

University of Twente

EEMCS / Electrical Engineering
Control Engineering



Wireless state feedback and control of a pipe inspection robot

Corné Doggen

MSc report

Supervisors:

prof.dr.ir. S. Stramigioli
dr.ing. R. Carloni
ir. E.C. Dertien
dr. C.J.A. Pulles

December 2010

Report nr. 029CE2010
Control Engineering
EE-Math-CS
University of Twente
P.O.Box 217
7500 AE Enschede
The Netherlands

Summary

Inspection of gas distribution mains in urban areas is a time consuming and expensive task. This is why a first robot prototype has been built at the Control Engineering Group of the University of Twente. Goal of this robot, called PIRATE, is to inspect the low pressure gas distribution network in the Netherlands from inside the pipe autonomously on leaks or other damages. The goal of this project is to use mostly the internal state data to perform this complex task of autonomous navigation.

The development is done partially on the robot itself and partially outside the robot in the 20sim environment. A bidirectional wireless communication channel operating in the 2,4GHz range is set up to perform wireless state feedback to the outside world. Furthermore, the link is used for sending parameters to the robot. The state data is transmitted to a prototype of a docking station which will in turn transmit this data to a PC. In this way the pirate robot can be monitored and controlled wirelessly which enables the user to get more insight in the complex tasks of tackling environmental features like T-joints or 90 degree bends.

On the PC side world modeling is done using the 20sim environment. The 3D animation editor is used to construct a graphical representation of the pirate robot and can perform a realtime animation. In the world model a pipe diameter and environmental feature estimation is made which can be used for further development of an autonomous inspection robot.

Preface

With this thesis I conclude my Master of Science program in Electrical Engineering at the University of Twente. A long educational road passing through MTS and HTS gave me the opportunity to learn all the theoretical and practical aspects of electronics. It looks like a puzzle of one thousand pieces has become an easy one of ten pieces.

With the gained knowledge my view to the world is changed during my study at the university. There will be always questions around the world and have to be answered by people like us, the engineers. Sometimes it is hard to find the correct answer on a question or a solution for a given problem, but that makes life challenging. In my opinion there is nothing more exciting than to be an engineer.

Fortunately, I got some support during my research and I would like to thank all of them. First of all the people of the Control Engineering group, my graduation committee and daily supervisor Edwin Dertien for their tips and support. Secondly I would like to thank my family. In particular my parents for their mental support and keeping me motivated at all times. At last I want to thank my girlfriend who was always there when I needed her. The conversations which I had with her calmed me down when I had a stressful time. Last but certainly not least I would like to thank some fellow students for the discussions and fun we had during the coffee breaks.

Corné Doggen
Enschede, December 2010

Contents

1	Introduction	1
1.1	Background	1
1.2	The PIRATE project	1
1.3	Assignment	2
1.4	Report outline	2
2	Analysis	3
2.1	Introduction	3
2.2	Naming conventions	3
2.3	Distributed control architecture	3
2.4	Control of pirate robot	6
2.5	Recent changes	8
2.6	I2C-Communication bus	8
2.7	Wireless communication	14
2.8	Design requirements	22
3	Design and implementation	23
3.1	Introduction	23
3.2	Pirate data acquisition	23
3.3	Docking station	28
3.4	World modeling	30
4	Testing and simulation	37
4.1	Testing the wireless bi-directional data link	37
4.2	Calibration joint angles	38
4.3	Pipe diameter and environmental feature estimation	39
4.4	Conclusions of the pipe diameter and environmental feature estimation	44
5	Discussion	45
5.1	Conclusions	45
5.2	Recommendations	45
A	Mathematical derivations	49
A.1	Measuring tilt using a three axis accelerometer	49
B	Power supply analysis	51
B.1	Introduction	51
B.2	Encountered problems	51

B.3	Software crash analysis	51
B.4	Conclusions	52
C	LPC2148 USB Bootloader	53
C.1	What is a bootloader	53
C.2	Why using LPC2148 USB bootloader	53
C.3	Bootloader performance	53
C.4	Using the bootloader	54
C.5	Conclusions	54
D	Docking station Installation manual	55
E	Control commands	59
F	List of files	61
F.1	20sim DLL	61
F.2	20sim world modeling	61
F.3	Pirate firmware	61
F.4	Docking station firmware	61
F.5	LPC2148 USB Bootloader	61
G	Schematics	63
G.1	Mainboard	63
G.2	Motor controller	67
G.3	Transceiver nRF24L01 Module with RP-SMA	71
	Bibliography	73

1 Introduction

1.1 Background

1.1.1 The gas network in the Netherlands

The national network of gas distribution mains can be divided into a high-pressure (1-8 bar) network for national distribution stretching 20.000 km and a low pressure network (30 mbar to 100 mbar) for local distribution with a length of roughly 100.000 km. The low-pressure net covers most of the urban area's. Therefore this network has the highest priority with regard to risks for public health and safety. Replacement of pipe sections in an urban area is expensive, so it is important to have accurate data on the locations of leaks and damaged pipe sections. Pipes of gray cast iron and white PVC are most likely to cause leakage. Gray cast iron is especially sensitive to corrosion. Polyethylene(PE) is sensitive to point-loads (by tools) and tension (bend, stretch) for example caused by tree roots. The main causes of leakage can be summarized as follows: creep, tension, brittleness, impacts, inferior connections, porous rubber sealing, corrosion. There is another issue with the gas mains, besides pipe leakage: the existing network is not well documented. New sections that have been created in the last decade are well documented, but there is not much information available on older sections in the network. Detailed information possibly never existed or got lost in company merges, takeovers and (computer)system changes.

1.1.2 Current methodology for pipe inspection

Currently, the low pressure distribution nets are only inspected by conventional leakage searching above ground. This is a labor-intensive process and does not give any information about layout and quality of the pipe, only leaks that can be 'smelled' can be detected. The accuracy of above ground detection is just several meters. By (Dutch) law, every segment of the gas pipe network has to be inspected every 5 years, but with the current methods this is nearly impossible. High pressure mains are already inspected by robotic systems. These systems are hardly full grown autonomous robots, but more passive data loggers. According to Kiwa Gastec(gas technology research), every year 2000 leaks are being found with the conventional leak inspection methods and 6000 leaks are reported by the public. Liander(energy company) has had 9000 public leak reports in 2005, from which 1000 were false alarm, 2000 leaks were found in house and 6000 were found in the gas distribution network (Dertien, 2006).

1.2 The PIRATE project

The Pirate project focusses on the development of an inspection robot for internal gas pipe inspection. The robot can navigate autonomously through gas pipes with inner diameters between 57 and 119 millimetres. The robot inspects the gas network for leaks and deformations and records information on the exact location of the pipes. The robot is called Pirate, which is an abbreviation for Pipe Inspection Robot for AuTonomous Exploration.

In autumn 2007 a first prototype of the robot has been built at the Control Lab of the University of Twente. A picture of this prototype is shown in figure 1.1. The Pirate prototype consists of 7 joints and 7 modules: 2 drive modules, 2 bend modules, 2 payload modules and 1 rotation module (Vennegoor op Nijhuis, 2007). In total 7 motors are used to drive and position the robot: 2 to drive the driven wheels, 4 to control the angular joints and 1 for the rotational joint. These motors are controlled by 5 motor controllers, which have identical design but different implementation (Ansink, 2007). The overall robot control is done by the main controller (Dertien, 2007) which is placed in one of the payload modules and can be controlled by a front-end Graphical User Interface (GUI) made in Matlab.

A vision system has been developed which is able to detect bends, joints and obstacles, but this system needs to be scaled down so it fits on the robot (Drost, 2009). The robot has 8 wheels of which 2 are driven. In normal operation the robot clamps itself into the pipe by making a V-shape with the first and last two modules.

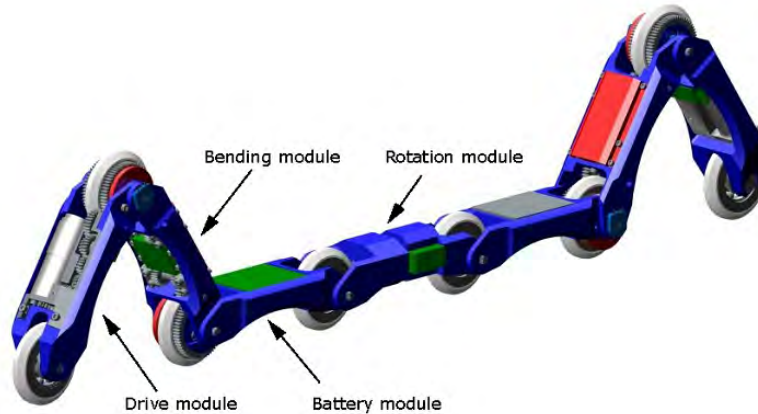


Figure 1.1: PIRATE pipe inspection robot

The design makes it possible to move besides obstacles like tree roots as well as smooth bends and T-junctions (Reemeijer, 2010). When the robot is fully developed and operational, multiple Pirate robots will swarm through the gas network together mapping out the network structure and finding leaks and weak spots, protecting the overground areas from potential disaster.

1.3 Assignment

In order to develop a pipe inspection robot which operates autonomously underground, internal state data of the robot is required to perform such a system. Due to the complexity of manouvring through the underground gas network the development is partially done on the robot itself and partially outside the robot in a development environment on the pc. In this assignment the pirate robot must be able to transmit its state data wireless to the pc. On the pc world modeling is performed inside the 20sim environment using this state data. The world model has to be able to estimate the pipe diameter and the environment the robot is located in underground.

1.4 Report outline

- Chapter 2: In this chapter the problem of this assignment is analysed. The performance of the internal bus communication present on the robot is investigated. Subsequently, a wireless communication is set up for testing purposes using the internal transceiver module on the robot.
- Chapter 3: The proposed design and its implementation is described in this chapter. Collecting state data from the motorcontrollers and composing a statevector from this data in the maincontroller is described first. The wireless transmission of this statevector to the builded prototype of a docking station is described afterwards. Finally the state data is used for world modeling in the 20sim animation editor.
- Chapter 4: This chapter covers the results of the implemented design. The pipe diameter and the environmental feature the robot is located in is estimated in the world modeling process.
- Chapter 5: Conclusions and recommendations for future work are givin in this last chapter.

2 Analysis

2.1 Introduction

This research mainly focusses on the wireless communication of the pirate robot. The wireless communication can be used for state feedback (for testing and simulation in the laboratory) on the one hand and making data exchange to the docking station possible on the other hand. In this research only the wireless state feedback is treated.

2.2 Naming conventions

The robot consists of seven modules which are numbered from the back to the front. The index number of the relevant module is displayed in the form of roman numerals. Module IV is separated in IVi and IVii by a rotational joint.

The robot has seven degrees of freedom (DOF) and are represented by joints. On each joint a wheel is attached, except for the rotational joint. The numbering of the joints and wheels is done similar to the modules. Furthermore, joints and wheels have the same index number as the module on the right side of it. On the robot are five motorcontrollers present, which are indicated with the letters A to E. These motorcontrollers are attached to the modules I, II, IVii, VI, and VII respectively. An overview of these naming conventions is depicted in figure 2.1.

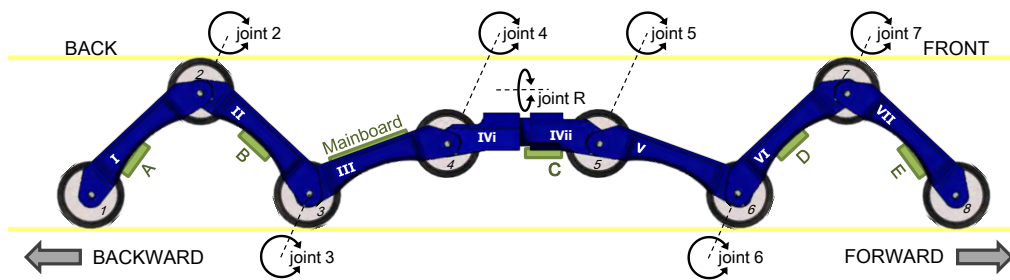


Figure 2.1: Naming conventions of the pirate robot

2.3 Distributed control architecture

2.3.1 Introduction

Since the robot consists of a modular design, the choice has been made to use distributed control. Using a distributed control system has a lot advantages in comparison to a single controller. A single controller can not handle all the control sequences and the data acquisition duo to space restrictions like the amount of wiring that is needed to interface all the motors and sensors. This distributed control system consists of five small local controllers and a main-controller which are interconnected through a communication bus. Since the local controllers are controlling the motors attached to them, they are called motorcontrollers in this report.

Because of its small number of wires, small code overhead, scalability and ease of implementation, I2C (also called TWI) has been chosen as communication protocol for the robot (Ansink, 2007). I2C is a serial 2-wire protocol with a maximum speed of 400[kbps]. It is very suitable for a system with a large number of modules on a bus, up to 128 in the normal address range. I2C uses 2 wires: a clock and a bidirectional data line. The communication speed is limited by the bus capacitance, which depends on the length of the wires. In this project the I2C-bus is set up as a single master multiple slaves configuration whereby the maincontroller will be the 'master' and the motorcontrollers the 'slaves'. Various commands can be send from the maincontroller

through the I²C-bus to the motorcontrollers. Figure 2.2 shows how the motorcontrollers internally processes these commands in the distributed control system.

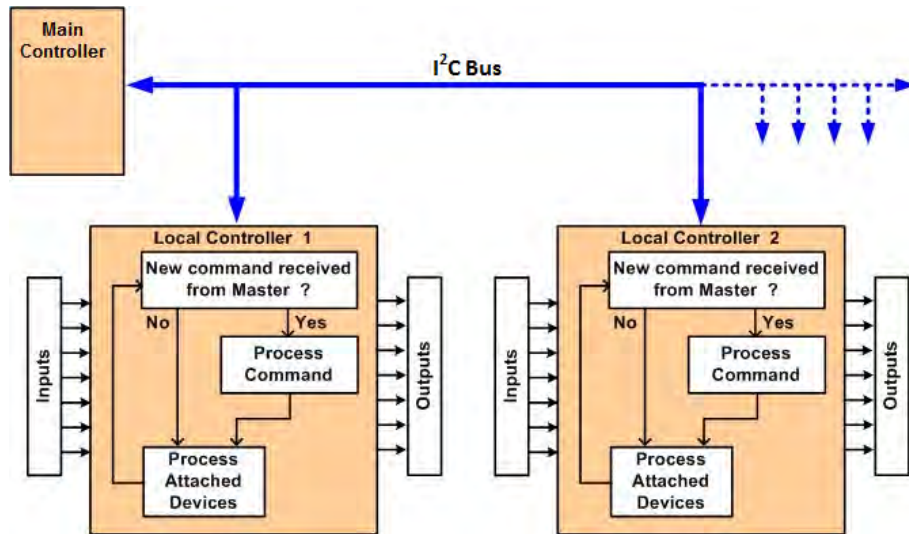
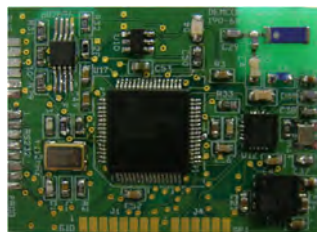


Figure 2.2: Main- and localcontrollers interconnected by I²C-bus

2.3.2 Maincontroller

The maincontroller consists of an ARM7 processor, a 3D acceleration sensor to determine the orientation of the robot, a memory card slot for data storage, a temperature sensor, and a radio module for external wireless communication. All these components are integrated in the so called mainboard which can be seen in figure 2.3 (Dertien, 2007).



(a) Mainboard top



(b) Mainboard bottom

Figure 2.3: Mainboard of pirate robot

For the microcontroller on the mainboard the choice has been made for the LPC2148 from NXP. Due to the tiny size and low power consumption, the LPC2148 is ideal for applications where miniaturization is a key requirement. Various serial communication interfaces ranging from a USB 2.0 Full Speed device, multiple UARTs, SPI, SSP to I²Cs, and on-chip SRAM of 32kB, make this device very well suited for a communication gateway in this project. Furthermore, the maincontroller can also be used to handle other tasks like data acquisition, decision making, global failsafe tasks etc. In appendix G can be seen how all the components on the mainboard are interconnected.

2.3.3 Motorcontrollers

The robot consists of seven modules of which five modules contain active components like actuators and sensors (Ansink, 2007). Each module needs a local controller that interfaces with the different actuators and sensors present in that module. Furthermore, the motorcontrollers are able to communicate with the maincontroller through the I²C or TWI (Two Wire Interface) bus. In figure 2.4 a motorcontroller board is depicted.

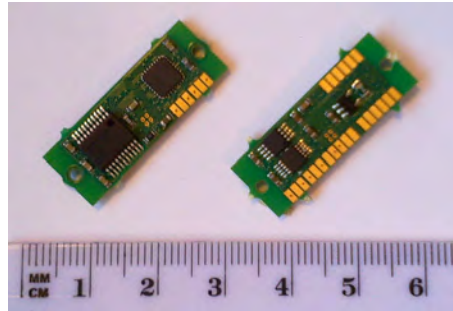


Figure 2.4: Motorcontroller board

Configurations

Designing different local controllers for each module is time consuming and expensive. This is why a universal local controller design is used. In this case on every active module the same local controller can be placed with a specific configuration.

A motorcontroller consists of a Atmel ATmega168 microcontroller, a double H-bridge (to drive 2 motors) and a few other components for the powersupply. Looking at the inputs of the board there can be connected a maximum of four potentiometers and two encoders which can be used as inputsignals for the control feedback loops running on the controllers (see figure 2.5). There are three different configurations made for the drive, bend and rotation module.

- Drive module: 1 motor, 1 encoder
- Bend module: 2 motors, 4 potentiometers
- Rotation module: 1 motor, 1 encoder, 1 potentiometer

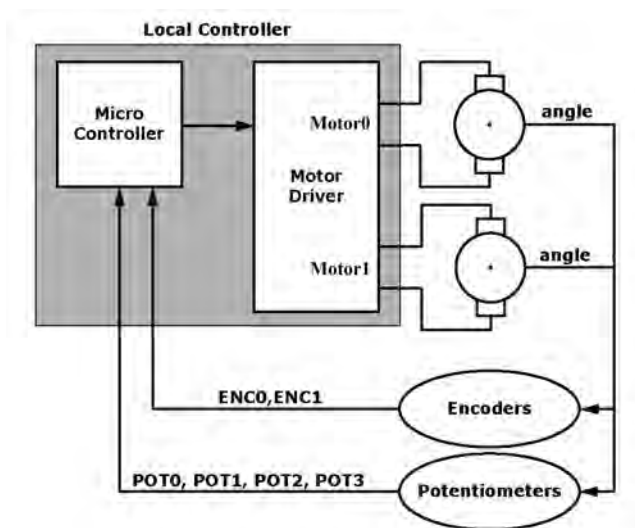


Figure 2.5: Hardware connections of motorcontrollers

Since there are two drive modules, two bend modules and one rotation module present on the robot, the total amount of motors equals seven. All these motors have their own function which is given in table 2.1. For the bendmodules there are two motors connected to the related motorcontroller and are indicated with an indexnumber '0' and '1'.

Control mode

As already mentioned, these motors can individually be controlled by PID controllers. Therefore, on each motorcontroller there can be run two PID loops. These PID controllers can run in position control, velocity control or torque control. For each control mode a appropriate input

Motorcontroller ID	Motornr.	Function
A(65)	0	Drive
B(66)	0	Clamp (adjust angle joint 2)
	1	Clamp (adjust angle joint 3)
C(67)	0	Rotation
D(68)	0	Clamp (adjust angle joint 6)
	1	Clamp (adjust angle joint 7)
E(69)	0	Drive

Table 2.1: Functions of all present motors on the robot

signal have to be selected. From table 2.2 it can be seen that there are eight PID control modes besides a direct PWM and a IDLE mode.

