it picks up from the vehicle itself.

The main lesson learned about software implementation on sensor nodes is that sampling inertial sensors requires a major share of the available CPU power and that a multitasking operating system is preferred. Scheduling all the tasks is difficult, especially due to the compass sensitivity to various interferences. We use a logic analyzer connected to the unused generic I/O ports of a node to visualize the scheduling of the tasks. We describe this procedure in Section 8.2.2. This visualization method is recommendable, since it works in real-time and produces an intuitive representation of what is going on inside the node. We particularly use this to check whether the compass and radio are activated simultaneously and to asses the relative CPU

utilization of each task in the system. The scheduling on the node can further be improved by reducing the time spent sampling the sensors and sending the control commands to the motor controller through the SPI and I^2C buses respectively. This time can be reduced by using a hardware implementation of these bus protocols.

The large amount of computation involved in inertial navigation and control algorithms is another potential source of hard-to-identify bugs. Introducing a feedback loop from the sensor node to the simulator proved to be an efficient debugging method for this problem. We learned that, when (simulated) evaluation on the PC shows no problems while strange things happen during field tests, the problems are usually caused by platform dependencies. By running the software on the node in the lab also, problems prove to be easily reproducible using the data collected from the failed field tests. This is used for debugging the controller, the mathematical function implementations and the data processing algorithms.

From field experiments, the most important lesson learned is that the constructive limitations of the vehicles can have both a negative impact (such as the lack of a mechanical braking system) and a positive influence (the limited engine power of the follower bounds the effect of error integration in velocity). Compass calibration remains a necessary step for producing accurate experimental results. And, very importantly, driving the toy cars proved to be challenging, as driving the designated pattern is difficult with the agile leader and collisions at 30 km/h could destroy the sensors mounted on top.

Another cause for hardware destruction proved to be the fact that one sometimes forgets that the follower responds autonomously to anything one does with the leader car while both are still active. Once, the follower was placed active on a table while the leader was reprogrammed, causing it to make a nose dive towards the floor. On the field, the follower went crazy on various occasions when we forgot to turn it off before picking up the leader, for example making it drive into the side of the field at full throttle. It is therefore important to keep the autonomy of the follower in mind at all times.

In general, experiments were more time-consuming than expected, required a dedicated, large open field and, last but not least, were dependent on the occasionally good Dutch weather. We strongly recommend reserving ample time for testing and performing such research during the summer months. Alternatively, a very spacious indoor location like a sports complex can be used to avoid the weather dependencies. The effects of using the compass sensor in such a building are unknown, but the compass' performance could prove to be satisfactory as long as the walls are kept at a distance.

10.4 Future Work

The primary open research issue for future work is to incorporate relative distance estimation into the fuzzy controller, in order to apply our solution to convoy-like applications. This can for instance be achieved using radio signal strength (RSSI) information, but this is not a very robust or reliable implementation.

Also, the research described in this thesis only considers a single follower. Because the used communication protocol is based on a leader broadcast, it should be extensible to multiple followers without much effort. Tests with multiple followers should show how well our implementation achieves the desired swarm behavior with the minimal means we currently use.

An important advantage of our approach is that it could be applicable to other types of mobile nodes as well. Although we assume our vehicles to be identical in this work, we chose very different vehicles and test results indicate how this influences the system's performance. To extend our research to a more generic FollowMe idea, group interactions among heterogeneous mobile entities should be explored, not necessarily being vehicles on wheels.

Finally, throughout this thesis the use of a gyroscopic sensor is mentioned as a means to solve various problems that our current design faces: it can keep track of inclination changes to correct compass and accelerometer measurements, it can be combined with the compass to make it more reliable when faced with dynamic metal distortions and it can be used to implement a true inertial navigation system. However, gyroscopic sensors also need integration to obtain an angular displacement value, making it prone to accumulating angle errors. Also, the calculations needed for true inertial navigation or to correct the accelerometer and compass measurements for inclination influence are quite involved and might prove hard to implement on the nodes.

Further investigation is necessary to assess the merit of gyroscopic sensors for our application.

Appendix A

Platform Details

In this appendix we describe the details of the hardware involved on the leader, the follower and the gateway. The leader vehicle is left largely unmodified, with only a sensor node and sensors attached. The follower vehicle has extra hardware to give the follower node the ability to control it. In Section A.1 we describe the details of how the sensors are connected to the nodes. We describe the motor controller, which is specific to the follower, in Section A.2. Finally, in Section A.3 we describe the software running on the gateway node and the serial protocol used for communication with the PC.