For an example let's assume one wants to control the motor of a drivemodule in velocity control mode. From table 2.2 it can be seen that control modes '2' and '4' satisfy velocity control. Which one to choose depends on which port the attached encoder on the motor is connected to the motorcontroller. The motorcontroller of the drive module has only one motor to drive and is connected to motor port '0'. In this example the correct control mode should be mode '2'.

Control mode	
0	Idle
1	Enc0 Position
2	Enc0 Velocity
3	Enc1 Position
4	Enc1 Velocity
5	Pot0 Position0
6	Pot1 Torque0
7	Pot2 Position1
8	Pot3 Torque1
9	Direct Pwm

Table 2.2: Control modes of motors

2.4 Control of pirate robot

2.4.1 Commandline parser

The current prototype can be controlled using a RS232 connection between the maincontroller and a terminal program. Commands can be send from the terminal to the maincontroller, whereby the maincontroller will function as a transparant command line parser. All the commands the maincontrollers receives will be parsed to the relevant motorcontroller. These commands always begin with a exclamation mark or a question mark which indicates if the command is a write or read operation respectively. Directly after this mark comes the letter which indicates the function that has to be performed, followed by a maximum of three arguments. Where the first argument is always the address of the relevant motorcontroller, have the other arguments a function specific meaning. A overview of all these commands can be found in appendix E.

In order to receive the status of drivemodule A(65) the following command has to be entered:

```
>> ?S 65
```


In table 2.3 the description of the statusbyte is given. If something went wrong or an exception occurred, the status of the motorcontrollers can be read at any time and will give the user comprehension of the state of the robot.

StatusByte	
Bit	Description
0	ResetType:2
1	
2	OperatingMode:2
3	
4	BridgeMode:1
5	OverCurrent:1
6	PositionOverflow:1
7	CommunicationError:1

Table 2.3: Statusbyte of motorcontrollers

2.4.2 Matlab GUI

When controlling the robot with the before mentioned commandset via a terminal program, there is not much flexibility for the user. Also problems arise when one wants to give two commands simultaneously (i.e. for driving the robot, both the drive modules have to be activated).

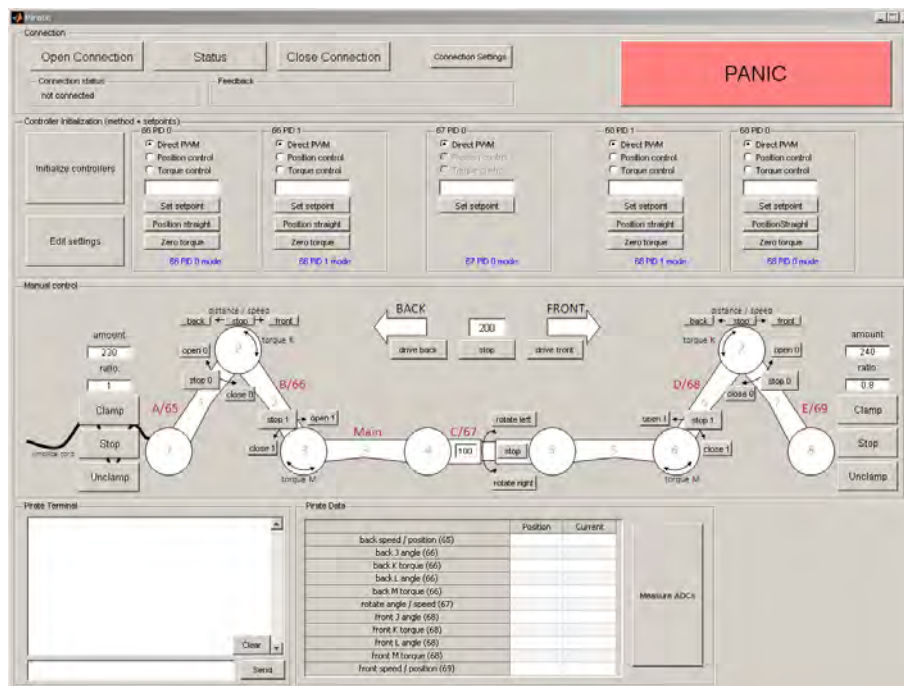


Figure 2.6: Extended matlab GUI

In order to make life easier a matlab GUI is developed and later on extended (Reemeijer, 2010). This GUI allows users to adjust every joint of the robot individually, set the control mode of the motorcontrollers, set setpoint for the PID controllers and measure the ADC values of all the motorcontrollers. Furthermore the GUI provides data logging functionality which enables the user to make a recording of a run the robot makes. Unfortunately, this datalogging feature operates at a poor update frequency of 0.4Hz. Figure 2.6 shows the GUI front-end.

2.5 Recent changes

2.5.1 Reduce DOF

In order to reduce the DOF of the robot it is possible to remove one of the payload modules without major modifications and implications. Doing so removes one of the joints. From earlier research it is proven that a decrease of DOF would result in more easily manouvring through a T-joint or mitered joint.

Payload module V is the most simple to remove, since it was not used in the prototype. It was intended as a housing for batteries and measurement electronics which are both not present yet. If the new wheel design from (Burkink, 2009) proves to work, the drive modules also become (partially) available for batteries and electronics. For convenience the numbering of the wheels, joints and modules will be kept the same, which means that wheel 5, joint 5 and module V are missing. Figure 2.7 shows the robot with the removed Module V, wheel 5 and joint 5.

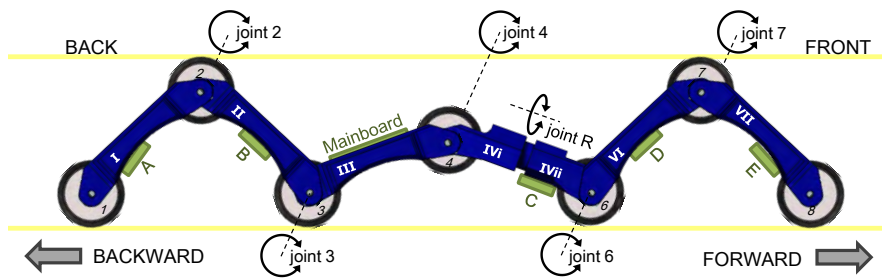


Figure 2.7: Main- and localcontrollers interconnected by I^2C -bus

A disadvantage of this new configuration is that the rotation module is not in the middle of the robot. To perform a rotation the rotation module should be in line with the tube the robot is in. Since the joints 4 and 5 are non-actuated in the previous design (see figure 2.1) the rotation module will mostly all of the time automatically be in line with the tube. To perform a rotation using this new configuration precaution must be taken of joint 6 to make sure the rotation module is in line with the tube.

2.6 I2C-Communication bus

2.6.1 Introduction

Since the internal communication of the robot rely on the I2C-bus, this protocol has been investigated to determine if the performance of the bus meets the requirements. Communication speed and a reliability analysis will be topics of this section.

I2C versus TWI

TWI stands for Two Wire Interface and for most marts this bus is identical to I2C. The name TWI was introduced by Atmel and other companies to avoid conflicts with trademark issues related to I2C. A description of the capabilities of TWI interfaces can be found in the data sheets of corresponding devices. Expect TWI devices to be compatible to I2C devices except for some particularities like general broadcast or 10 bit addressing.

2.6.2 Communication speed

For the I2C-bus there can be four modes of operation be distinguished. The first I2C specification dated from 1982 had a limit of 100Kbit/s. In 1992, a newer version of the I2C specification was released. This new specification contained some additional sections covering FAST mode and 10-bit addressing. The protocol, bus levels, capacitive load etc. remain unchanged.

However, the data rate has been increased to 400 Kbit/s. In May 2006 NXP, formerly Philips introduced Fast-mode Plus allowing up to 1 Mbit/s. Additionally, there is the high speed mode running the bus with up to 3.4 Mbit/s. In table 2.4 a overview of these modes and their compatibility with the TWI interface is given.

I2C		TWI
Mode	Datarate	
Standard	< 100kbit/s	Supported
Fast mode	< 400kbit/s	Supported
Fast mode plus	< 1Mbit/s	Not Supported
High speed mode	< 3,4Mbit/s	Not Supported

Table 2.4: Operation modes of I2C and TWI bus

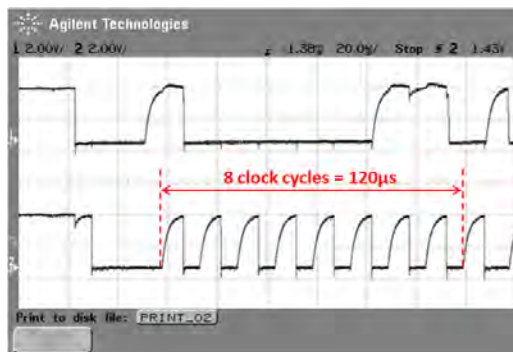
Requirements

In this research the I2C bus mainly will be used for data aquisition from the motorcontrollers (slaves) to the maincontroller (master). To make sure the bus can handle the data which have to be transferred, the highest possible datarate for the bus will be selected.

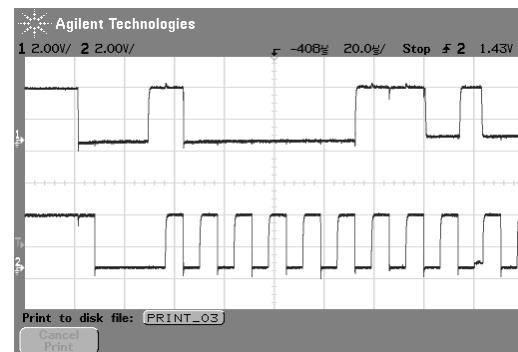
Since the motor controllers contain a Atmel chip as controller, the maximum speed is limited at 400Kbit/s (see table 2.4). The master in the system will always generate the clock signal, and therefore the clock speed should be set in the LPC2148 to 400 KHz.

Analysis of the bus signals SDA and SCL

It is obvious to verify the communication speed of the bus by monitoring the clock signal (SCL) and data signal (SDA) with a oscilloscope. Doing so, the bus seems not to be running on the desired 400KHz. From figure 2.8 it can be seen the clock signal runs at 66,7KHz.



(a) $R_p = 4.7K\Omega$



(b) $R_p = 330\Omega$

Figure 2.8: SDA and SCL signals with different pull-up resistors

$$f_{bus} = \frac{8}{120\mu s} = 66,7KHz \quad (2.1)$$

The LPC2148 features two I2C busses namely, I2C0 and I2C1. Here the I2C0 bus will be used. Software must set values for the registers I2SCLH and I2SCLL to select the appropriate data rate and duty cycle. I2SCLH defines the number of PCLK cycles for the SCL HIGH time, I2SCLL defines the number of PCLK cycles for the SCL low time. The frequency is determined by the following formula (PCLK is the frequency of the peripheral bus APB = 60MHz).

$$I2C_{bitfrequency} = \frac{PCLK}{I2SCLH + I2SCLL} \quad (2.2)$$

The values for I2SCLL and I2SCLH should not necessarily be the same. Software can set different duty cycles on SCL by setting these two registers. For example, the I2C-bus specification defines the SCL low time and high time at different values for a 400 kHz I2C rate. The value of the register must ensure that the data rate is in the I2C data rate range of 0 through 400 kHz.

Both I2SCLL and I2SCLH were set to 400, which results in a theoretical clock frequency of 75KHz from 2.2. There is a discrepancy between the actual clockrate and the measured clock-rate of 75KHz - 66,7KHz = 8,3KHz. This deviation is caused by clock stretching and termination, which will be treated in one of the following sections. The register values are adapted to a value of 75 to ensure a clock frequency of 400KHz.

Termination versus capacitance

The I2C bus transmits data and clock with SDA and SCL. SDA and SCL are open-drain (also known as open-collector in the TTL world), that is I2C master and slave devices can only drive these lines low or leave them open. The termination resistor R_p pulls the line up to V_{cc} if no I2C device is pulling it down (see figure 2.9). This allows for features like concurrent operation of more than one I2C master or clock stretching.

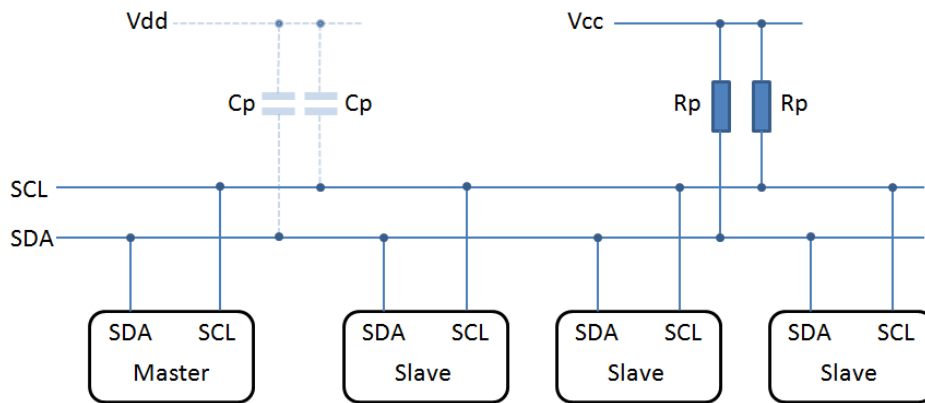


Figure 2.9: Termination resistors R_p to pull the lines to V_{cc} when the bus is in IDLE state

Together with the wire capacitance C_p the termination resistor R_p affects the temporal behaviour of the signals on SDA and SCL. While I2C devices pull down the lines with open drain drivers or FETs which can in general drive at least about 10mA or more, the pull-up resistor R_p is responsible to get the signal back to high level. R_p commonly ranges from 1 k Ω to 10 k Ω , resulting in typical pull-up currents of about 1 mA and less. This is the reason for the characteristic sawtooth-like look of I2C signals. In figure 2.8a every 'tooth' shows the charge-characteristic of the line on the rising edge and the discharge-characteristic on the falling edge.

C_p and R_p effectively limit the maximum data rate which can be transferred over SDA and SCL. Hence, the actual clock rate may be lower than the nominal clock rate e.g. in I2C buses with large rise times due to high capacitances. A high C_p can be compensated with a low R_p and vice versa. Long wires increase C_p dramatically. Note that the I2C standard limits C_p to the maximum value of 400 pF. However, with an appropriate termination resistance it is often possible (although not recommended) to operate I2C buses with higher capacitance. I2C connections should always be as short as possible and connected by a suitable wiring pattern.

Clock stretching

When the master is reading from the slave, it's the slave that places the data on the SDA line, but it's the master that controls the clock. What if the slave is not ready to send the data? If the slave device will be a EEPROM for example this is not a problem, but when the slave device is actually a microprocessor with other things to do, it can be a problem. The microprocessor on the slave

device will need to go to an interrupt routine, save its working registers, find out what address the master wants to read from, get the data and place it in its transmission register. This can take several microsecond to happen, meanwhile the master is sending out clock pulses on the SCL line that the slave cannot respond to.

One of the more significant features of the I2C protocol is clock stretching. An addressed slave device may hold the clock line (SCL) low after receiving (or sending) a byte, indicating that it is not yet ready to process more data. The master that is communicating with the slave will attempt to raise the clock to transfer the next bit, but must verify that the clock line was actually raised. If the slave is clock stretching, the clock line will still be low (because the connections are open-drain). Fortunately, the hardware I2C ports on most microprocessors will handle this automatically.

Read/write operations

Since the maincontroller is the master on the bus, read or write operations should always be initiated by the maincontroller. Reading from a slave is a little bit more complicated in comparison with writing to the slave. Both the operations will be explained in this section. To give a overview of these operations the I2C software is investigated and translated to flowcharts which can be seen in the figures 2.10 and 2.11. The flowcharts must be read in perspective of the maincontroller (master).

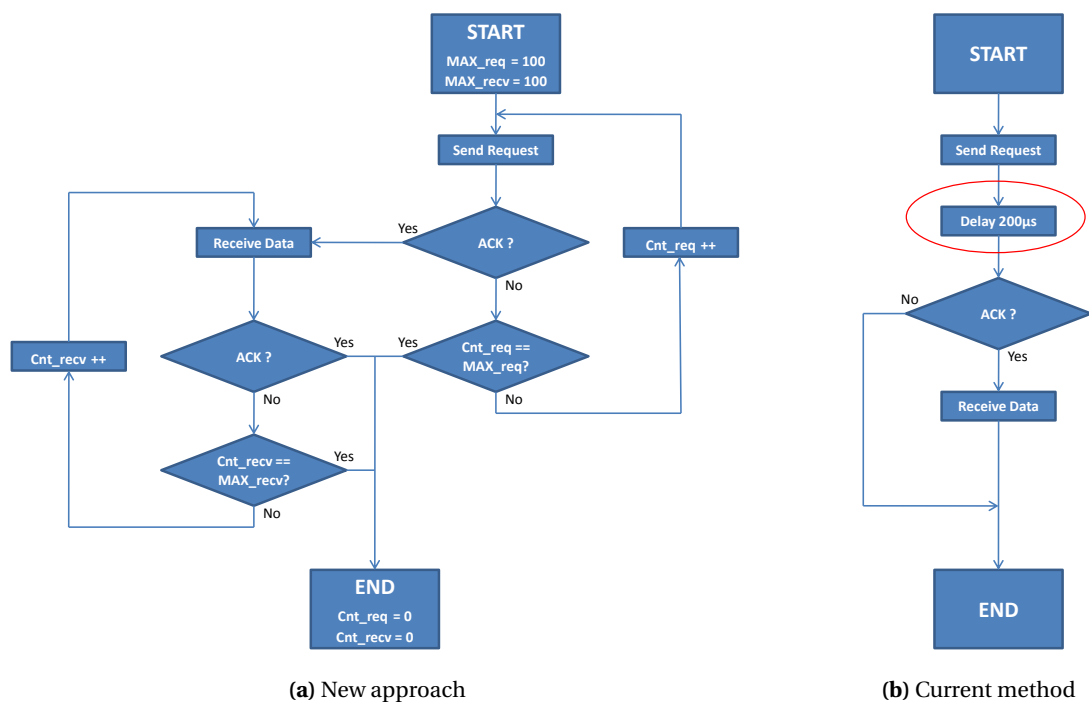


Figure 2.10: Procedure for reading from a slave device

To perform a read operation first there is send a request to the relevant slave. After that there is a delay of $200\mu\text{s}$ which doesn't make any sense since the ACK bit is verified after this delay. If the slave acknowledged the request data can be received from the slave, and operation can be terminated. If the slave does not acknowledge the request the communication will be direct terminated with a failed read operation as a result.

There are a few disadvantages of reading from a slave in this fashion. First the delay in the loop will dramatically decrease the maximum data rate which can be achieved. Secondly, the received data would not be verified by looking at the acknowledge bits. Finally the probability of

a succesful read operation is minimal since the master will do a one shot request on a arbitrary moment, while the slave can be active with other tasks. Figure 2.10b represents the flowchart of this read operation.

To avoid these problems a new approach of reading from a slave is made. The new approach is derived from the current methodology and is based on a iterative process. However, there are significant improvements in the new approach. The most important enhancement will be the elimination of the delay inside the loop. The read procedure can be split up in a request part and a receive part. Starting the read operation the master will first try to get the attention of the slave by sending a request. This process will be repeated till there is a ACK received, or terminated if the amount of retries has reached the MAX_req value.

If there is a ACK received program flow will continue to the receive part. The master will clock in the amount of bytes (expresed in the request) once at a time. On each received byte there will be a ACK check to verify valid data is received. If not, the process repeats again as in the request part. When received all the bytes communication procedure will be terminated.

This new approach has many advantages in comparison to the current methodology. As already mentioned the datarate of the bus will dramatically increase due to the elimination of the delay in the loop. Furthermore the propability of receiving valid data is increased. Experiments and tests have led to defining appropriate values for MAX_req and MAX_recv. These values are set to 100.

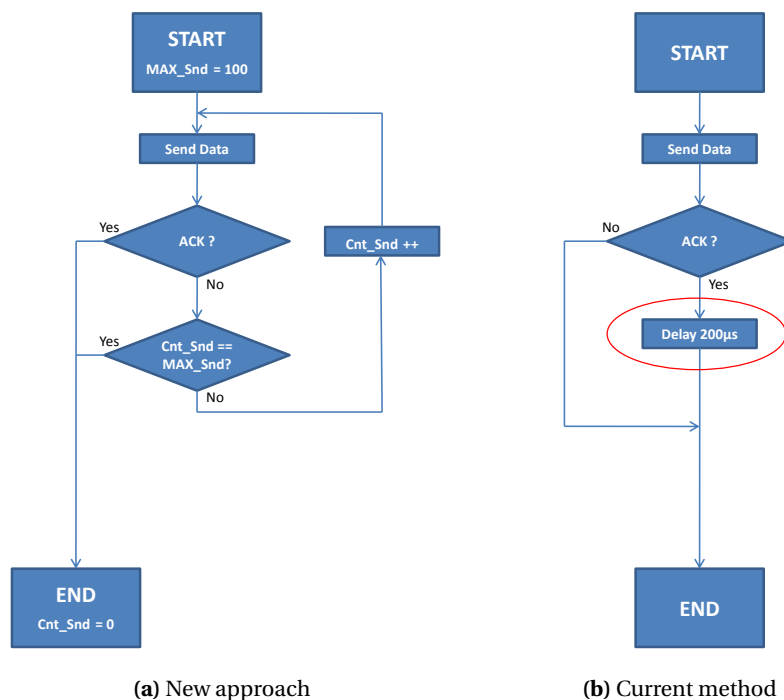


Figure 2.11: Procedure for writing to a slave device

To perform a write operation, a more simple procedure have to be walk through. The difference will be that there is no need for sending a request (with the containing command) to the slave. A data byte imediately can be transmitted to the slave. If there is a acknowledge bit received, the write operation was succesfull en communication will be terminated. If not, the communication will be directly terminated.

Also in this case there is a delay inside the loop which is not desirable. Another disadvantage is the one shot chance to transmit a byte and receive the acknowledge from slave. For the new

write approach a similar solution as in the read operation is used. Figure 2.11 shows the current and new approach to perform a write operation.

2.6.3 Reliability analysis

Interference

The current prototype uses a 5-pole ribbon cable to connect all the motorcontrollers and main-controller with each other. In this ribbon cable the power supply as well as the I2C bus are included. The cable is attached on each module along the robot with the use of connectors. From a digital point of view, ribbon cable is an ideal way to connect all the devices. However, from an analog point of view, these cables are problematic. Ribbon cables were highly efficient antennas, broadcasting essentially random signals across a wide band of the electromagnetic spectrum and therefore could interfere with the I2C bus.

Since the ribbon cable is attached on the outside of the modules along the robot, the DC motors are nearly close to the cable. When motors are turned on they could also interfere with the I2C bus. Because the I2C bus is a unbalanced bus with open collector drivers and weak pull-ups, it is quite sensitive to interference.

Wiring pattern of the bus lines

In general, the wiring must be so chosen that crosstalk and interference to/from the bus lines is minimized. The bus lines are most susceptible to crosstalk and interference at the HIGH level because of the relatively high impedance of the pull-up resistors. The longer the cable the more the I2C bus suffer from interference. To prevent as much as possible these disturbances a predefined wiring pattern is desirable.

Figure 2.12 shows the wiring pattern of the ribbon cable in the current prototype and the desired wiring pattern. Unfortunately, they do not agree. For the time being the current wiring pattern will be maintained since there needs to be done a redesign to agree with the desired wiring pattern, which is beyond the scope of this project.

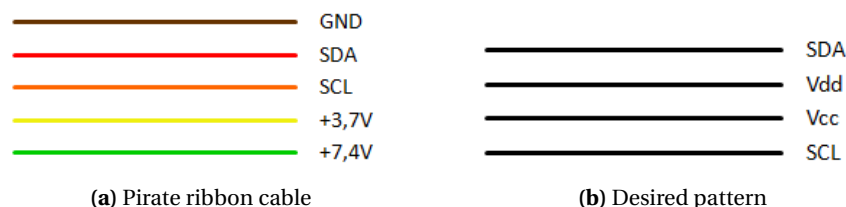


Figure 2.12: Wiring pattern of I2C signals

2.6.4 Conclusions

Earlier reports and preliminary testing brought some problems with the I2C bus to light. Especially the speed of the bus was disappointing. The analysis of the communication speed resulted in a measured clock frequency of 66,7KHz which deviates from the desirable 400KHz. To achieve a 400KHz clock the relevant registers in the LPC2148 controller are adapted, and the termination resistors are replaced. With these changes one can get the maximum performance out of the bus, which resulted in a measured clock speed of 343KHz.

Using the new approaches of reading and writing to the motorcontrollers has led to a more stable and smooth communication. No more software crashes (which are caused by motorcontrollers who are keeping the CLK line low) occurred during testing the communication.

2.7 Wireless communication

2.7.1 Introduction

The pirate robot is designed to explore the underground low pressure gas network autonomously. Once the robot is inside the pipe it starts inspection and will store the collected data to the onboard SD-card. After a while the stored data on the robot needs to be reported to the outside world. Since it is impossible to setup a continuous connection while robot is inspecting the underground network, docking stations are introduced.

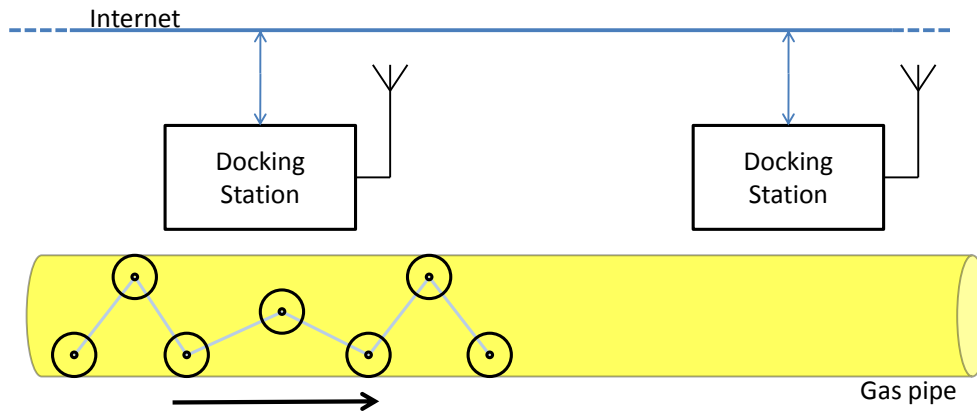


Figure 2.13: Representation of docking station usage

Docking stations

In this project a docking station is a station placed near a gas pipe which maintains the communication for data exchange. Because the robot is located inside the gas pipe a wireless connection is a suitable solution to accomplish communication. Only when the robot is approaching the docking station and is in the range of the wireless signal communication is possible. Pirate should be able to recognize the relevant docking station and will stop near the station to perform data exchange.

Besides the data exchange functionality the docking station will be used for recharging the batteries of the robot. Also for this feature conventional charging by a wired solution might be difficult. Inductive coupling and resonant inductive coupling can be used for wireless power transmission. Since this is beyond the scope of this project recharging will not be discussed in more detail in this report.

Every five year the entire low pressure gas network needs to be inspected by dutch law. This network is stretched out over 100.000km. Assuming a robot is able to inspect two kilometers of gaspipe a day (8cm/sec), 30 robots are required to inspect the entire network within the 5 year timeslot. The action range of one robot is expected to be 10km and therefore eventually 10.000 docking stations are needed.

2.7.2 Nordic nRF24L01 module

To accomplish a wireless link between the pirate robot and a docking station a suitable communication device is required. The custom-made mainboard on the robot is already prepared for this feature in the design trajectory and contains the Nordic nRF24L01 transceiver module.

The nRF24L01 is a single chip 2.4GHz transceiver with an embedded baseband protocol engine (Enhanced ShockBurst™), designed for ultra low power wireless applications. The nRF24L01 is designed for operation in the world wide ISM frequency band at 2.400 - 2.4835GHz. Below some key features of the module are enumerated:

- Worldwide 2.4GHz ISM band operation
- 126 RF channels
- 2Mbit/s air data rate
- Programmable output power: 0, -6, -12 or -18dBm
- Enhanced ShockBurst™
- Data and Control interface

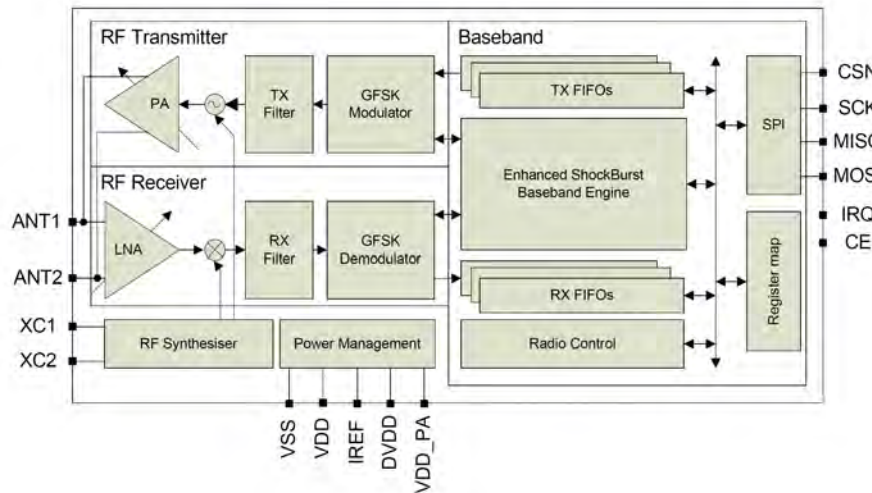


Figure 2.14: nRF24L01 Block diagram (Nordic Semiconductor, 2007)

Data and Control interface

The data and control interface gives access to all the features in the nRF24L01. The nRF24L01 is configured and operated through the SPI0 port on the maincontroller (see appendix G). Through this interface the register map is available. The register map contains all configuration registers in the nRF24L01 and is accessible in all operation modes of the chip. Furthermore the CE pin (Chip Enable) is used to enable the chip for receiving packets in RX mode or transmitting packets by toggle this pin in TX mode. The IRQ pin is the interrupt pin and will report if a data packet is received, a data packet is successfully sent, or the maximum retransmits is reached. Figure 2.14 shows the block diagram of the module including the pin descriptions.

Operation modes

The nRF24L01 can be configured in one of the following four main modes of operation:

- Power down mode
- Standby modes
- RX mode
- TX mode

In figure 2.15 the state transition diagram of the nRF24L01 is given. The module can switch between the different modes by controlling the PRIM_RX and PWR_UP bits in the CONFIG register and the state of the CE pin.

Power down mode

In power down mode nRF24L01 is disabled with minimal current consumption. In power down mode all the register values available from the SPI are maintained and the SPI can be activated. Power down mode is entered by setting the PWR_UP bit in the CONFIG register low.

Standby modes

By setting the PWR_UP bit in the CONFIG register to 1, the device enters standby-I mode. Standby-I mode is used to minimize average current consumption while maintaining short

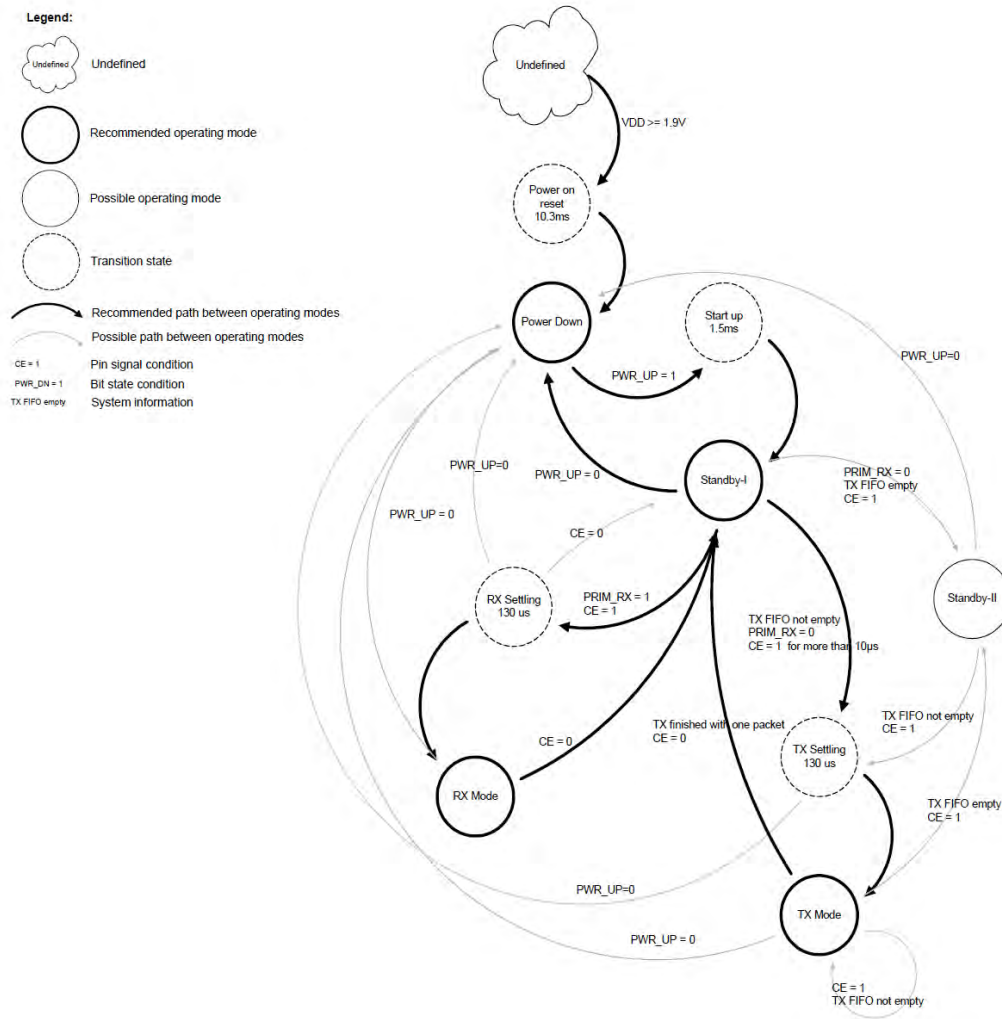


Figure 2.15: Radio control state diagram (Nordic Semiconductor, 2007)

start up times. In this mode part of the crystal oscillator is active. This is the mode the nRF24L01 returns to from TX or RX mode when CE is set low. In standby-II mode extra clock buffers are active compared to standby-I mode and much more current is used compared to standby-I mode. Standby-II occurs when CE is held high on a PTX device with empty TX FIFO. If a new packet is uploaded to the TX FIFO, the PLL starts and the packet is transmitted.

RX mode

The RX mode is an active mode where the nRF24L01 radio is a receiver. To enter this mode, the nRF24L01 must have the PWR_UP bit set high, PRIM_RX bit set high and the CE pin set high. In this mode the receiver demodulates the signals from the RF channel, constantly presenting the demodulated data to the baseband protocol engine. The baseband protocol engine constantly searches for a valid packet. If a valid packet is found (by a matching address and a valid CRC) the payload of the packet is presented in a vacant slot in the RX FIFO. If the RX FIFO is full, the received packet is discarded. The nRF24L01 remains in RX mode until the MCU configures it to standby-I mode or power down mode. If the automatic protocol features (Enhanced ShockBurst™) in the baseband protocol engine are enabled, the nRF24L01 can enter other modes in order to execute the protocol.

TX mode

The TX mode is an active mode where the nRF24L01 transmits a packet. To enter this mode, the nRF24L01 must have the PWR_UP bit set high, PRIM_RX bit set low, a payload in the TX FIFO and, a high pulse on the CE for more than 10 μ s. The nRF24L01 stays in TX mode until it finishes transmitting a current packet. If CE = 0 nRF24L01 returns to standby-I mode. If CE = 1, the next action is determined by the status of the TX FIFO. If the TX FIFO is not empty the nRF24L01 remains in TX mode, transmitting the next packet. If the TX FIFO is empty the nRF24L01 goes into standby-II mode.

Enhanced Shockburst

The embedded baseband protocol engine (Enhanced ShockBurst™) is based on packet communication and supports various modes from manual operation to advanced autonomous protocol operation. Internal FIFOs ensure a smooth data flow between the radio front end and the maincontroller. Enhanced Shockburst reduces system cost by handling all the high-speed link layer operations. The main features of Enhanced ShockBurst™ are:

- 1 to 32 bytes dynamic payload length
- Automatic packet handling
- Auto packet transaction handling
 - Auto Acknowledgement
 - Auto retransmit
- 6 data pipe MultiCeiver for 1:6 star networks

nRF24L01 library

In order to configure the internal registers of the nRF24L01 a library has been written which can be found in appendix F.3. This library consists of the files nRF24L01.c and nRF24L01.h.

2.7.3 Wireless COM-port redirector setup

The nRF24L01 is a transceiver which can be in TX or in RX mode. To accomplish a bidirectional data link between the pirate robot and a docking station, the transceivers should switch from TX into RX mode or visa versa. When switching from mode a settling time is involved which can be seen in figure 2.15. In order to test the performance of a bidirectional data link a wireless COM port redirector setup is used (see figure 2.16).

Used Hardware/Software

For the hardware there are two single mainboards used in the setup which are named 'Local' and 'Remote'. The local device is connected through a serial RS232 cable to the COM port of a PC. The remote device is not connected to any other hardware and is only provided from a power supply. On the PC there is a terminal program running which enables the user to send a character to the serial COM port.

Interrupts

The nRF24L01 has an active low maskable interrupt (IRQ) pin. The IRQ pin is activated when TX_DS IRQ, RX_DR IRQ or MAX_RT IRQ are set high by the state machine in the STATUS register. These interrupt sources have the following meaning:

- TX_DS: Succesfully transmits a packet in TX mode
- RX_DR: Received data in RX mode
- MAX_RT: Maximum retransmits of a packet reached in TX mode

The IRQ pin resets when MCU writes '1' to the IRQ source bit in the STATUS register. The IRQ mask in the CONFIG register is used to select the IRQ sources that are allowed to assert the IRQ pin. By setting one of the MASK bits high, the corresponding IRQ source is disabled. By default all IRQ sources are enabled.

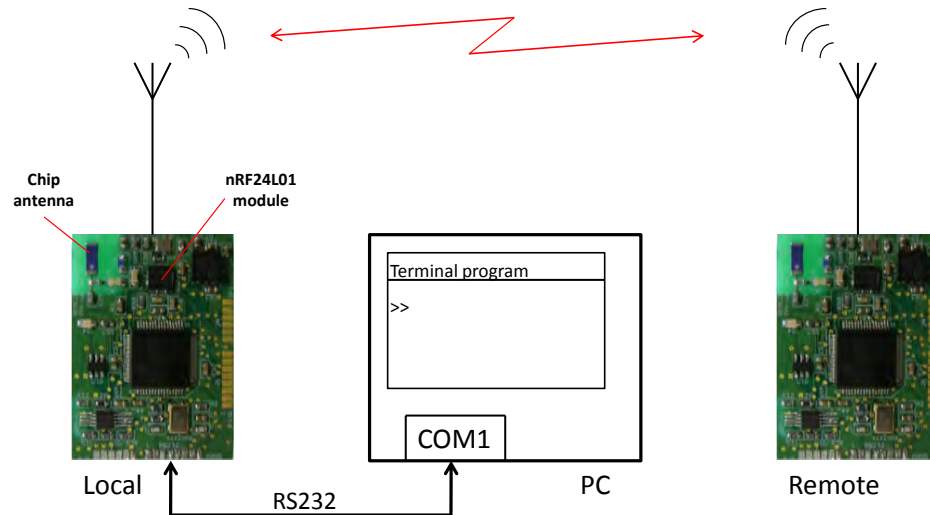


Figure 2.16: Wireless data redirector setup

Performance

The character send by the terminal program on the PC is received by the local device and passed through to the nRF24L01 which transmits the character into the air. The remote device receives this character and sends it back to the local device. Finally the local device redirects the received character to the PC trough the serial RS232 connection. In this fashion the local and remote device are operating in TX as well as RX mode. The programflow of the local- and remotedevice are shown in figure 2.17 and 2.18 respectively.