A.1 Sensor Interface

In Section 2.3.1, we provide a schematic overview of the connection of sensors to the sensor nodes and we describe the SPI bus used for this connection. In this appendix section, we provide a more detailed description of how the sensors are physically connected to the node.

The connections of the LIS3LV02DQ accelerometer and the Micromag3 compass sensor are identical for leader and follower. Figure A.1 shows the electric circuit diagram involving the μNode and the compass sensor and accelerometer. The node is connected to the sensors using a software SPI bus on CPU ports *6.2*, *6.3* and *6.4*. The node is the SPI master and the sensors are two SPI slaves. The master's SPI serial in and out lines (*MISO* and *MOSI* respectively) and the SPI clock signal (*SCLK*) are shared between the sensors. However, only one sensor can be active on the bus at a time. This is regulated by the slave select lines called *ACC_SEL* and *COMP_SEL* which select the accelerometer and the compass respectively.

Both sensor devices have a *DRDY* or *INT* output which can be used to notify the node's processor of changes, like completed measurements. For the compass sensor the signal line is called *COMP_RDY* and for the accelerometer it is called *ACC_RDY*. It is most efficient to connect these to an interrupt input of the node and configure these sensor outputs accordingly, but as we describe in Section 3.1.2 this is currently avoided. This means that the node needs to poll these ready signals. Currently, this is only done for the compass sensor. The accelerometer is polled directly through the SPI bus for completed measurements.

The compass has an additional *RESET* input, which is required to reset the sensor for new measurements. A separate node output is used to generate this signal. The accelerometer also has an extra input called *CK* for an external clock. This could be used to use the sensor at sample frequencies other than the standard ones available. Currently, we do not use this feature and therefore this input is connected to ground.

A.2 Motor Controller

When first bought, toy cars like the ones we use contain a controller of their own with a remote control. Such a remote control typically uses a frequency somewhere in the 27 or 40 MHz bands. For our research a μNode needs to control the follower car. However, as we show in this section, this is not trivial.