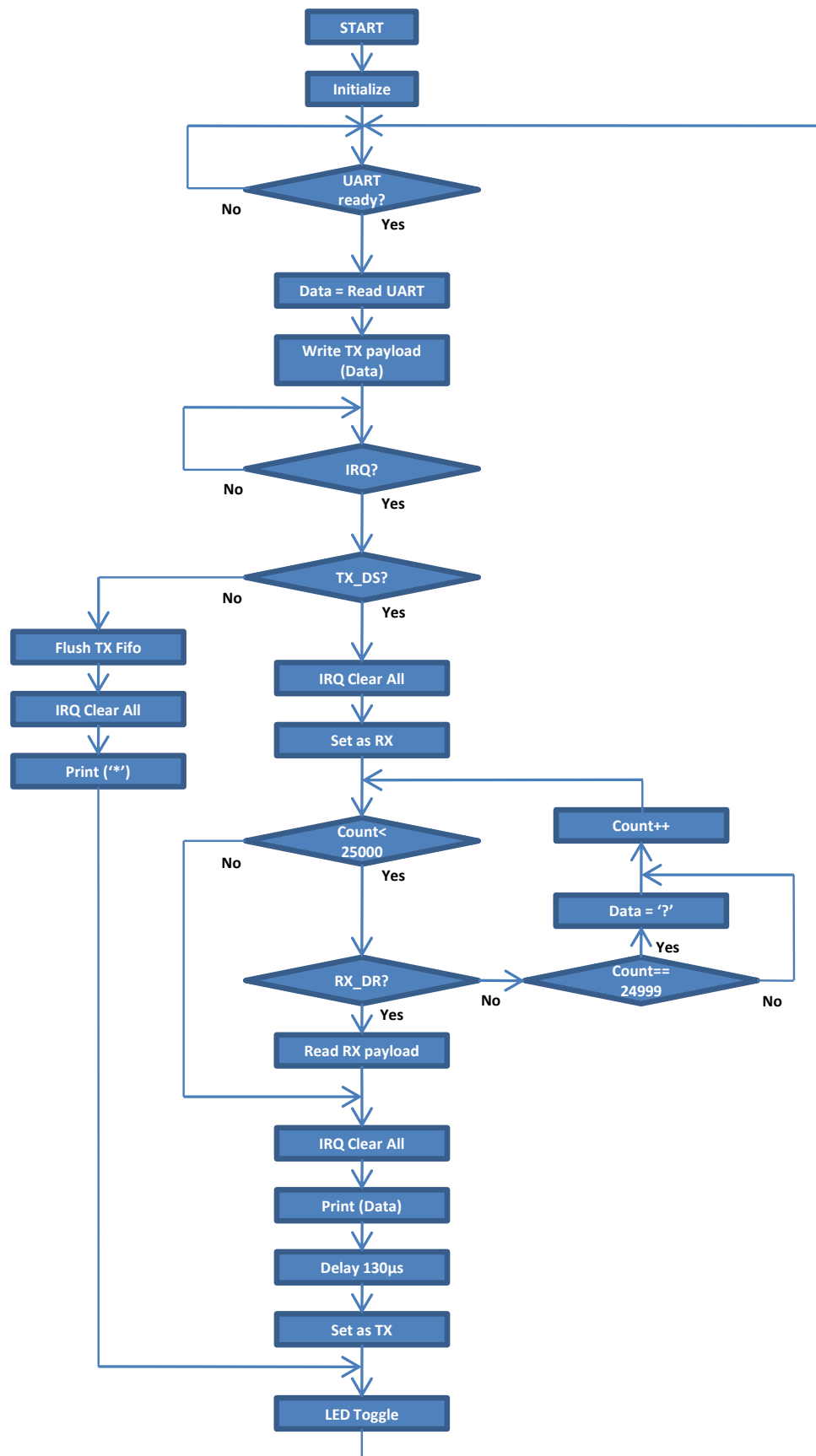


Figure 2.17: Programflow of local device

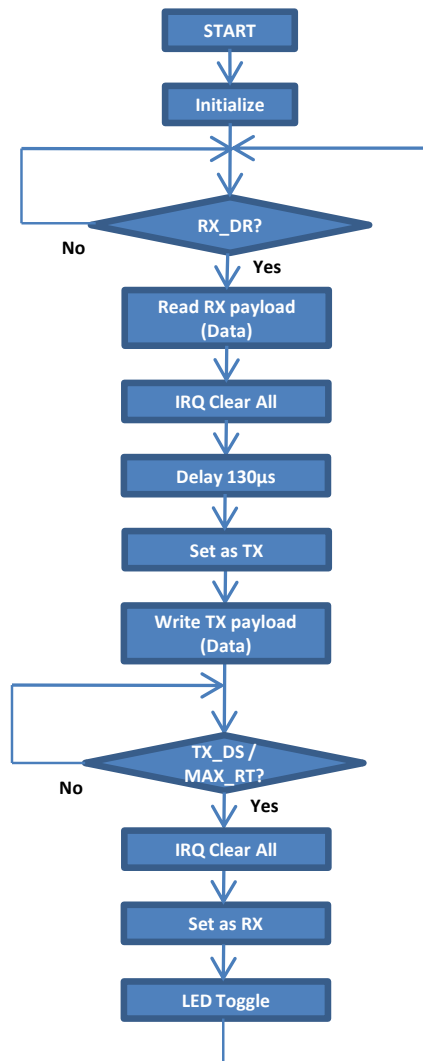


Figure 2.18: Programflow of remote device

Local

The first task the local device will perform when powering up is calling the `Initialize()` function. In this function all peripherals are initialized like PLL, I/O, UART, SPI. Also the nRF24L01 is initialized here in TX mode, one byte payload length, and enabled Enhanced Shockburst by calling the function `nrf24l01_initialize_debug(false, 1, true)`.

After initialization the program waits for available data from the UART. When a character is received from the PC it will store the character in the variable `Data`. Subsequently `Data` is transmitted through the SPI bus to the TX fifo of the nRF24L01 ready for transmitting into the air. When transmitting a packet in Enhanced Shockburst, `TX_DS` will not go active if the packet did not go through. Instead, `MAX_RT` IRQ will become active and the TX fifo is flushed (since the data is left in the TX FIFO), IRQ is cleared, and finally a '*' is printed to indicate failure on sending a character. If the `TX_DS` IRQ is asserted the packet did make it to the remote device and the IRQ is cleared by the function `nrf24l01_irq_clear_all()`.

If the packet did make it to the remote device, the local device is switched to RX mode for retrieving the character back from the remote device. The program passes a iterated loop where the `RX_DR` IRQ is polled to determine if the packet is received. If the `Count` variable equals 24999 and the packet is not received, the question mark is assigned to `Data`. However, if the `RX_DR`

IRQ is asserted, program exits the loop and the RX payload is read, IRQ is cleared and the received data is printed. After that there is a delay of $130\mu\text{s}$ inserted to give the remote device the opportunity to switch back to RX mode (RX settling time).

Finally the program toggles a LED to indicate data transfer, and returns to the beginning of the program where the UART is polled for receiving a new character.

Remote

The remote device starts with the same `Initialize()` function as in the local device, except for the nRF24L01 initialization. The nRF24L01 on the remote device is initialized in RX mode, one byte payload length, and enabled Enhanced Shockburst by calling the function `nrf24l01_initialize_debug(true, 1, true)`.

After initialization the program waits for a packet to arrive in the RX fifo of the nRF24L01. If the RX_DR IRQ is asserted the payload is read from the RX fifo and stored in the `Data` variable. Subsequently the IRQ is cleared and a delay of 130μ is inserted to give the local device the opportunity to switch to RX mode (RX settling time). After that, the remote device is switched to TX mode and the value of data is transmitted through SPI to the TX fifo ready to transmit the received character back to the local device.

Similar to the local file, both the TX_DS IRQ and MAX_RT IRQ have to be observed to ensure that the program do not get locked up. Whether the packet was sent correctly or not is ignored (TX_DS or MAX_RT interrupt active, respectively), and interrupts are cleared. Finally the program toggles a LED to indicate data transfer and goes back to the beginning of the loop waiting for a new character received through the RF link.

2.7.4 Conclusions

In figure 2.19 the window of Terminate (terminal program) shows the program working. The local and remote devices are approximately 2 meters away from each other in this setup. Only one of the packets actually did not make it through. This packet can be seen on the 3rd line and the 8th character as an asterisk. The fact that it is an asterisk indicates that the MAX_RT interrupt was asserted.

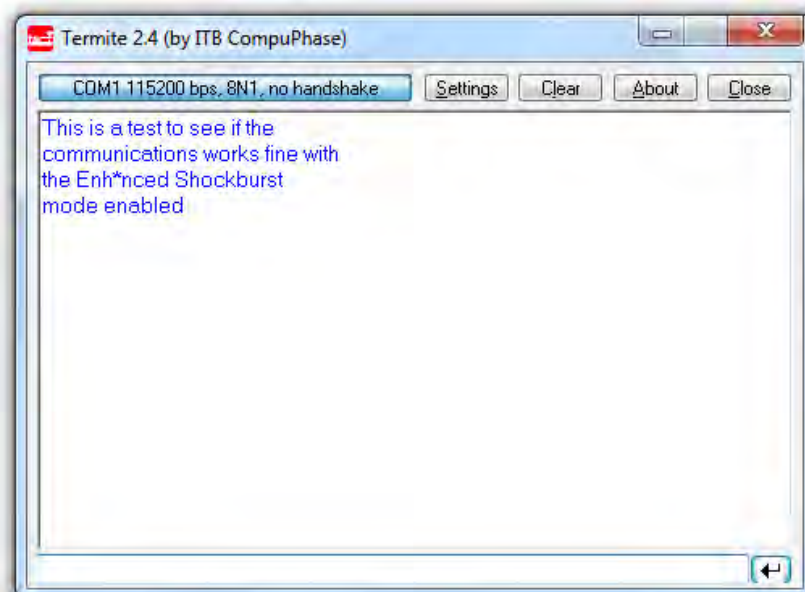


Figure 2.19: Output window from terminal program: Terminate (Compu Phase, 2009)

2.8 Design requirements

The goal of this project was to develop a wireless solution to perform state feedback from the robot to the pc. Furthermore this link can be used to transmit parameters to pirate. To develop such a system requirements are determined based on the analysis which is done in this chapter.

2.8.1 Full state feedback

The variables which are involved in the state feedback in this project are divided in two parts. The first set of variables (states) are normally used in closed-loop control systems and contain:

- Angular velocities of the driven wheels 2 and 7
- Angular displacement of the joints 2, 3, 4, 6, and 7
- Torque applied on joints 2, 3, 6, and 7

Furthermore the system must be able to extend the state feedback with additional variables like: Timestamp, Temperature, Status of motorcontrollers, and orientation of the robot.

2.8.2 Control of pirate

In order to control pirate the human operator needs to be able to give commands to the robot. These control commands can be setpoints for the PID controllers running on the motorcontrollers, enable or disable state feedback, and set parameters.

2.8.3 Docking station

A docking station is needed for the communication between the pirate robot and the pc. This docking station is used in the future for data exchange mainly. In this assignment the docking station is only used to pass the state data to the pc. The docking station must be able to connect to the pc with a commonly used interface which requires minimal software overhead.

2.8.4 Wireless requirements

The pirate robot must be able to communicate with the docking station in a range of several meters. Also when the robot is located in a pipe communication must be possible. To check the validity of the transmitted data back and forth a data integrity control methodology has to be implemented in the wireless protocol. When a datapacket has not reached the receiver correctly, the system must be able to do a retransmission of the data as long as the updatefrequency is guaranteed.

2.8.5 Representation in simulation

To give the human operator information about the state of the robot, a representation of pirate must be made from the incoming state data. A simulation can be done to perform a realtime animation of the pirate robot. This representation can be used for development of an autonomous inspection robot.

2.8.6 Update frequency

The update frequency is the frequency on which pirate transmits state data to the outside world. The required update frequency depends on the speed pirate operates and the human operator can handle. The human eye can handle a maximum of 25fps whereby motions look fluid. If the frame rate is higher than this 25fps the human eye does not see any difference. Therefore, the realtime simulation should be run on at least 25Hz.

The maximum driving speed of the robot is determined to be 80mm/s (Dertien, 2006). When sampling on 25Hz this would result in a 3,2mm movement every state update. To ensure a more fluid looking motion in the simulation of pirate, the update frequency is determined to be 50Hz.

3 Design and implementation

In this chapter all the discussed subjects from chapter 2 will come together to construct the final design of this assignment. From this design the implementation is treated as well.

3.1 Introduction

In order to acquire state feedback from the robot on the one hand and control the robot on the other hand a bi-directional connection between pirate and PC is required. The signalpath can be divided into three main parts which can be seen in figure 3.1. In this chapter the entire signalpath between pirate and pc is described.

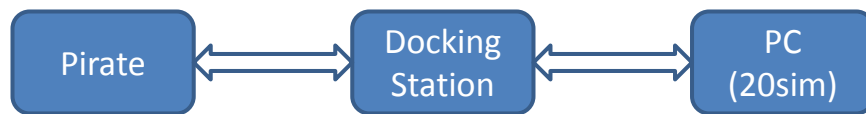


Figure 3.1: Signalpath between pirate and PC

From the design requirements pirate must be able to communicate wireless with the outside world and therefore the signalpath between pirate and the docking station should be a wireless solution. From a compatibility point of view the signalpath from the docking station to the PC is performed using a USB cable.

3.2 Pirate data acquisition

On the robot the maincontroller is responsible for the overall control and gathering internal state data used for state feedback. This state data needs to be transmitted wireless to the dockingstation. To accomplish this, the functionality of the maincontroller can be divided into the following sequentially executed processes:

- Compose statevector
- Send statevector
- Send beacon
- Receive data
- Process received data

3.2.1 Compose statevector

The first task the maincontroller will perform is composing a statevector of the internal state data. To do so, first one has to determine which variables have to be included in the statevector. After that, the state data needs to be acquired from the different motorcontrollers and the maincontroller itself. In table 3.1 an overview of all the included variables in the statevector is given.

The most state variables in table 3.1 are coming from the motorcontrollers (modules) and have to be acquired using the functions given in appendix E.1. Although the functions can be called using the commandline functionality, here the functions are directly called from within the mainprogram. These states include the angles of joints, torque applied to joints (bendmodules), angular velocities of actuated wheels, the current each motorcontroller dissipates by its connected motors and the status of the motorcontrollers. The remaining states are coming from the maincontroller which includes 3D acceleration data (to determine orientation of the robot), time and temperature.

State variable	Mainboard	20sim
X Acceleration	Statevector[0]	y[1]
Y Acceleration	Statevector[1]	y[2]
Z Acceleration	Statevector[2]	y[3]
Joint2 Angle	Statevector[3]	y[4]
Joint3 Angle	Statevector[4]	y[5]
Joint4 Angle	Statevector[5]	y[6]
Joint6 Angle	Statevector[6]	y[7]
Joint7 Angle	Statevector[7]	y[8]
Joint2 Torque	Statevector[8]	y[9]
Joint3 Torque (locked)	Statevector[9]	y[10]
Joint6 Torque (locked)	Statevector[10]	y[11]
Joint7 Torque	Statevector[11]	y[12]
Angular vel. wheel2	Statevector[12]	y[13]
Rotation angle	Statevector[13]	y[14]
Angular vel. wheel7	Statevector[14]	y[15]
Current Module A(65)	Statevector[15]	y[16]
Current Module B(66)	Statevector[16]	y[17]
Current Module C(67)	Statevector[17]	y[18]
Current Module D(68)	Statevector[18]	y[19]
Current Module E(69)	Statevector[19]	y[20]
Status Module A(65)	Statevector[20]	y[21]
Status Module B(66)	Statevector[21]	y[22]
Status Module C(67)	Statevector[22]	y[23]
Status Module D(68)	Statevector[23]	y[24]
Status Module E(69)	Statevector[24]	y[25]
Time Hours	Statevector[25]	y[26]
Time Minutes	Statevector[26]	y[27]
Time Seconds	Statevector[27]	y[28]
Time Milliseconds	Statevector[28]	y[29]
Temperature	Statevector[29]	y[30]

Table 3.1: Statevector

All these states together deliver a statevector of 30 variables. Since the most state variables contain a 16bit value the statevector is declared as a `short int` array of size 30 and is named `Statevector`:

```
short int Statevector[30];
```

3.2.2 Send statevector

The next step is to transmit this composed statevector wireless to the docking station. The before mentioned Nordic tranceiver modules from chapter two are used here to perform this task.

The nRF24L01 module on the mainboard is standard configured in RX mode. When one wants to transmit data the module needs to be reconfigured into TX mode. Once the command is given to switch to TX mode a delay of 130 μ s settling time is necessary to ensure the device is in TX mode (see figure 2.15). The Enhanced ShockBurst™ feature is used to achieve a stable communication.

Enhanced ShockBurst packet format

The Enhanced ShockBurst™ packet contains a preamble field, address field, packet control field, payload field and a CRC field. Figure 3.2 shows the packet format with MSB to the left.

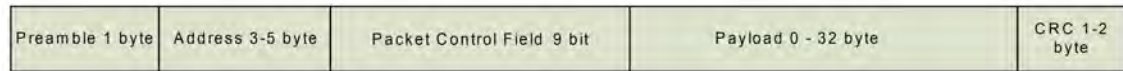


Figure 3.2: An Enhanced ShockBurst™ packet with payload (0-32 bytes) (Nordic Semiconductor, 2007)

Packet handling

The payload length can be adjusted from 0 to 32 bytes. Since the statevector consists of an array of 30 `short int` variables (which are 60 bytes), the statevector can not be sent at once. Therefore the statevector is divided into two parts. The first part contains the first 15 states, and the second part contains the states 15 till 29. In figure 3.3 the content of the two datapackets is given.

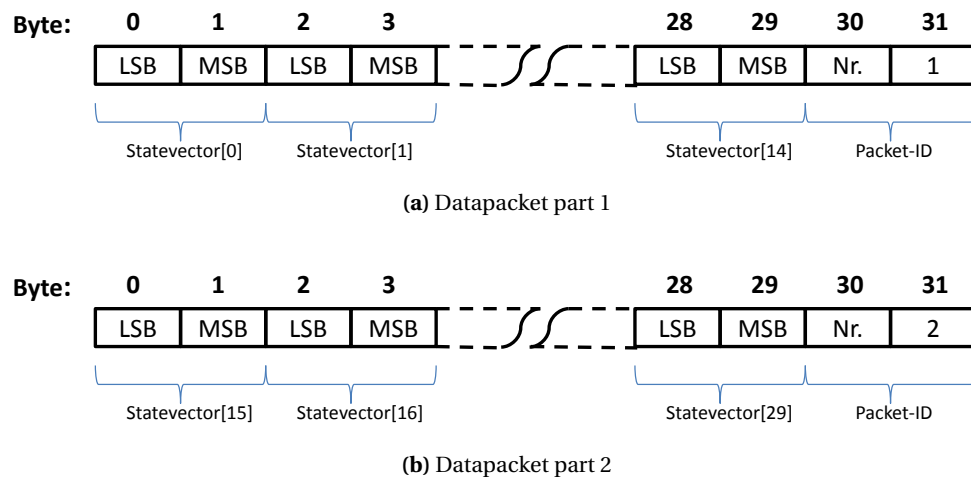


Figure 3.3: Statevector divided into two datapackets

The payload length is adjusted to 32 bytes and can therefore contain one datapacket. The remaining two bytes are used for packet handling. Byte 30 is used for the relative packetnumber and byte 31 for the partnumber of that particular packetnumber. In this fashion the docking station can determine if two received packets belong together by comparing the packetnumbers. The partnumbers are used to construct one single statevector from two data packets with the same packetnumber.

3.2.3 Send beacon

After the statevector has been sent a beacon packet is transmitted into the air. The beacon packet can be recognized by the first two bytes which contain two hash-keys. This beacon packet can be used to transmit additional information to the outside world like pirate ID, current status, or an SOS signal when pirate gets stuck and is therefore in an emergency situation for example. This features are not implemented yet.

The second function of the beacon packet is to indicate the docking station it is allowed for sending data back to pirate for a fixed timeslot. Figure 3.4 shows the content of the beacon packet.

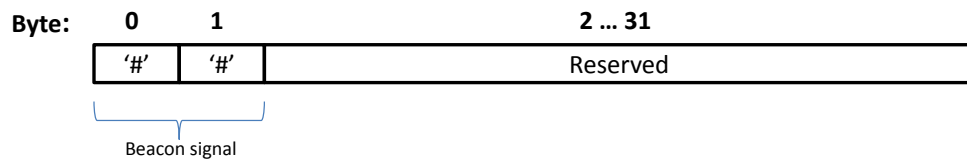


Figure 3.4: Beacon packet

3.2.4 Receive data

In order to control the robot the user must be able to send data to pirate from the pc. Directly after the beacon packet was sent pirate will switch to RX mode for a fixed timeslot to receive data coming from the docking station. When the docking station received the beacon packet and there is data available for sending it immediately starts transmitting these data to pirate. The sending time of the docking station may not exceed the fixed timeslot defined by the robot.

3.2.5 Process received data

The last task the maincontroller performs is processing the incoming data from the docking station. This can be for example setpoints for the pid controllers running on the motorcontrollers and therefore have to be passed to the relevant motorcontroller through the I2C-bus.

3.2.6 Timing issues

In this subsection the time each task or process needs to be executed on the maincontroller is described. A sequence diagram of all these processes is given in figure 3.5.

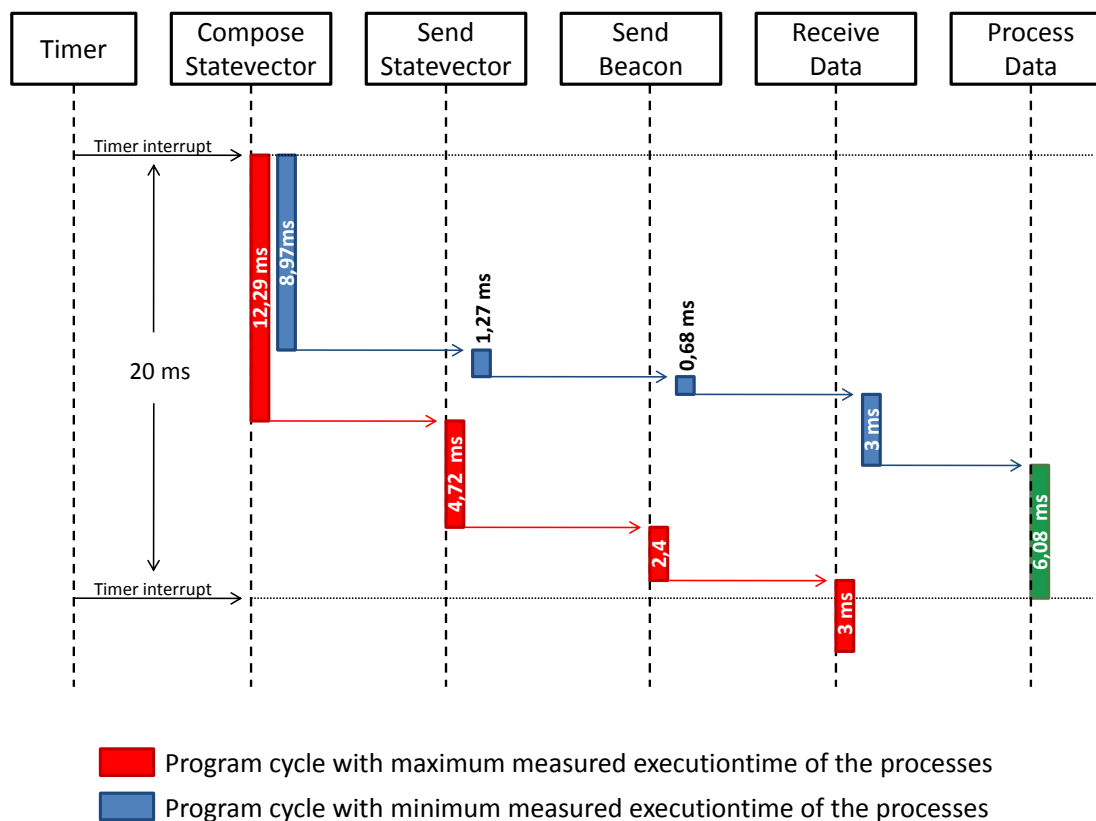


Figure 3.5: Sequence diagram of maincontroller

From the design requirements it follows that pirate must be able to perform state feedback to the PC with a update frequency of 50Hz. To realize this a timer on the maincontroller is used to generate an interrupt every 20ms. When an interrupt occurred, the system will start executing the processes in a sequential way.

Process executiontime variation

From figure 3.5 there can be two trajectories distinguished. The red trajectory represents the worst case scenario regarding the executing time of the individual processes. The blue trajectory represents the trajectory whereby the processes are executed with the minimal execution time.

The executiontime of the `Compose Statevector` process can be between 8,97ms and 12,29ms. This is caused due to the unpredictable responsetime of the motorcontrollers through the I2C-bus. The executiontime of the `Send Statevector` process can be between 1,27ms and 4,72ms which is caused by the possible retransmits the maincontroller performs with the Enhanced Shockburst™ feature enabled. The maximum allowed retransmits is adjusted to three times. After three failed attempts to transmit the statevector the program continuous with the `Send Beacon` process.

The executiontime of the `Send Beacon` process can be between 0,68ms and 2,4ms. This is caused by the same reason as in the `Send Statevector` process. In the maincontroller there is a fixed timeslot reserved of 3ms for the `Receive Data` process and therefore the executiontime is constant. The last task to perform is the `Process Data` process and is only executed if there is time left till the next interrupt will occur.

The red trajectory in figure 3.5 shows the maincontroller is still busy with the `Receive Data` process when the next timer interrupt occurs and therefore is not able to execute the `Process Data` process. The blue trajectory shows there is 6,08ms time left for the `Process Data` process after finished the other processes.

Cycle time

The cycle time is the sum of the executiontime of all the processes except for the `Process Data` process:

- Cycle time red trajectory: >20ms
- Cycle time blue trajectory: 8,97ms + 1,27ms + 0,68ms + 3ms = 13,92ms

Because the cycle time of the red trajectory exceeds the 20ms there is no time left for processing the received data. The cycle time of the blue trajectory is 13,92ms which enables the maincontroller to process received data by executing the `Process Data` process for 6,08ms.

Enable/Disable state feedback

Since the robot uses a commandline interface it is possible to enable and disable the state feedback feature by sending the commands 'D1' or 'd0' respectively to the robot. When state feedback is disabled the `Compose Statevector` and `Send Statevector` processes are not executed and therefore the program jumps immediately to the `Send Beacon` process. When the robot is in this IDLE state only the `Send Beacon` and `Receive Data` processes are executed which will decrease the cycle time as a result.

- Cycle time red trajectory (IDLE): 2,4ms + 3ms = 5,4ms
- Cycle time blue trajectory (IDLE): 0,68ms + 3ms = 3,68ms

When the state feedback is disabled and the robot is therefore in the IDLE state the cycle time is decreased dramatically. In the red trajectory case there is 20ms-5,4ms = 14,6ms left for processing received data. The cycle time of the blue trajectory is 3,68ms which enables the main-

controller to process received data by executing the `Process Data` process for 20ms-3,68ms = 16,32ms. In figure 3.6 the cycle time is depicted whereby:

- T1: Compose Statevector
- T2: Send Statevector + Send Beacon + Receive Data
- T3: Process Data

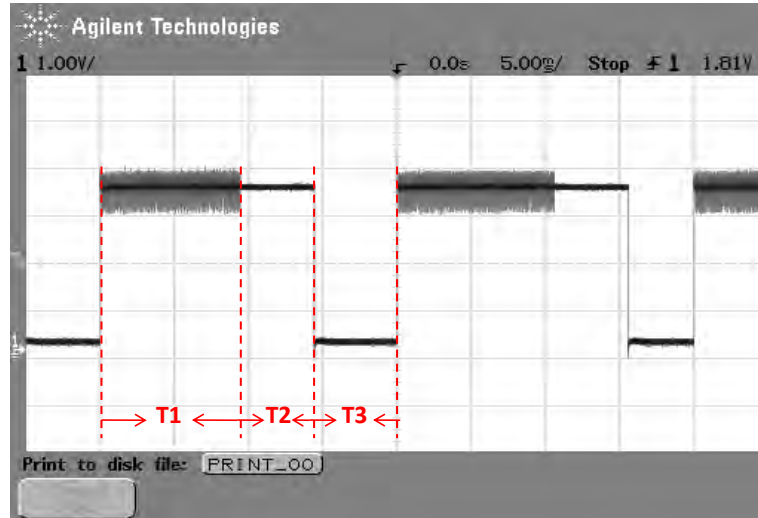


Figure 3.6: Cycle time with state feedback enabled

3.3 Docking station

In this section the builded prototype of a docking station is described which will provide a bridge between the pirate robot and the PC (see figure 3.7). Pirate will send state data wireless to the docking station, where the docking station passed this data to the PC through a wired USB connection. However, commands coming from the PC can be received by the docking station and transmitted wireless to the robot.



Figure 3.7: Prototype docking station

3.3.1 Electronics

In figure 3.8 all the hardware components present in the docking station are shown. From a reuseability point of view the mainboard is used as embedded system in the docking station. The mainboard provides the docking station of all the functionalities it needs. There is a mini USB connector onboard of the mainboard which is used to ensure the USB data link to the PC.

Sparkfun tranceiver module

For the wireless communication there is ofcourse the onboard nRF24L01 module present on the mainboard. This module is not used since the module is located inside the docking station and uses a chip antenna which limits the maximum available wireless range as a result. Therefore, a module from sparkfun is used to maintain the wireless communication (Appendix G.3). On this module there is a nRF24L01 device present and can be accessed by the breakout SPI pins on the board. Thes SPI pins are connected to the SPI0 port on the mainboard. The advantage of using this sparkfun module is that it has a RP-SMA antenna connector which is brought out of the docking station and therefore increases the wireless range.

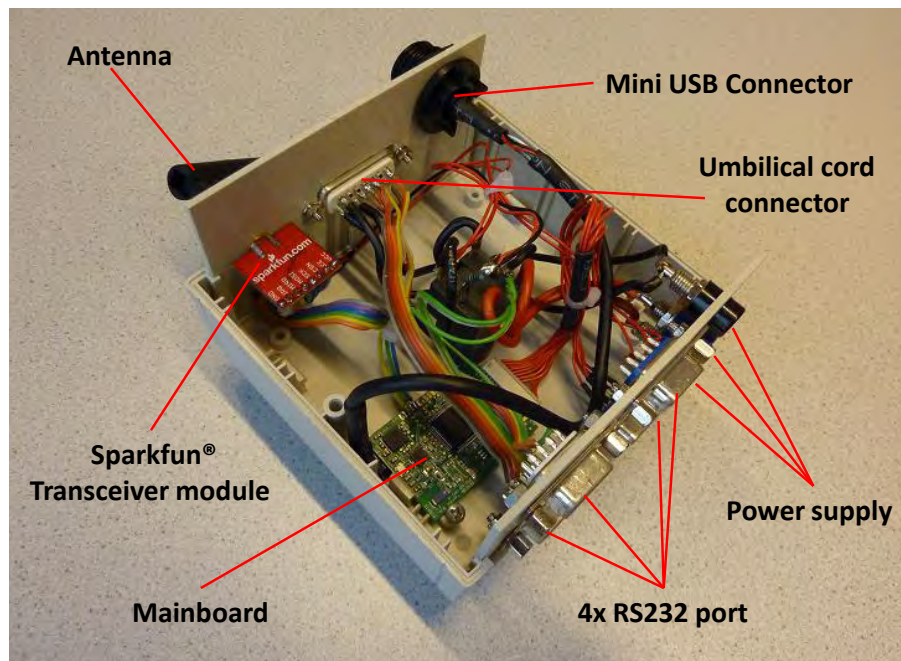


Figure 3.8: Inside the docking station

RS232 ports

On the docking station there are four RS232 ports present. Two of those ports are connected to the mainboard located in the dockingstation and provide a programming interface for the LPC2148 and a communication port. The other two RS232 ports are redirected to the umbilical cord which is currently used to control the robot. With these two ports it is possible to program new firmware into the robot and use the other port as debug/communication port.

Power supply

Finally there are three connectors present for the power supply which contain +7,4V(Red), +3,7(blue) and ground(black). These power lines are connected to the umbilical cord connector after they passed the emergency stop. Currently the robot uses the umbilical cord for the power supply. When robot is operational underground this is not possible anymore and therefore batteries should be attached and used on the robot. Furthermore the powerlines are used for powering the sparkfun module and the mainboard.

3.3.2 Docking station firmware

The firmware running on the docking station has to perform two tasks. The first task is to setup the USB communication between the PC and docking station. The second task is to setup the wireless communication using the sparkfun module before mentioned. The USB communication channel and the wireless communication channel are connected to each other in such a way the docking station functions like a service-hatch.

USB serial port

On the docking station the LPCUSB, an USB device driver for LPC microcontrollers (B. Sikken, 2006) is used to perform the USB communication with the PC. With this device driver the docking station will act as an USB-serial device. When connecting the docking station to the PC by means of the USB cable, PC will recognize the docking station as a virtual comport and will ask the user for the required drivers to install the device. In appendix D an installation manual is given to ensure a succesful installation on a windows XP machine.

Two fifo buffers are used on the docking station to avoid data packets get lost. There is a RX and a TX buffer which have a configurable size. At the moment the buffersize is set to 1024 bytes which seems to be large enough during testing. Since the USB driver is interrupt driven, data coming from the pc will be stored in the RX fifo buffer in the interrupt service routine. For transmitting data to the pc one can just put data into the TX fifo and the USB driver handles the further communication.

Wireless data handling

As already mentioned pirate will alternately transmit the beacon packet and data packets if state feedback is enabled. The docking station must be able to filter the beacon packet out of the data packets. This is done by examining the first two bytes of the received packet. If the two bytes contains the hash-keys a beacon packet is received and therefore the docking station is able to transmit data from the RX fifo buffer to pirate for a 3ms timeslot.

If the received packet is not a beacon packet it has to be a data packet. If the received data packet is the first part of the statevector (part 1) the packetnumber and partnumber are stored in memory. The next data packet which is received will be compared to the saved datapacket. If the packetnumbers are the same and the packet is the second part of the statevector (part 2) the packets belong together and are merged to one single statevector as it was before transmitting by pirate. Directly after the statevector is composed it is send to the TX fifo where the USB driver sends this statevector to the PC as a long string with the state variables separated by a tab.

3.4 World modeling

In this section the world modeling based on state feedback is described. For the world modeling in this project the 20sim modeling and simulation environment is used. World modeling is used to give the user feedback of the actual state of the robot and can provide a helping hand for further development of an autonomous inspection robot.

3.4.1 DLL

For the world modeling there is a connection required between the 20sim environment and the USB port which is connected to the docking station. This connection is implemented through a multithreaded DLL which enables 20sim to read and write to the virtual comport created by the docking station. From earlier research the use of a multithreaded DLL has led to a smooth communication between 20sim and other applications or hardware communication ports. The DLL contains a reader and writer thread which performs read and write functionality to a serial port. In this case the virtual comport is the port the DLL will access (see figure 3.9).

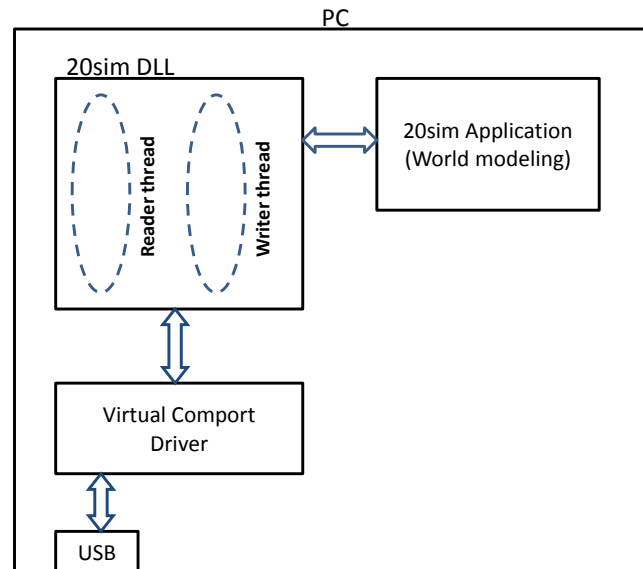


Figure 3.9: Multithreaded DLL for 20sim

The DLL is written in C++ and compiled with Visual C++ compiler. In appendix F.1 the source code of this DLL can be found. In the DLL there are two functions written which 20sim will use to receive state data on the one hand and send parameters to pirate on the other hand. These two functions are `GetData()` and `SetParameters()` respectively. With the `SetParameters()` function parameters of the motorcontrollers can be uploaded to the robot. These parameters contain:

- PID gains
- PID mode
- PWM output limit
- Currentlimit H-bridge
- H-bridge Enable/Disable
- Integral Antiwindup

In figure 3.10 can be seen how easy these parameters can be adapted. The parameters are uploaded before the simulation is started.

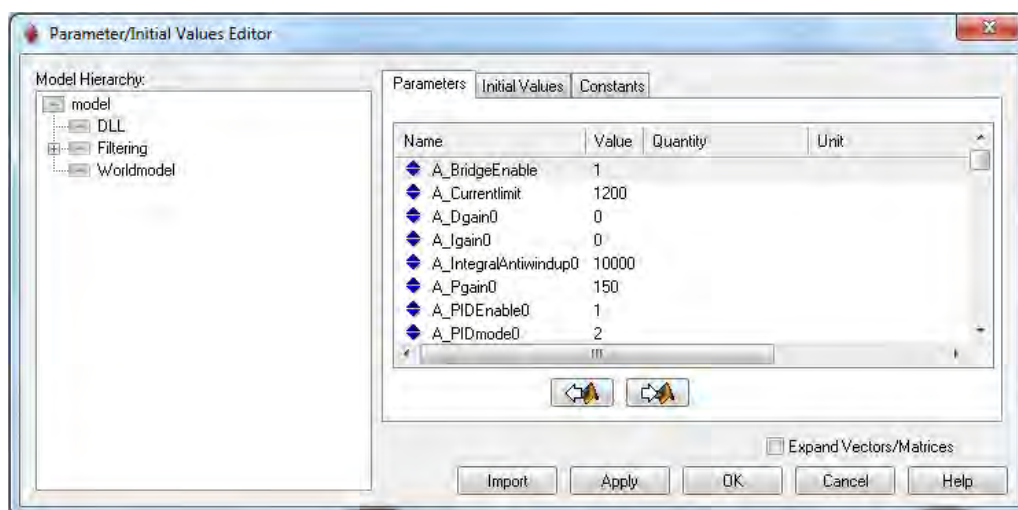


Figure 3.10: Initial parameters for pirate

3.4.2 Filtering

In figure 3.11 an overview is given of the world modeling processes. These processes are the already mentioned DLL for the communication to the outside world, a filtering process and finally a worldmodel. In this section the filtering process is described.