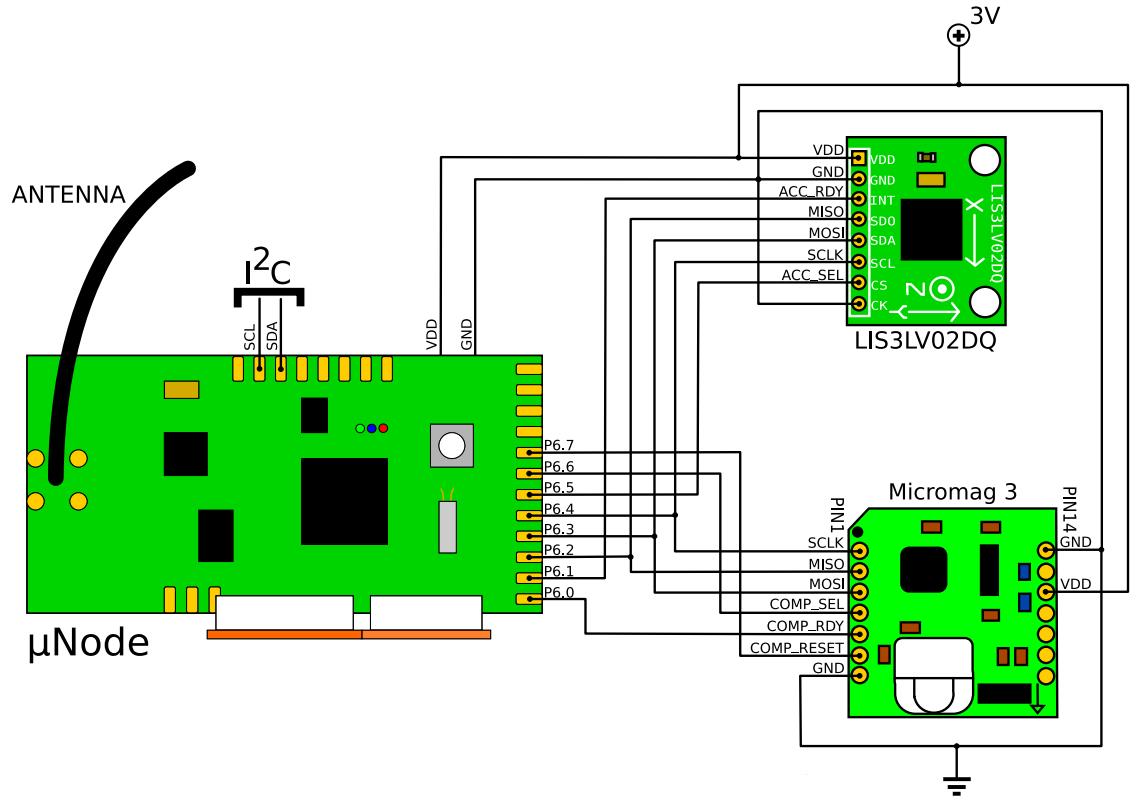


Figure A.1: Sensor node connections with external sensors.

First, there are alternative solutions to this problem one may consider. The intuitively easiest solution is to interface the sensor node to the remote control of the car. This means that the original controller on the can be retained. However, this solution is not particularly clean, fast or reliable: the remote needs to be mounted on the car itself and an extra transmission path is created between the node controller software and the car's actuators, creating extra delay and potential for failure.

Therefore, we replace the original motor controller of the toy car with a custom controller that directly communicates with the sensor node over an electrical bus. The I^2C bus specification was chosen as bus protocol for its simplicity and flexibility.

A.2.1 Hardware

Our custom motor controller needs to control two separate DC motors on our follower car: the drive motor for the car's acceleration and the steering motor for changing the angle of the front wheels. Unlike many other (more expensive) toy car designs, the steering motor of our follower is not a servo, but rather a normal motor that deflects the steering from its center angle. If the steering motor is turned off, the wheels return to their center angle by means of a weak spring. Therefore, the intensity or torque of the driving motor, i.e. the force applied on the motor's axle, determines the vehicle's forward or backward acceleration (throttle) and the torque of the steering motor determines (with some granularity) the angle of the front wheels (steering).

Therefore, our controller needs the ability to let the motors turn in either direction at varying torque. This torque is proportional to the electric power applied to the DC motor. A naive method to control this power would be to vary the voltage over (and thereby the current through) the motor. This, however, requires relatively complex (analog) circuitry and such solution is energy-wise not efficient since much power is dissipated in the resistance required to vary the voltage. The most common solution to control a DC motor's torque is varying the *average* voltage over the motor by modulating that voltage using pulse-width modulation (PWM).

For PWM, the control voltage is a square wave and the average voltage of that square wave is altered by

changing the duty-cycle of the signal accordingly. The duty cycle of a digital signal is the fraction of time the signal is active, defined as the ratio between the pulse duration and the pulse period. The average voltage of the signal is thereby directly proportional to its duty cycle. For example, at a duty cycle of 0 % the signal is 0 V all the time, at 50 % the signal is a symmetric square wave with an average of half the signal's amplitude and at 100 % the signal is at maximum voltage all the time, meaning that the average voltage is maximum as well. At 0 % and 100 % the signal is not a square wave anymore, but rather a continuous voltage.

The frequency of the PWM signal is not very critical, but higher frequencies are desirable for the following reasons: (1) if the frequency is very low, the motor will not rotate smoothly with constant torque (2) the minimum duty-cycle needed to get the motor started in the presence of friction depends on the frequency, with higher frequencies giving better performance at lower speeds and (3) frequencies below 20 kHz produce audible sounds through vibration of the motor's axle that can generally be perceived to be annoying[1]. If a whistling car is not a problem, a frequency beyond 1 kHz is usually sufficient for toy cars.

Clearly, all that is needed to apply PWM for motor control is a simple power switch that intermittently switches the motor's (maximum) power on an off at high frequency to create the PWM square wave. But, the second requirement for our motor controller is that it can drive the DC motors in either direction. A property of the brushed DC motors on our cars is that their turning direction depends on the direction of the current flowing through their two leads. Therefore, we need an electrical circuit that provides the ability to alternate the direction of current flow through a motor of our follower car.

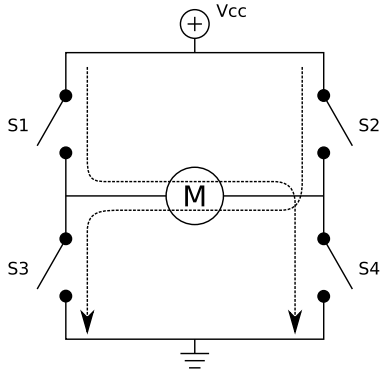


Figure A.2: H-bridge circuit configuration.