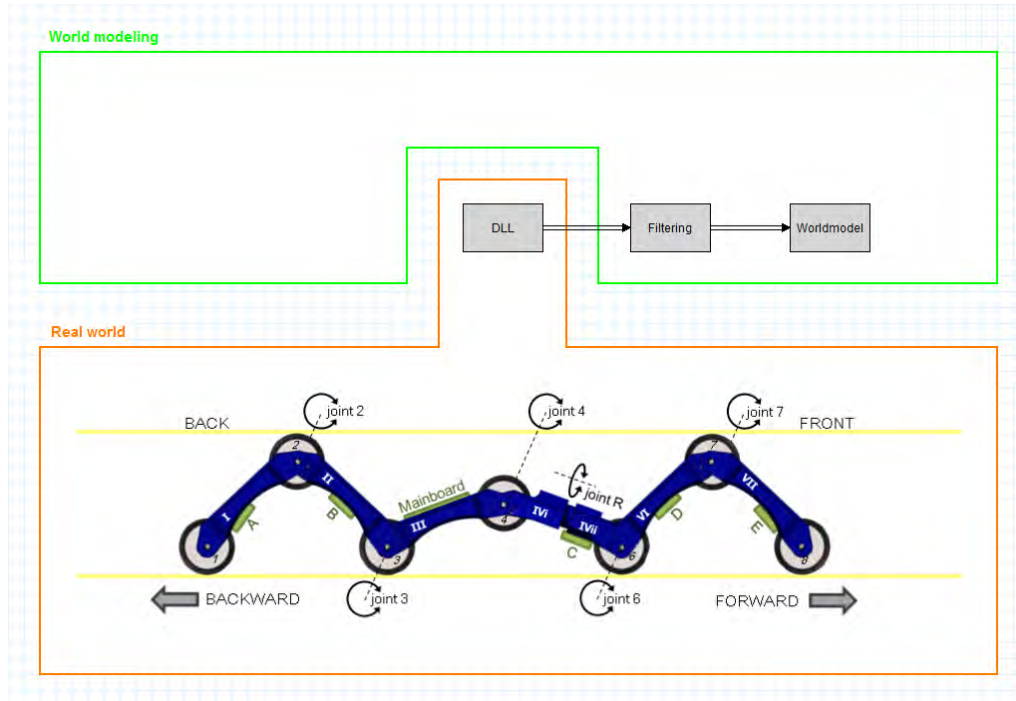


Figure 3.11: World modeling in 20sim environment

The filtering process is inserted to filter out the noise present on the state data. Each state can be individually filtered as shown in figure 3.12. A simple lowpass filter of 5Hz has led to convenient results on all states. The status, time, and temperature states are not filtered.

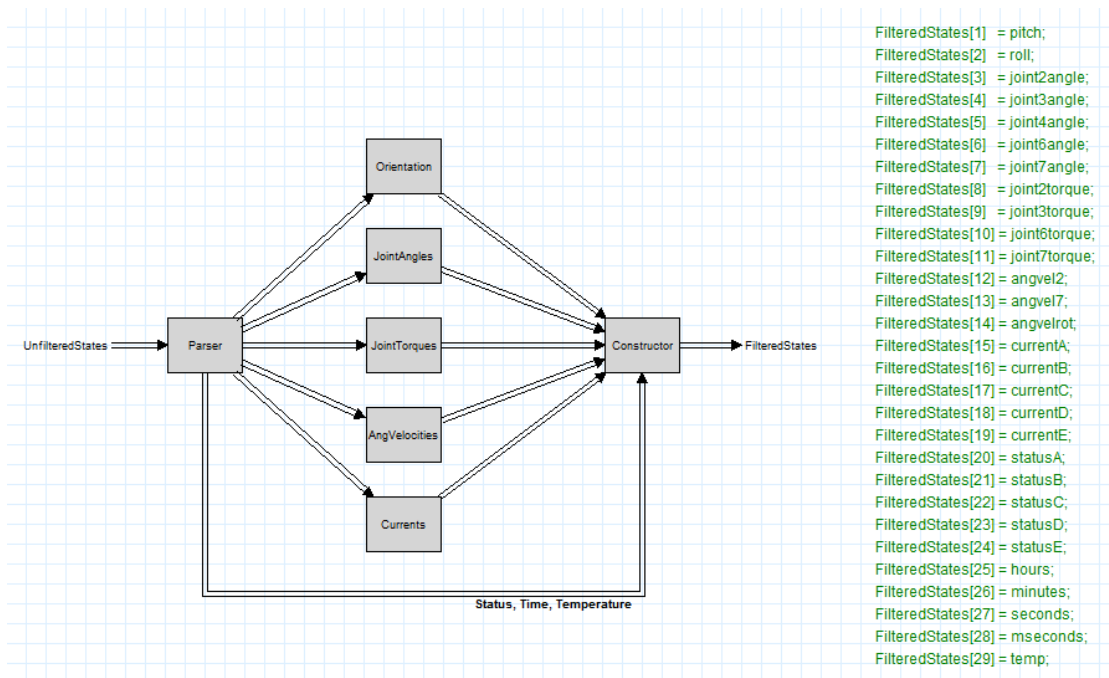


Figure 3.12: Filtering the incoming states from pirate

3.4.3 World model

Introduction

In the world model the filtered state data is used to construct the pirate robot in the 3D Animation editor (see figure 3.13). Doing so, a real time animation of pirate can be realized. This enables the user to get a visualisation of the robot when it is inside a non-transparent pipe for testing purposes in the laboratory of the control engineering group.

Since pirate has to operate fully autonomous underground it is necessary for the robot to have information about the environment it is located at that moment. The pipe diameter seems to be important data and is therefore estimated in the world model. Furthermore, there are all kinds of pipe transitions, bends and joints in the underground gasnetwork and therefore pirate needs information about these to get through them. This information is gathered by doing an environmental feature estimation in the world model.

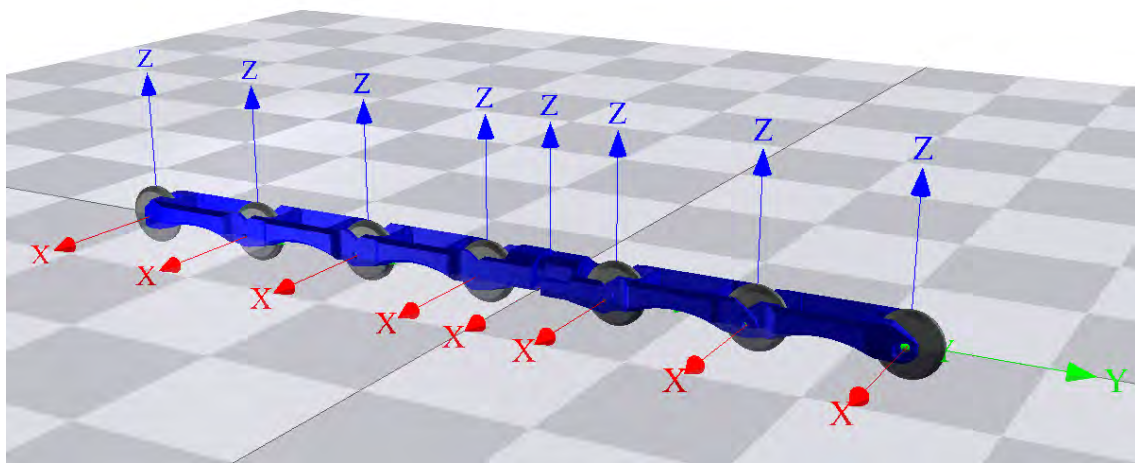


Figure 3.13: Pirate constructed from Solidworks-files in 3D Animation editor

Forward kinematic animation

The essential concept of forward kinematic animation is that the positions of particular parts of the model at a specified time are calculated from the position and orientation of the object, together with any information on the joints of an articulated model.

The pirate model is constructed in the 3D Animation editor of 20sim using the existing solidworks files of the different parts (see figure 3.13). On each joint there is a reference frame attached which are interconnected by the module parts. The orientation and position of these reference frames can be adapted by the world model. Figure 3.14 shows the tree structure of these reference frames.

Orientation of the model

The main frame of the model is located at the middle of the rotation module. The orientation and position of all other reference frames are determined using forward kinematics with respect to this main frame. The orientation of the entire robot in this animation model is determined by the onboard 3D acceleration sensor. This orientation is applied to the main frame. A conversion from acceleration data to euler angles is required to obtain the 3D orientation. In appendix A.1 this conversion is described.

Color management

In the animation editor the present objects can be represented in every possible color. This feature is used to give the user feedback in a very intuitive way. Initially, the module parts of

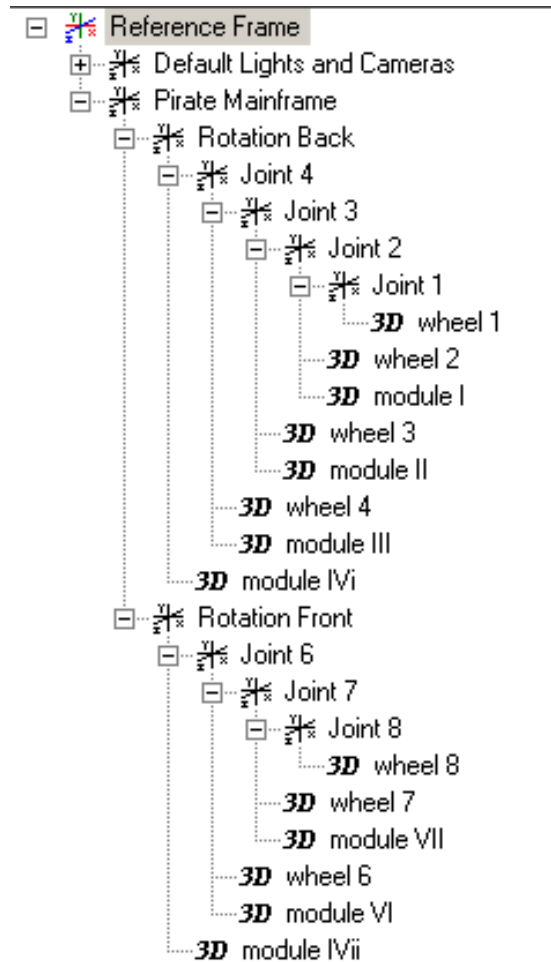


Figure 3.14: Reference frames tree structure

the model are given a blue color. When realtime simulation starts parameters are send to pirate and state feedback is retrieved. The status byte of each module is correlated to the color of the relevant module. Green indicates an online active module where red indicates an exception occurrence.

Pipe diameter estimation

In this section the pipe diameter estimation process is described. Since the robot clamps itself in a pipe at the back and the front by the bending modules, the pipe diameter estimation process is done also separately for the back and front. In the animation editor there are two pipes fitted on the wheels 1, 2, 3 and 6, 7, 8 respectively. This is shown in figure 3.15.

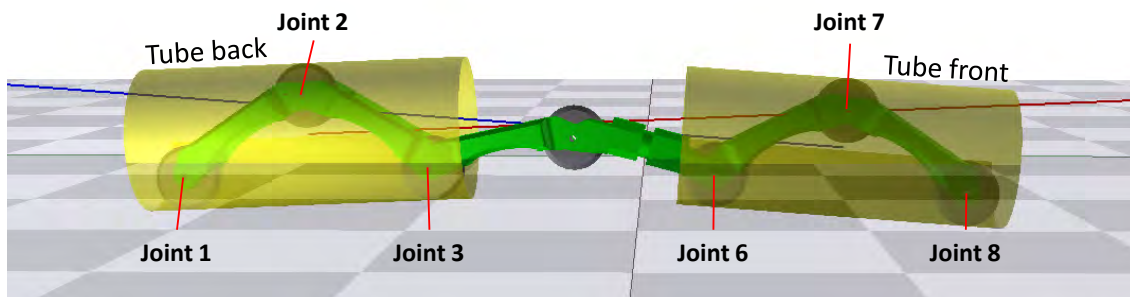


Figure 3.15: Pipe fitting

For fitting the pipes on the robot at the back and the front the position, orientation, and diameter of the pipes must be determined. The position of the pipe is determined by the three joints who are involved in the clamping process. In the equations 3.1 and 3.2 the calculation of the positions of the tubes is given.

$$\vec{pos_tubefront} = \begin{bmatrix} (pos_joint6[1] + 2 \cdot pos_joint7[1] + pos_joint8[1])/4 \\ (pos_joint6[2] + 2 \cdot pos_joint7[2] + pos_joint8[2])/4 \\ (pos_joint6[3] + 2 \cdot pos_joint7[3] + pos_joint8[3])/4 \end{bmatrix} \quad (3.1)$$

$$\vec{pos_tubeback} = \begin{bmatrix} (pos_joint1[1] + 2 \cdot pos_joint2[1] + pos_joint3[1])/4 \\ (pos_joint1[2] + 2 \cdot pos_joint2[2] + pos_joint3[2])/4 \\ (pos_joint1[3] + 2 \cdot pos_joint2[3] + pos_joint3[3])/4 \end{bmatrix} \quad (3.2)$$

The orientation of the pipes is determined by multiplying the required rotation matrices:

$$R_{tubefront} = R_{main} \cdot R_{rotationfront} \cdot R_{joint6} \cdot R_{tubefront_offset} \cdot R_{joint7_half} \quad (3.3)$$

$$R_{tubeback} = R_{main} \cdot R_{rotationback} \cdot R_{joint4} \cdot R_{joint3} \cdot R_{tubeback_offset} \cdot R_{joint2_half} \quad (3.4)$$

Finally the radius and therefore also the diameter of the pipes is calculated by:

$$radius_tubefront = \| \vec{pos_tubefront} - \vec{pos_joint7} \| + wheelradius \quad (3.5)$$

$$radius_tubeback = \| \vec{pos_tubeback} - \vec{pos_joint2} \| + wheelradius \quad (3.6)$$

Environmental feature estimation

In order to determine the environment the robot is in, six common environmental features can be detected in the world model. These are shown in figure 3.16. The environmental features contain pipe diameter transitions (which can be determined by the pipe diameter estimation process) and different angle configurations according to those transitions.

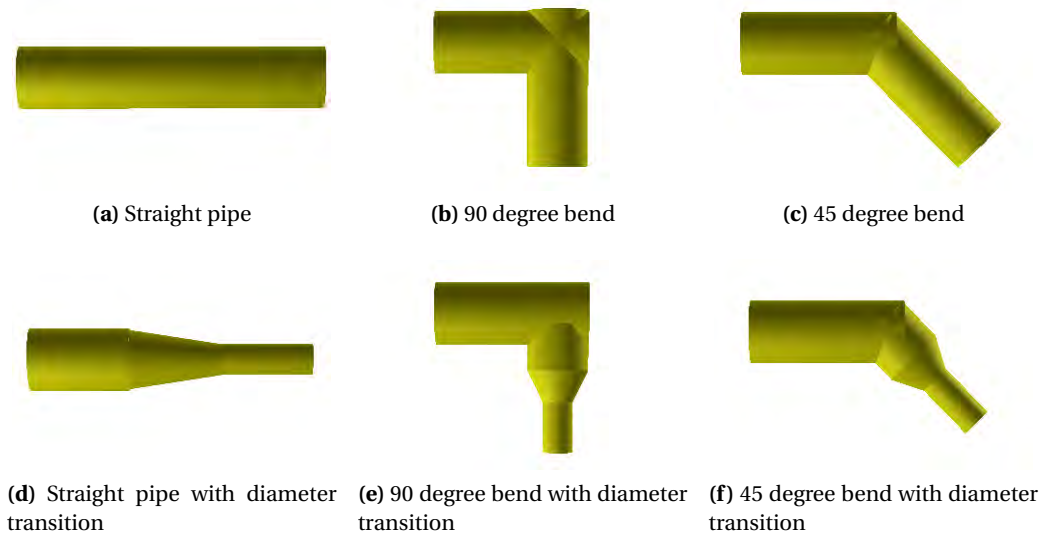


Figure 3.16: Environmental features

The angle between the two pipes is required to detect the correct feature. To obtain a vector which is along the pipe the third column of the rotationmatrix of the pipe should be extracted (since the Z-axis is in the center and along the pipe).

$$vector_tubefront = R_{tubefront} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (3.7)$$

$$vector_tubeback = R_{tubeback} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (3.8)$$

These vectors are shown in figure 3.15 as the red and blue line. The inner and cross product of the vectors are calculated to determine the angle between the two pipes:

$$inner = \langle vector_tubefront, vector_tubeback \rangle \quad (3.9)$$

$$cross = \| vector_tubefront \times vector_tubeback \| \quad (3.10)$$

$$angle_frontback[rad] = atan2(cross, inner) \quad (3.11)$$

Matching model

In order to give a quality value to the detected environmental feature a matching model is made. This matching model has a lower and upper limit of the angle to be measured. This lower and upper limits are given for a straight pipe, 45 degree bend and 90 degree bend:

	Lower limit		Upper limit	
	[deg]	[rad]	[deg]	[rad]
Straight pipe	0	0	20	$\frac{\pi}{9}$
45 degree bend	25	$\frac{5\pi}{36}$	65	$\frac{13\pi}{36}$
90 degree bend	70	$\frac{7\pi}{18}$	110	$\frac{11\pi}{18}$

Table 3.2: Lower and upperlimits of the angle between the pipes

The equations for matching based on the angle between the pipes are:

$$match_straight_pipe[\%] = (1 - angle_frontback \cdot \frac{9}{\pi}) \cdot 100\% \quad (3.12)$$

$$match_45_degree_bend[\%] = (1 - |\frac{\pi}{4} - angle_frontback| \cdot \frac{9}{\pi}) \cdot 100\% \quad (3.13)$$

$$match_90_degree_bend[\%] = (1 - |\frac{\pi}{2} - angle_frontback| \cdot \frac{9}{\pi}) \cdot 100\% \quad (3.14)$$

The matching percentage is correlated with the color of the detected feature. The color varies from red to green which indicates a matching percentage of zero to 100% respectively.

Now the shape of the feature is detected, there are two possible solutions left for the feature to detect. If the diameters of the pipes differ more than 10mm from eachother a pipe diameter transistion is assumed. With this given the solution of the finded feature is trivial.

4 Testing and simulation

In this chapter the implemented design from chapter 3 is tested and evaluated. First the wireless bi-directional communication between pirate and the pc is tested. After that, the necessary calibration procedure for the joint angles of pirate is described. Finally the performance of the world model is tested by carry out the pipe diameter estimation and environmental feature estimation.

4.1 Testing the wireless bi-directional data link

In this section the entire communication channel from pirate to pc is tested. The path the state data needs to travel through goes from the local motorcontrollers via the I2C-bus to the main-controller where the statevector is send into the air. The docking station receives the statevector and send it to the pc by the usb-serial driver running on it. On the pc there is a terminal program called 'termite' running which is connected to the comport which belongs to the docking station.