A circuit that implements such behavior is the H-bridge, which is so named for its configuration as shown in Figure A.2. It consists of four individually controlled switches (transistors). If in Figure A.2 switches *S1* and *S4* are closed and the other two are left open, the current will flow from left to right through the motor. If the state of these switches is inverted the current will flow the other way, making the motor turn in the opposite direction. These two alternative currents are shown in the figure as the dashed arrows starting at the supply voltage and ending at the common ground. Note that it is a very bad idea to simultaneously activate *S1* and *S3* or *S2* and *S4* as this will short-circuit the power supply over the switches. This will destroy the circuit of this situation is maintained for too long.

Our follower car does not have a mechanical brake actuator to stop the car. To reliably slow down the vehicle without applying negative throttle we build an inductive brake for the drive motor. An inductive brake works by short-circuiting the motor. A turning motor axle will then induce a short-circuited current in the motor's coils, which counters the axle's motion by self-induction. Although an inductive brake can be very effective, it is no match for a mechanical brake in comparison. In the H-bridge configuration of Figure A.2 short-circuiting the motor can be achieved by simultaneously activating either *S1* and *S2* or *S3* and *S4*. This short-circuits the motor either over the power supply or over the common ground.

Figure A.3 at page 148 shows the full schematic of our follower's motor controller. The follower uses a 9.6 V battery. This directly used as the supply voltage for the H-bridges presented to the right of the figure. For simplicity, both H-bridges are built using complementary bipolar junction transistors. The H-bridge of the drive motor is more complex, since it needs to be able to conduct more current and it includes an additional brake circuit. The steering motor does not draw as much current and it can suffice with less

[1]This can also be used to the designer's advantage to produce system beeps without the need for a buzzer.

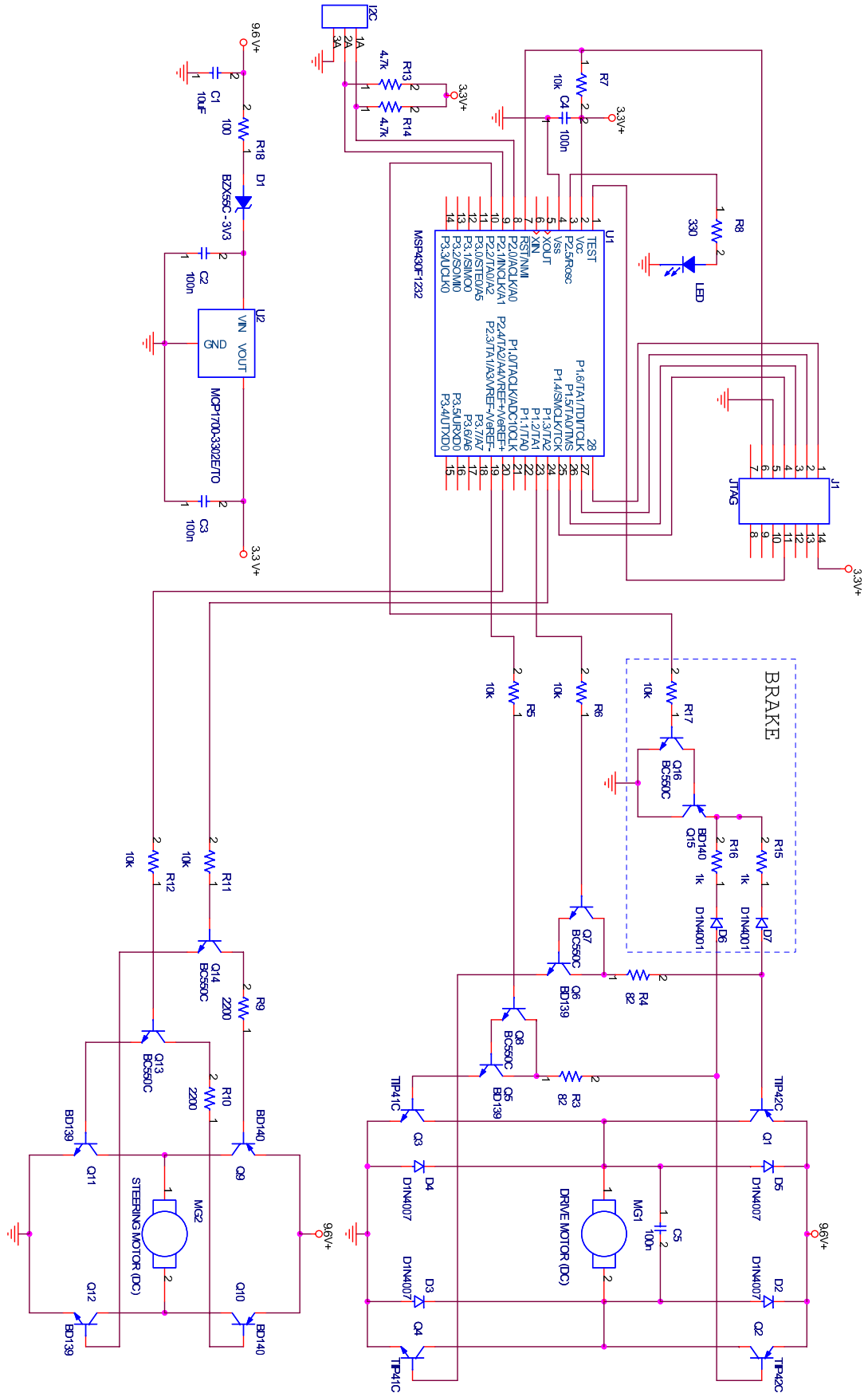


Figure A.3: Circuit schematic of the follower motor controller.

powerful transistors.

Conceptually, the two H-bridges are identical however. The crosswise activation of the transistors for activating the motor in either direction is implemented by transistors that connect the base lead of the top PNP transistor with the base lead of the corresponding bottom NPN transistor (e.g. Q14 connects Q9 and Q12 for the steering motor), thereby opening both transistors when it is activated. Thus, for each H-bridge, two of these transistors are used (four in a Darlington configuration for the drive motor). For each direction of movement, only one of these two transistors may be opened: as we explain above, activating these two transistors at the same time could destroy the circuit.

The separate brake control, as indicated by the dashed box in the figure, works in a similar manner, but it only controls the two top transistors by drawing current from their base leads to ground. This short-circuits the motor over the power supply. The two diodes prevent the brake from interfering with the normal H-bridge operation.

The motor controller uses a MSP430 processor similar to the one used on the sensor nodes, albeit less powerful: the MSP430F1232 [47] has only 8 KB flash ROM and 256 bytes RAM available. As we briefly explain in the next section, the software needed for this controller is very simple and straightforward, meaning that these limitations are no problem. The required 1.8 to 3.6 V power supply needed for the processor is derived from the 9.6 V battery supply voltage. This is shown in the lower left corner of Figure A.3. We use a linear voltage regulator to obtain a stable power supply of 3.3 V for the processor. However, the noise on the main power supply produced by motors and the PWM control is too significant for the regulator to handle. The low-pass filter formed by C1 and R18 filters the power supply to attain a cleaner DC voltage. If this filter is omitted, the processor resets itself at random intervals, rendering the controller useless.

The processor directly connects to the control transistors of the H-bridges. This is possible because the MSP430 can source more than 20 mA from its I/O ports and the control transistors do not draw much current at their base leads. As we describe in the next section, the ports we choose from the available I/O ports on the processor are those that can be configured as PWM output. The brake is an exception, as it uses an arbitrary digital output of the transistor. Other connections to the processor include its JTAG interface for programming, a simple status LED and the connections needed for the software I^2C bus. Figure A.1 of the previous section shows where this I^2C bus connects to the node.

A.2.2 Software

The main task of the motor controller's software is to translate motor commands from the node to the appropriate output signals for the H-bridge circuits we described in the previous section. An advantage of used the MSP430 processor is that it has hardware-based support to produce two different PWM signals with identical frequencies. We have just two motors, so two PWM signals are exactly what we need. Additionally, these PWM signals than be applied at a set of different output ports of the processor. This means that each PWM signal can be output on at least two different ports in a fully configurable manner. Therefore, no external hardware is needed to feed the PWM signal to either one of the two control transistors of each H-bridge. Choosing the direction of the motor is thus simply a matter of selecting the appropriate output port for the motor's PWM signal.