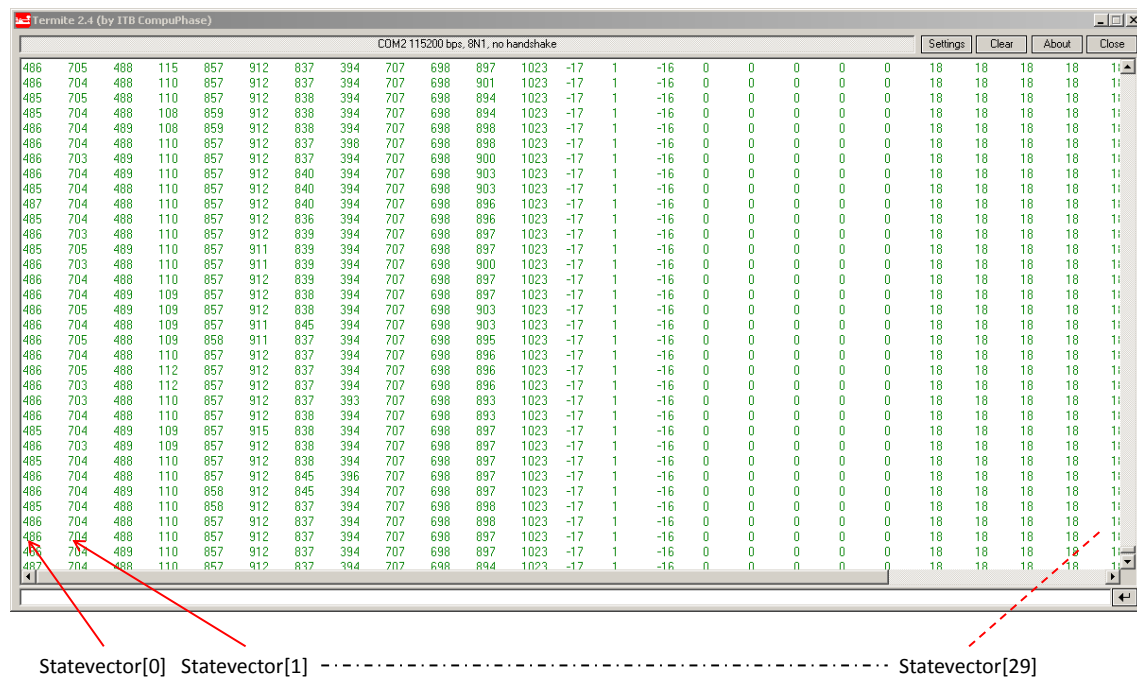


Figure 4.1: Output of terminal program: Termite

In order to determine if the connection works fully bidirectional the user must be able to send commands to the pirate as well as receiving state data from pirate. This is done by entering the command 'D1' in the terminal window, which is the command to enable state feedback. When pirate receives the D1-command it starts transmitting statedata on a frequency of 50Hz. Figure 4.1 shows the output of the terminal program. One can see all the states are received by the terminal program. To terminate the state feedback the command 'd0' is entered.

4.1.1 Wireless link delay

To examine the realtime behaviour of the system the delay of the wireless link is measured. This test is done by toggling a debugpin on the mainboard of pirate when a beacon packet is transmitted. On the docking station a debugpin is toggled when a beaconpacket is received. In this way the delay between sending and receiving a packet can be determined. Figure 4.2

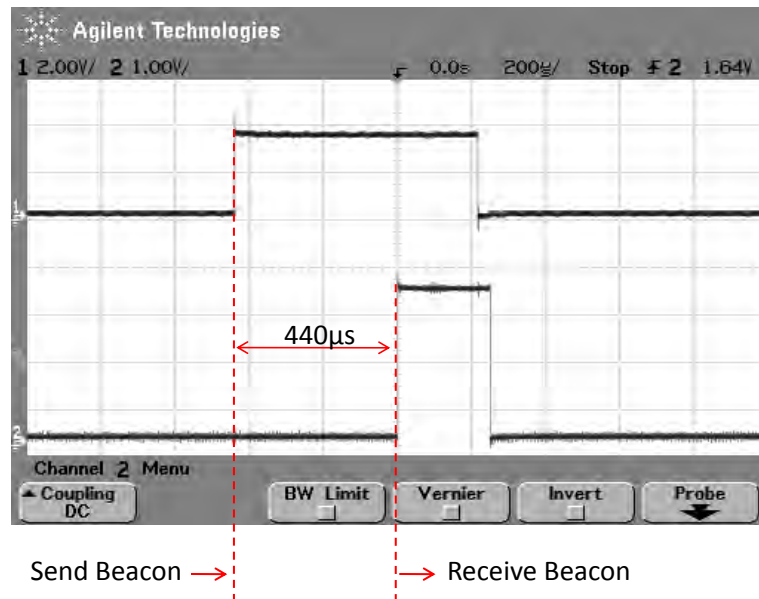


Figure 4.2: Delay between pirate and docking station

shows there is a delay of $440\mu\text{s}$ between sender and receiver when sending a packet of 32 bytes (beacon packet).

4.2 Calibration joint angles

To perform a realtime animation of pirate in the world model the potentiometers which are measuring the joint angles needs to be calibrated. In this calibration process the ADC values of the potentiometers with a zero and -90 degree angle must be determined. The founded values are later on adapted to the world model.



Figure 4.3: Calibration of the joint angles

The first step is to determine the ADC values of the joints with a zero degree angle. This is the pose when the robot is complete stretched out. This can be seen in figure 3.13. The ADC values of the joint angles are obtained by enabling state feedback from the terminal program which is described in the section before.

Figure 4.3 shows there is used a square tool to adjust the joints to a reference angle of -90 degree. This calibration is repeated for the joints 2, 3, 4, 6 and 7. For certainty the calibration is performed twice. The results are given in table 4.1.

Measurementnr.	$\varphi=0$		$\varphi=-\pi$	
	1	2	1	2
Joint2 Angle(φ)	377	389	102	110
Joint3 Angle(φ)	858	868	591	590
Joint4 Angle(φ)	636	639	913	915
Joint6 Angle(φ)	856	851	576	566
Joint7 Angle(φ)	393	406	125	130

Table 4.1: ADC values of potentiometers attached to the joints of the robot

4.3 Pipe diameter and environmental feature estimation

In this section the performance of the pipe diameter estimation and environmental feature estimation is tested. First the pipe diameter estimation algorithm is tested on straight pipes with a different diameter. After that, the remaining five defined environmental features are simulated and evaluated. Since the environmental features are not all available in the lab of the control engineering group, they are faked from straight perspex pipes of different diameters. There are used three pipes in this experiment which all have a different diameter of respectively 56mm, 86mm and 116mm.

4.3.1 Pipe diameter estimation of straight pipes

The pipe diameter estimation is tested on the three available perspex pipes. First the 56mm pipe is examined:

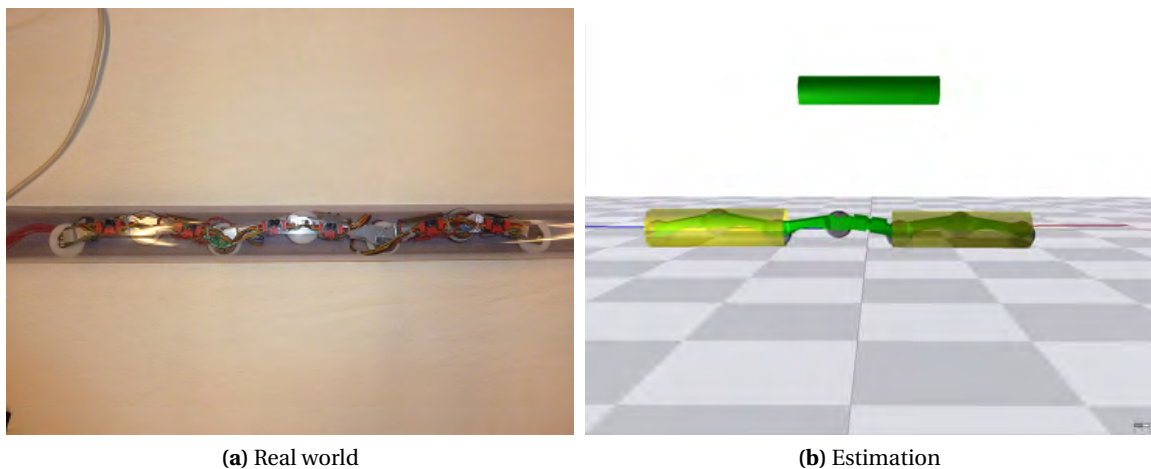


Figure 4.4: Straight 56mm perspex pipe

From figure 4.4 it can be seen in the real setup (figure 4.4a) and in the animation editor (figure 4.4b). The orientation, position and diameter of the pipes at the back and the front are estimated and visualized by the two yellow pipes in the model. The detected feature is displayed in the animation editor above the model.

	Real world	Estimation	Error
Diameter pipe front [mm]	56	56,28	0,28
Diameter pipe back [mm]	56	56,37	0,37
Angle between pipes [deg]	0	0,76	0,76
Match [%]	100	96,18	3,82

Table 4.2: Results of straight 56mm pipe

The bright green color of the feature in figure 4.4b indicates a good match is found. Specific results of this experiment are given in table 4.2.

The second experiment was done with a 86mm pipe which can be seen in figure 4.5. One can see the color of the detected feature (straight pipe) is a little bit more dark compared to the color of the feature in the experiment before. The color gives information about the feature matching based on the angle between the pipes. The color of the feature in figure 4.5b is dark green since the two yellow pipes are not exactly in line with each other. Measurement results are given in table 4.3.

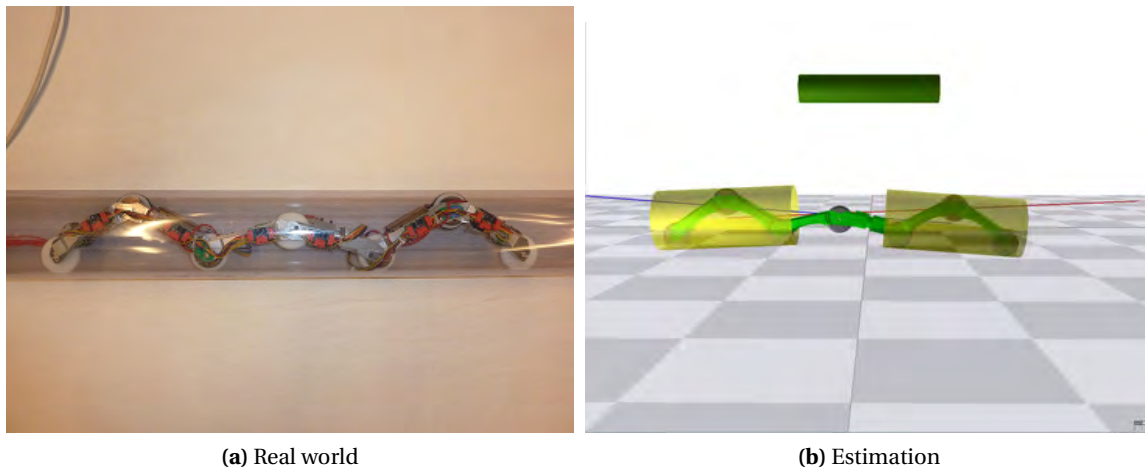


Figure 4.5: Straight 86mm perspex pipe

	Real world	Estimation	Error
Diameter pipe front [mm]	86	85,44	0,56
Diameter pipe back [mm]	86	87,88	1,88
Angle between pipes [deg]	0	5,80	5,80
Match [%]	100	70,98	29,02

Table 4.3: Results of straight 86mm pipe

The next and also last experiment of pipe diameter estimation with straight pipes is done with a 116mm pipe. In this experiment some problems arise which can be seen in figure 4.6.

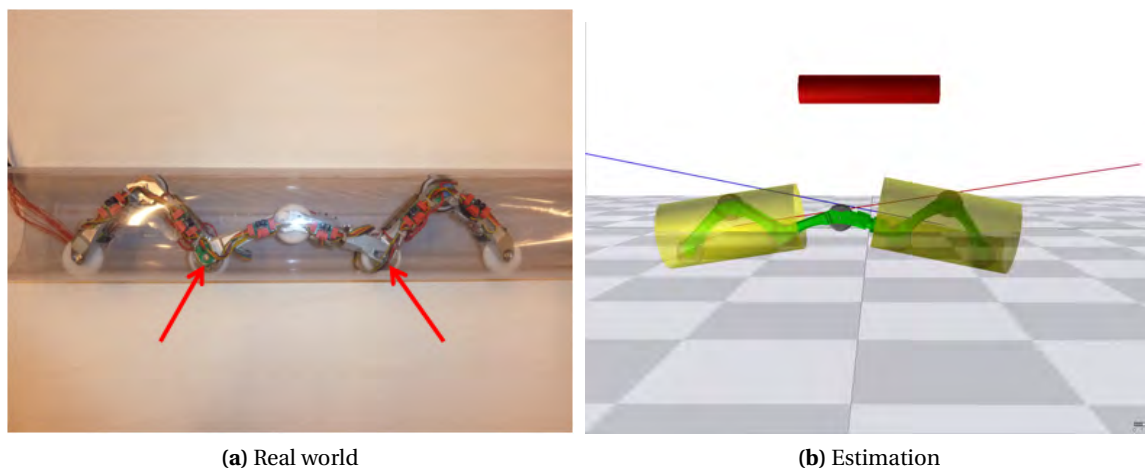


Figure 4.6: Straight 116mm perspex pipe

The red color of the feature indicates a bad match of a straight pipe which is confirmed by the measurement results in table 4.4. This mismatch is caused by the potentiometers which measures the joint angles 3 and 6. Further investigation of the incoming ADC values of joint angle 3 and 6 has led to signaling overflow errors. The potentiometers are attached to the robot in such a way the measureable angle range is not large enough to measure a 116mm pipe.

	Real world	Estimation	Error
Diameter pipe front [mm]	116	104,33	11,67
Diameter pipe back [mm]	116	96,67	19,33
Angle between pipes [deg]	0	19,65	19,65
Match [%]	100	1,75	98,25

Table 4.4: Results of straight 116mm pipe

4.3.2 Environmental feature estimation of angled pipes

In this section the environmental feature estimation is performed with angled pipes. Since the overflow of the ADC values of the potentiometers attached to joint 3 and 6 when measuring a 116mm pipe this pipe is not used anymore in the following experiments. The first experiment is done with 86mm pipes which are fixed at an angle of 45 degree. Figure 4.7 shows the real setup and the estimation in the world model. Results are given in table 4.5.

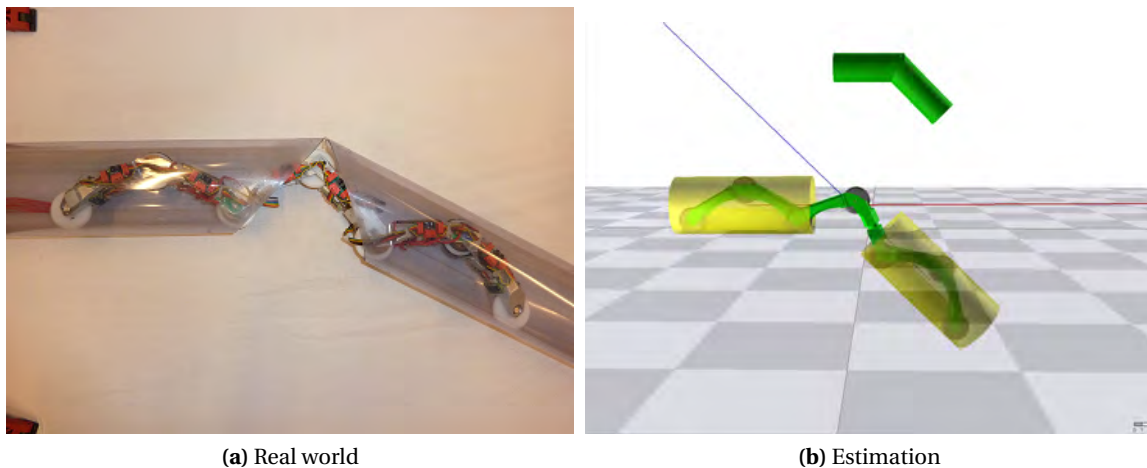
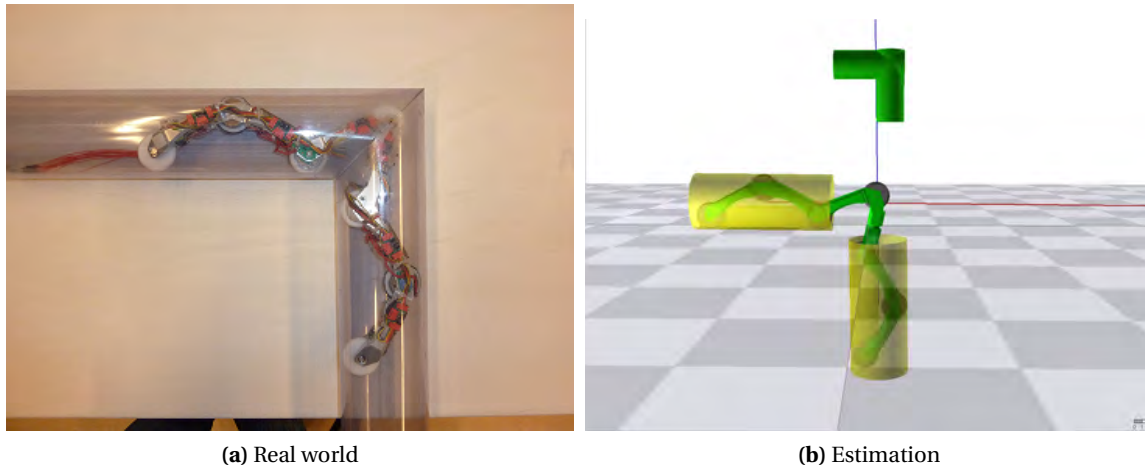


Figure 4.7: 45-degree 86mm bend

	Real world	Estimation	Error
Diameter pipe front [mm]	86	87,82	1,82
Diameter pipe back [mm]	86	84,70	1,30
Angle between pipes [deg]	45	44,34	0,66
Match [%]	100	96,71	3,27

Table 4.5: Results of 45-degree 86mm bend

The next experiment is done with 86mm pipes which are fixed at an angle of 90 degree. Figure 4.8 shows the real setup and the estimation in the world model. Results are given in table 4.6.

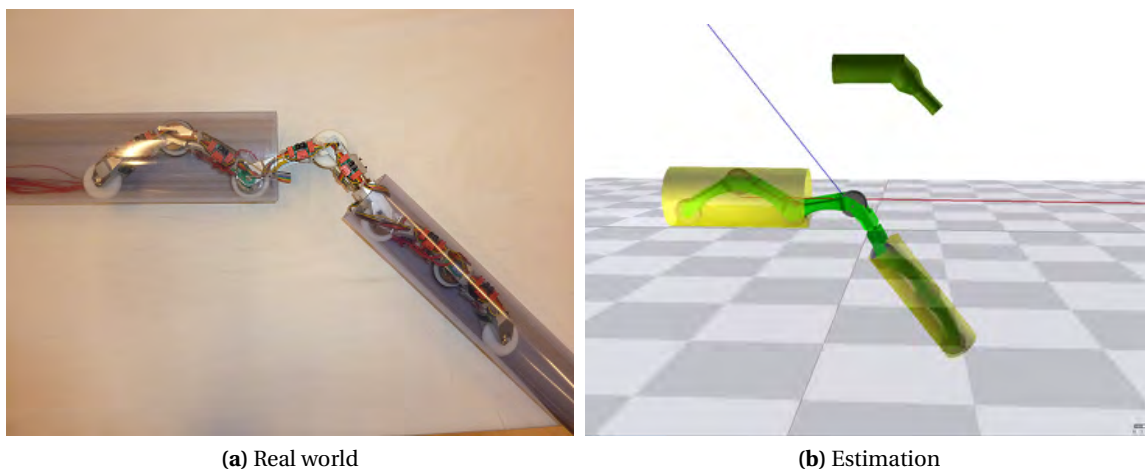
**Figure 4.8:** 90-degree 86mm bend

	Real world	Estimation	Error
Diameter pipe front [mm]	86	86,10	0,10
Diameter pipe back [mm]	86	82,12	3,88
Angle between pipes [deg]	90	88,76	1,24
Match [%]	100	93,84	6,16

Table 4.6: Results of 90-degree 86mm bend

4.3.3 Environmental feature estimation of angled pipes with diameter transition

The next experiments are performed with angled pipes and the addition of a diameter transition according to this feature. In figure 4.9a the experiment with 45 degree bend with 86mm to 56mm pipe transition is shown. The feature which is detected indicates a 45 degree bend and the transition to a smaller pipe. This is correct since the world model assumes a pipe diameter change occurrence when there is a diameter difference larger than 10mm between the two pipes. Table 4.7 show the measurement results.

**Figure 4.9:** 45-degree bend with 86mm to 56mm pipe transition

This last experiment is repeated with a 90 degree bend. Figure 4.10 shows the real setup and the estimation in the world model. Results are given in table 4.8.