The PWM signals are generated using the processor's timer A. The frequency of this timer is derived from the processor frequency and this in turn determines the frequency of both produced PWM signals. As seen in Figure A.3 at page 148, no external crystal is used to select the processor frequency. Instead, we use the internal DCO oscillator [47], which is less stable, less precise and temperature dependent in terms of frequency. However, since the frequency is not a critical factor of the PWM signal, the frequency drift is no problem. The DCO is configured at its highest frequency of about 8 MHz. The PWM period is chosen to be 512 CPU cycles long. This means that the PWM duty cycle can be configured with a resolution of $\frac{1}{512}$ % and that the PWM frequency is $\frac{8000000}{512} = 15.6$ kHz. The PWM frequency lies just within the audible range, making particularly the drive motor whistle when the follower is starting to drive. Although the resolution of the PWM signal is $\frac{1}{512}$ %, we only use a resolution of $\frac{1}{256}$ % to make the torque values no larger than a byte[2].

[2]Therefore, we could double the PWM frequency, but for historical reasons the current frequency is retained.

The motor controller is connected to the sensor node through a software I^2C bus. The I^2C [38] protocol is implemented using a combination of interrupts and polling. Obviously, building a polled I^2C slave in software is never desirable. Unfortunately, the used processor has no hardware slave I^2C support. Alternatively, SPI can be used, which does have hardware support on the processor. For historical reasons we chose to use I^2C and, since all works well, converting to SPI would be a waste of time.

The I^2C slave only listens to incoming messages and never sends any replies other than the ACK bit specified in the I^2C protocol. At most five bytes are exchanged as the I^2C payload. The torque and direction values for the two motors are four bytes total with each byte represented as a writable I^2C 'register'. For each motor one register specifies the torque and the other specifies the action for that motor: *axle free* (no torque), *left turn*, *right turn* and *brake* (for the drive motor only). The first byte of the payload specifies at which register the write operation starts and the length of the I^2C payload determines how many of the four registers are written. This way, it is possible to write individual registers to change the parameters for only a single motor. Keep in mind that the motor controller retains the last known parameters even if the sensor node is reset or fails by some other means.

A.3 Gateway

The gateway node is a normal sensor node with the addition of a serial line driver. This way, it can be connected to a PC through a standard RS232 connection. Other than the serial link, the gateway node is void of external hardware. Due to the one-way nature of the radio protocol, the serial connection is used in one direction: from the gateway to the PC. On the PC, a program must continuously listen for incoming messages. The gateway does nothing more than receiving radio messages and forwarding them over the serial connection without any changes.

To send a message over the serial connection, it is wrapped into custom serial protocol frame. This frame is starts with a start byte (0x02) and ends with a stop byte (0x03). The byte after the start byte specifies the type of payload this frame carries. Currently, there are two types of messages: the forwarded radio messages from the mobile nodes and reports generated by the gateway itself specifying the occurrence of radio errors. The second type of message is particularly useful to identify the cause of packet loss. Commonly, noise on the used frequency causes verification errors (CRC) in the incoming radio messages. Alternatively, someone else may be using the selected frequency, which causes different error reports from the radio chip. The next byte of the frame specifies the length of the payload in bytes, which therefore cannot exceed 255 bytes. As we explain in Chapter 5, the radio messages have a fixed length of 32 bytes and therefore this maximum is adequate.

The receiver waits for the occurrence of the start byte, verifies the type of message and checks whether the length is consistent with that type of message. Finally, when the payload of the specified length is received, the presence of the stop byte is verified. If any of the checks fail, the message is discarded and the receiver scans the incoming serial stream for the next potential start byte. Note that our serial protocol does not include any form of redundancy check like CRC. We trust that the very short serial link we use is reliable.

The serial receiver at the PC can directly be implemented in MatLab. However, practical experience shows that the serial port support in MatLab suffers from software errors and therefore we implement a small binary program in C that listens for incoming messages and directly writes them to a text file in a comma-separated format. This format is directly readable in MatLab. Any other processing, like distinguishing the messages between leader and follower and the detection of packet loss is all performed in MatLab using the recorded messages.

Bibliography

[1] I.F. Akyildiz and I.H. Kasimoglu. Wireless sensor and actor networks: Research challenges. *Ad Hoc Networks Journal*, 2(4):351–367, 2004.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks (Amsterdam, Netherlands: 1999)*, 38(4):393–422, 2002.

[3] Jude Allred, Ahmad Bilal Hasan, Saroch Panichsakul, William Pisano, Peter Gray, Jyh Huang, Richard Han, Dale Lawrence, and Kamran Mohseni. Sensorflock: an airborne wireless sensor network of micro-air vehicles. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 117–129, New York, NY, USA, 2007. ACM.

[4] Ambient Systems. http://www.ambient-systems.net.

[5] G. Beni. From Swarm Intelligence to Swarm Robotics. *Swarm Robotics: SAB 2004 International Workshop*, 2005.

[6] J. Bih. Paradigm shift - an introduction to fuzzy logic. *IEEE Potentials*, 25(1):6–21, 2006.

[7] W. Burgard, M. Moors, C. Stachniss, and F.E. Schneider. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, 21(3):376–386, 2005.

[8] M.J. Caruso. Application of Magnetoresistive Sensors in Navigation Systems. In *Sensors and Actuators 1997*, volume SAE SP-1220, pages 15–21, February 1997.

[9] M.J. Caruso. Applications of Magnetic Sensors for Low Cost Compass Systems. *Position Location and Navigation Symposium, IEEE 2000*, pages 177–184, 2000.

[10] M.J. Caruso, T. Bratland, C.H. Smith, and R. Schneider. A New Perspective on Magnetic Field Sensing. *Sensors*, 15(12):34–46, 1998.

[11] R. Cassinis, G. Bianco, A. Cavagnini, and P. Ransenigo. Strategies for navigation of robot swarms to be used in landminesdetection. *Advanced Mobile Robots, 1999.(Eurobot'99) 1999 Third European Workshop on*, pages 211–218, 1999.

[12] M. Chang. Evaluation of accelerometers mounted on wireless sensor motes. Technical report, Department of Computer Science, University of Copenhagen, Denmark (Joint technical report with the University of Cambridge), January 2006.

[13] H.-H. Chiang, L.-S. Ma, J.-W. Perng, B.-F. Wu, and T.-T. Lee. Longitudinal and lateral fuzzy control systems design for intelligent vehicles. *Networking, Sensing and Control, 2006. ICNSC '06. Proc. of the 2006 IEEE Int. Conf. on*, pages 544–549, 23-25 April 2006.

[14] A. Dannenberg. Fuzzy logic motor control with msp430x14x. Technical report, Texas Instruments, February 2005.

[15] AK Das, R. Fierro, V. Kumar, JP Ostrowski, J. Spletzer, and CJ Taylor. A vision-based formation control framework. *Robotics and Automation, IEEE Transactions on*, 18(5):813–825, 2002.

[16] M. Marin-Perianu et al. Decentralized enterprise systems: A multi-platform wireless sensor networks approach. *IEEE Wireless Communications*, 14(6):57–66, 2007.

[17] E. Gaura and R. Newman. Wireless sensor networks: The quest for planetary field sensing. In *Local Computer Networks (LCN)*, 2006.

[18] J. Gaysse. Using fuzzy logic in low cost microcontroller to increase accelerometer performances. *Soft Computing in Industrial Applications, 2005. SMCia/05. Proceedings of the 2005 IEEE Mid-Summer Workshop on*, pages 6–11, 28-30 June 2005.

[19] D.L. Hall and J. Llinas. An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1):6–23, Jan 1997.

[20] D. Hayat. Traffic regulation system for the future automated highway. *Systems, Man and Cybernetics*, 3, 2002.

[21] S. J. Henkind and M.C. Harrison. An analysis of four uncertainty calculi. *IEEE Transactions on Systems, Man and Cybernetics*, 18(5):700–714, 1988.

[22] N.E. Hodge, L.Z. Shi, and M.B. Trabia. A distributed fuzzy logic controller for an autonomous vehicle. *J. Robot. Syst.*, 21(10):499–516, 2004.

[23] T.J. Hofmeijer, S.O. Dulman, P.G. Jansen, and P.J.M. Havinga. AmbientRT - real time system software support for data centric sensor networks. *Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004. Proceedings of the 2004*, pages 61–66, 14-17 Dec. 2004.

[24] J. Jantzen. *Foundations of Fuzzy Control*. Wiley, 2007.

[25] M. Kam, Xiaoxun Zhu, and P. Kalata. Sensor fusion for mobile robot navigation. *Proceedings of the IEEE*, 85(1):108–119, Jan. 1997.

[26] G.J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logc: Theory and Applications*. Prentice Hall, 1995.

[27] K.R.S. Kodagoda, W.S. Wijesoma, and E.K. Teoh. Fuzzy speed and steering control of an agv. *Control Systems Technology, IEEE Transactions on*, 10(1):112–120, Jan 2002.

[28] N. Kottege and U.R. Zimmer. Relative localization for AUV swarms. In *International Symposium on Underwater Technology*, 2007.

[29] R.J. Larsen and M.L. Marx. *An Introduction to Mathematical Statistics and Its Applications - Third Edition*. Prentice Hall, 2000.

[30] M.F.R. Lee, K.Stanley, and Q.M.J. Wu. Implementation of sensor selection and fusion using fuzzy logic. *IFSA World Congress and 20th NAFIPS International Conference, 2001. Joint 9th*, 1:328–333 vol.1, 25-28 July 2001.

[31] D. Lymberopoulos, Q. Lindsey, and A. Sawides. An Empirical Characterization of Radio Signal Strength Variability in 3-D IEEE 802.15. 4 Networks Using Monopole Antennas. *Wireless Sensor Networks: Third European Workshop, EWSN 2006, Zurich, Switzerland, February 13-15, 2006: Proceedings*, 2006.

[32] E. H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1):1–13, 1975.

[33] M. Marin-Perianu and P. J. M.Havinga. D-FLER: A distributed fuzzy logic engine for rule-based wireless sensor networks. In *International Symposium on Ubiquitous Computing Systems (UCS)*, pages 86–101, 2007.

[34] MEMS and Nanotechnology Clearinghouse. http://www.memsnet.org.

[35] F. Mondada, L.M. Gambardella, D. Floreano, S. Nolfi, J.-L. Deneuborg, and M. Dorigo. The cooperation of swarm-bots: physical interactions in collective robotics. *Robotics & Automation Magazine, IEEE*, 12(2):21–28, June 2005.

[36] D. Naffin and G. Sukhatme. Negotiated formations, 2004.

[37] H.T. Nguyen, N.R. Prasad, C.L. Walker, and E.A. Walker. *A First Course in Fuzzy and Neural Control*. CRC Press, 2003.

[38] Philips Semiconductors. *The I^2C-Bus Specification*, January 2000. Version 2.1.

[39] PNI Corporation. *Application Note: Discrete Circuit for Magneto-Inductive Sensors*, December 2003.

[40] PNI Corporation. *PNI ASIC: 3-Axis Magneto-Inductive Sensor Driver and Controller with SPI Serial Interface*, November 2006.

[41] PNI Corporation. *PNI MicroMag 3: 3-Axis Magnetic Sensor Module*, June 2006. Datasheet.

[42] J. Pugh and A. Martinoli. Small-Scale Robot Formation Movement Using a Simple On-Board Relative Positioning System. In *International Symposium on Experimental Robotics 2006*, 2006.

[43] V. N. S. Samarasooriya and P. K. Varshney. A fuzzy modeling approach to decision fusion under uncertainty. *Fuzzy Sets and Systems*, 114(1):59–69, 2000.

[44] PG Savage. What do accelerometers measure? Technical report, Strapdown Associates, Inc., 2005. http://www.strapdownassociates.com/Accels%20Measure.pdf.

[45] Sparkfun Electronics: Triple Axis Accelerometer Breakout - LIS3LV02DQ. http://www.sparkfun.com/commerce/product_info.php?products_id=758.

[46] ST Microelectronics. *LIS3LV02DQ - MEMS INERTIAL SENSOR: 3-Axis - 2g/6g Digital Output Low Voltage Linear Accelerometer*, October 2005. Data sheet.

[47] Texas Instruments. *MSP430x11x2, MSP430x12x2: Mixed Signal Microcontroller*, August 2004. Data sheet, SLAS361D.

[48] Texas Instruments. *MSP430x15x, MSP430x16x, MSP430x161x: Mixed Signal Microcontroller*, February 2005. Data sheet, SLAS368E.

[49] D. Titterton and J. Weston. *Strapdown Inertial Navigation Technology, 2nd Edition*. Institution onf Electrical Engineers, 2004.

[50] K. Tuck. Tilt sensing using linear accelerometers. Technical Report AN3461, Freescale Semiconductor, 2007.

[51] Z. Wang, P. Chen, Z. Song, Y. Chen, and K.L. Moore. Formation control in mobile actuator/sensor networks. *Unmanned Ground Vehicle Technology VII*, 5804(1):706–717, 2005.

[52] Z. Wang, Z. Song, P.-Y. Chen, A. Arora, D. Stormont, and Y. Chen. Masmote a mobility node for mas-net (mobile actuator sensor networks). *Robotics and Biomimetics*, pages 816–821, 22-26 Aug. 2004.

[53] R. Yates. Fixed-point arithmetic: An introduction. Technical report, Digital Signal Labs, August 2007. http://www.digitalsignallabs.com/fp.pdf.