	Real world	Estimation	Error
Diameter pipe front [mm]	56	54,31	1,69
Diameter pipe back [mm]	86	85,65	0,35
Angle between pipes [deg]	45	52,09	7,09
Match [%]	100	64,52	35,48

Table 4.7: Results of 45-degree bend with 86mm to 56mm pipe transition

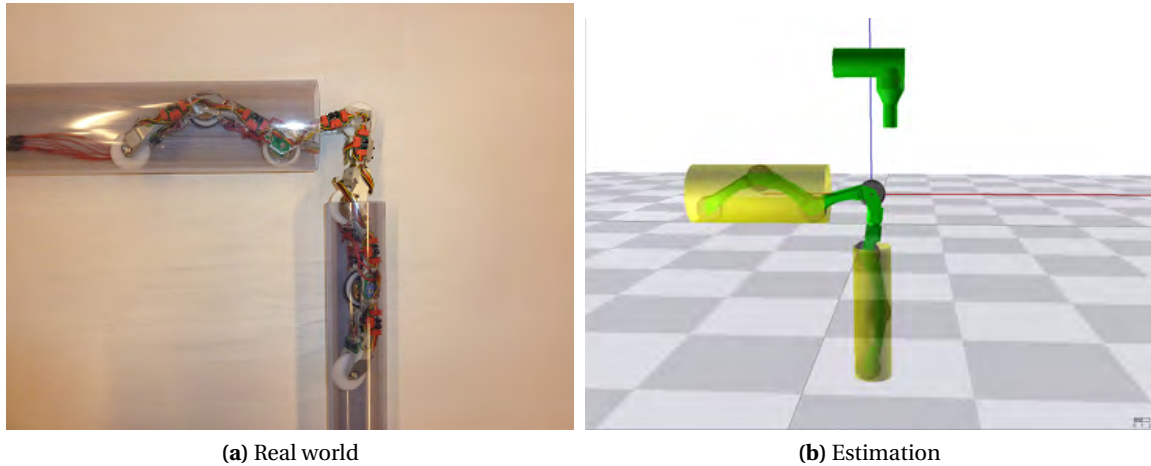


Figure 4.10: 90-degree bend with 86mm to 56mm pipe transition

	Real world	Estimation	Error
Diameter pipe front [mm]	56	56,29	0,29
Diameter pipe back [mm]	86	83,89	2,11
Angle between pipes [deg]	90	87,26	2,74
Match [%]	100	86,32	13,68

Table 4.8: Results of 90-degree bend with 86mm to 56mm pipe transition

4.3.4 Environmental feature estimation of straight pipe with diameter transition

Finally the last environmental feature is simulated which consists of a straight pipe with a pipe diameter transition. Figure 4.11 shows the real setup and the estimation in the world model. Results are given in table 4.9.

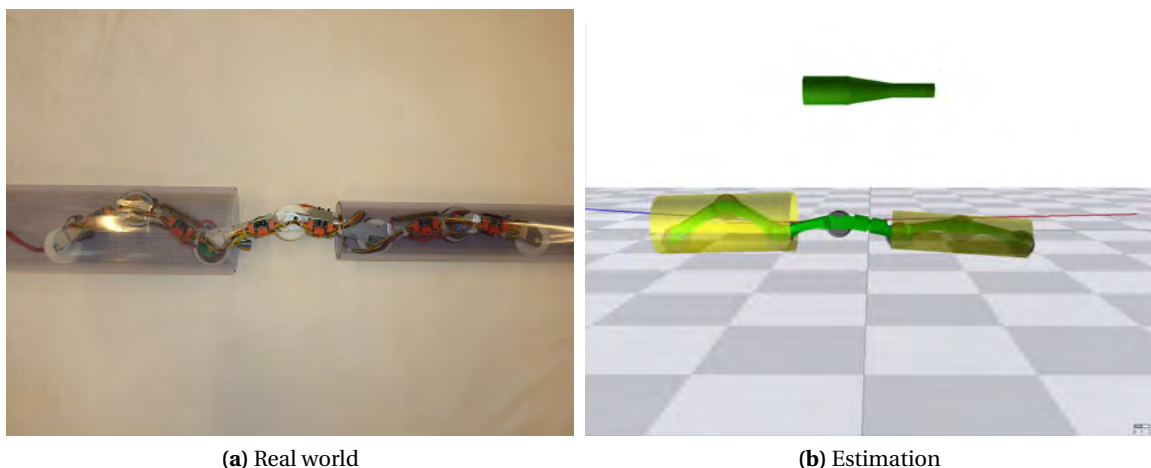


Figure 4.11: Straight pipe with 86mm to 56mm pipe transition

	Real world	Estimation	Error
Diameter pipe front [mm]	56	55,56	0,44
Diameter pipe back [mm]	86	85,68	0,32
Angle between pipes [deg]	0	5,14	5,14
Match [%]	100	74,26	25,74

Table 4.9: Results of straight pipe with 86mm to 56mm pipe transition

4.4 Conclusions of the pipe diameter and environmental feature estimation

In this chapter the pipe diameter and environmental feature estimation is tested by doing experiments with different pipes in different angle configurations. The goal of detecting all the six environmental features is successfully achieved. It is not possible to differentiate a 90 degree bend of a T-joint with only state feedback used in this world model. However, when incorporating the vision system developed by (Mennink, 2010) this will be possible.

The measurement results gives reasonable values except for measuring a 116mm or greater pipe. This is caused by the overflow errors of potentiometers 3 and 6. The absolute error value in the tables shown with the experiments are mainly caused by the backlash of the potentiometers with the shaft.

5 Discussion

This chapter discusses several conclusions that can be made about the different elements used and mentioned throughout this thesis. The final section makes a proposal for future development of a fully autonomous inspection robot a step closer to reality.

5.1 Conclusions

The goal of this research was to develop a wireless communication system which can be used for state feedback between the pipe inspection robot and the outside world. This report demonstrates that the robot indeed is able to transmit its state data wireless to a docking station which in turn will pass this data to a PC through a USB connection. From experimental results it follows that the required update frequency of 50Hz is achieved.

The analysis of the I2C communication bus (which is used for the internal communication on the robot) has led to an increase of the data rate up to 343KHz. With this data rate the robot is able to obtain all the states from the motorcontrollers within a 20ms timeslot which corresponds to a frequency of 50Hz. However, since the I2C protocol is based on a non-differential bus it suffers from interference caused by the DC motors present on the robot which can disturb the bus drastically and therefore results in unpredictable response from the motorcontrollers.

During this research a prototype of a docking station is build which provides the communication bridge between pirate and the PC. The developed wireless protocol which is used by pirate and the docking station provides a bidirectional connection used for uploading parameters to pirate on the one hand and retrieving state data on the other hand. From a compatibility point of view the connection from the docking station to the PC is performed using the USB bus. Furthermore the docking station is able to do a firmware update of the pirate software using the onboard umbilical connector.

On the PC world modeling based on state feedback is done using the 20sim environment. With the in C++ written multithreaded dll a stable communication is acquired between the 20sim application and a RS232 port. The 3D animation editor within 20sim is used for a realtime graphical representation of pirate. The created world model is able to do a pipe diameter estimation with an average accuracy of 1,4mm. Furthermore six common environmental features can be recognized by the model which can be used later on for manouvring to the underground gasnetwork autonomously.

5.2 Recommendations

For future work some recommendations can be made. These recommendations are divided into Electrical enhancements and Software and control.

5.2.1 Electrical enhancements

Angle measurement

The current methodology of measuring the joint angles of the robot is performed using potentiometers. Accurate feedback of the joint angles can be obtained as long as there are no motors turned on which are interfering the potentiometer output signal. Another disadvantage of this methodology is the limited measurable angle range which is not large enough for joint 3 and 6 to measure pipes of 116mm or larger.

In order to avoid these problems the use of absolute encoders have to be investigated. The encoders can be attached to the shaft of the joint similar to the potentiometers. However, they

are able to do a much more accurate angle reading compared to the potentiometer even when motors are turned on. A known problem with these absolute encoders is their relative large size which make them not suitable for fitting on the robot. Development of custom made absolute encoders seems to be necessary for implementing this new approach of measuring joint angles.

Internal communication bus

In this research the internal I2C communication bus is used for the distributed control system. As already mentioned the I2C bus is not a very stable communication bus regarding interference problems and therefore maybe not the best solution in this project. Other bus systems have to be investigated to find out if there is a more suitable protocol available.

The protocols which can be proposed are for example CAN and RS485. The RS485 communication bus is a electrical protocol which is already used for many years in industrial environments whereby interference is commonly present from other equipment. The CAN-bus is mostly used in the automotive industry to perform distributed control systems.

5.2.2 Software and control

Hardware in the loop simulation

The world modeling which is done in this research can be extended by a motion profile and a connection between the world model and this motion profile (see figure 5.1). Doing so, a real hardware in the loop simulation can be performed which can be used to let the robot tackling a environment feature in the laboratory of the control engineering group. Measurement data which is logged in the world model can easily be examined afterwards and will therefore contribute to the development process.

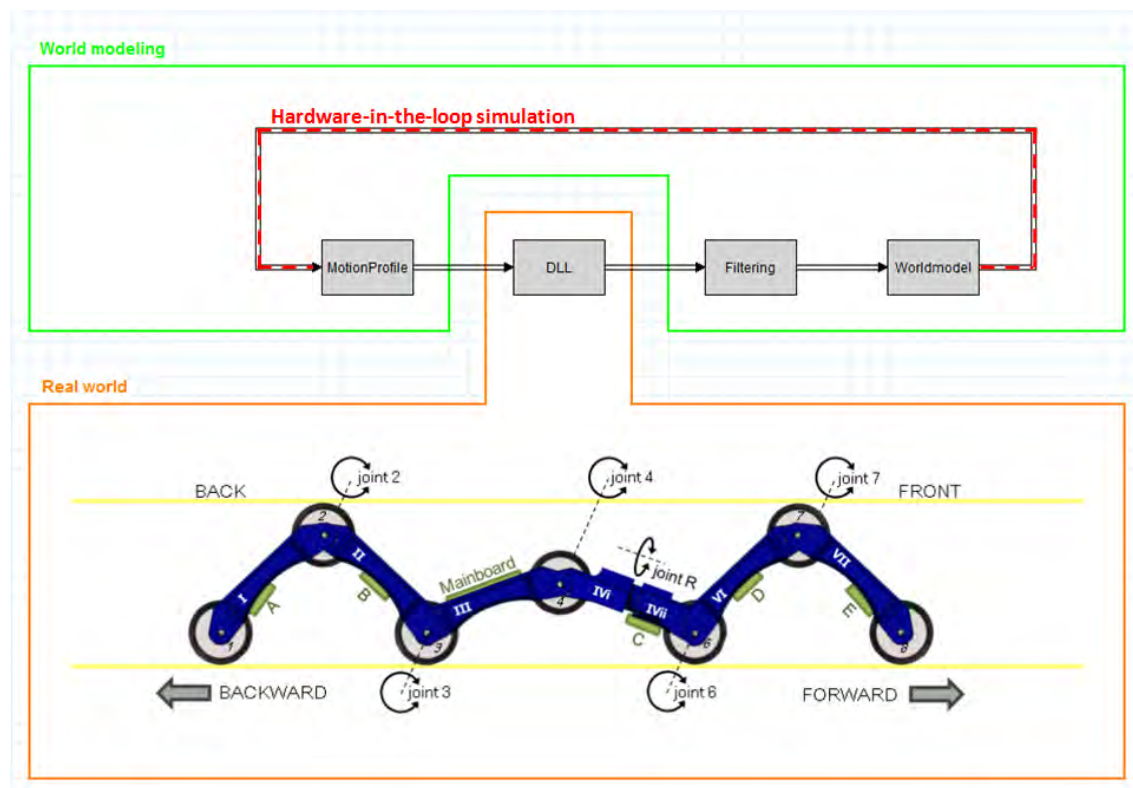


Figure 5.1: Hardware-in-the-loop simulation

Extended state feedback

For tuning the PID controllers present on the motorcontroller the state feedback could be extended by including the PID related signals to the statevector like setpoint, error, process output, integral error. Since the current statevector consists already of 30 variables another option would be to construct a separate PID related statevector which can be enabled by a 'D2' command similar to the current state feedback enabling method.

Firmware update

Currently the pirate project is in development stage and therefore firmware running on the pirate needs to be updated and uploaded to the robot time to time. The programming of the LPC2148 on the mainboard of the robot is currently done by a serial cable which can be a time consuming task when program size increases during development. The use of a USB bootloader could decrease this relative long programming time drastically. In appendix C the way how to use this bootloader is described.

A Mathematical derivations

A.1 Measuring tilt using a three axis accelerometer

In order to define the angles of the accelerometer in three dimensions the pitch, roll and theta are sensed using all three outputs of the accelerometer. Pitch (ρ) is defined as the angle of the X-axis relative to ground. Roll (ϕ) is defined as the angle of the Y-axis relative to the ground. Theta (θ) is the angle of the Z axis relative to gravity

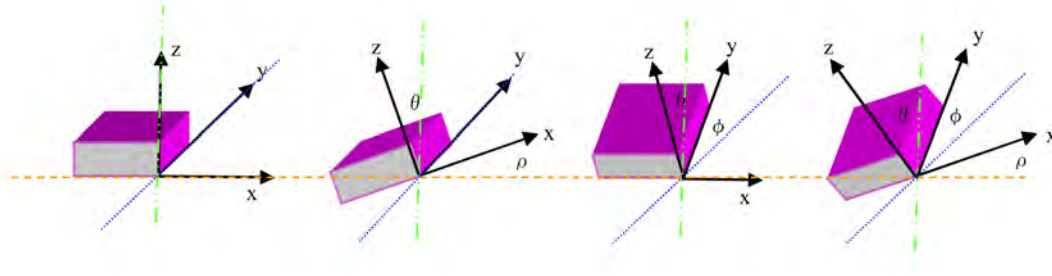


Figure A.1: Three axis for measuring tilt (K. Tuck, 2007)

$$\begin{aligned}\rho &= \arctan \left(\frac{A_x}{\sqrt{A_y^2 + A_z^2}} \right) \\ \phi &= \arctan \left(\frac{A_y}{\sqrt{A_x^2 + A_z^2}} \right) \\ \theta &= \arctan \left(\frac{\sqrt{A_x^2 + A_y^2}}{A_z} \right)\end{aligned}\tag{A.1}$$

Now the acceleration due to gravity on the X-axis, Y-axis and Z-axis are combined. The resultant sum of the accelerations from the three axes is equal to $1g$ when the accelerometer is static.

$$\sqrt{A_x^2 + A_y^2 + A_z^2} = 1g\tag{A.2}$$

B Power supply analysis

B.1 Introduction

The pirate robot must be supplied from two power sources. One power supply of 3,7V for the logic and microcontrollers, and one power supply of 7,4V for powering the DC motors are present on the robot. The current prototype uses an umbilical cord to accomodate the pirate with the two power sources. This is only a solution for testing purposes since the robot must be able to run on batteries when it is operating underground. On the robot itself the power sources are distributed through the before mentioned ribbon cable.

B.2 Encountered problems

From earlier research there are some problems encountered which may be caused by an insufficient power supply. Tests with pirate in the laboratory of the control engineering group has led to software crashes of some motorcontrollers on the robot. When such a software crash of a particular motorcontroller occured there is no possibility anymore to get control of that motorcontroller. A hard reset is the only solution to get the crashed motorcontroller back to life again.

B.3 Software crash analysis

In order to determine the cause of these software crashes further analysis of the problem is done. In the most test cases it is motorcontroller E that will crash. Furthermore, the crash only occured when motors are running and dissipate a certain amount of energy (For example when robot is driving inside a tube both the drive motors are on). Since there seems to be a relation between the software crashes and the energy the robot dissipates reasonably the cause of the problem can be found in the power supply.

In figure B.1 an overview is given the way the power sources are distributed on the robot. R_c represents the internal cable resistance of the umbilical cord. R represents the internal cable resistance from one module to another module whereby the connector-resistance is included.

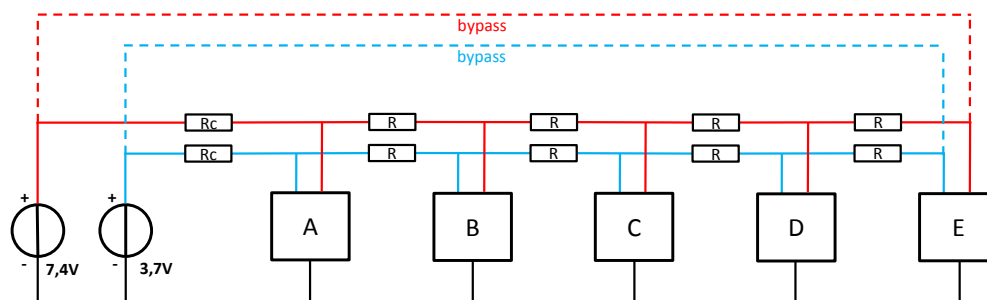


Figure B.1: Added bypass for the two power sources

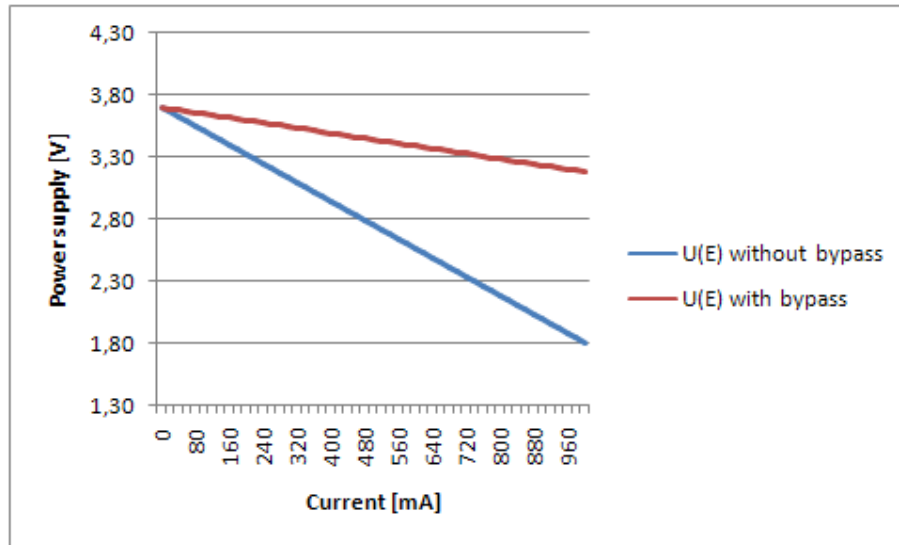
From B.1 it is obvious that motorcontroller E will crash the most of the time since the internal resistance of all the modules are summed up till motorcontroller E. If the robot dissipates energy there is a voltage drop across each internal resistance. The supply voltage of motorcontroller E will collapse dramatically when the current increases.

To compensate for these unwanted resistances in the power lines bypass wires are added in parallel to the power lines. From measurements the values of R_c and R are determined and given in table B.1. Figure B.2 shows the 3,7V supply voltage of motorcontroller E with and without the bypass wires versus current. Since the 3,7V is regulated to a stable 3,3V on the motorcontroller

	without bypass	with bypass
$R [\Omega]$	0,15	0,03
$R_c [\Omega]$	1,3	0,4
$R_{tot} [\Omega]$	1,9	0,52

Table B.1: Cable and connector resistance

there is no problem if the voltage collapse from 3,7V to 3,3V. However, if the voltage collapse under the 3,3V unpredictable behaviour of the controller can occur. In worst case scenario a brown out detection of the motorcontroller can happen.

**Figure B.2:** 3,7V Power supply vs current

B.4 Conclusions

The current powersupply offered by the umbilical cord is the main problem of the software crashes of the motorcontrollers. Besides that, also the ribbon cable with the internal resistances is accessory to the problem. The internal resistances are caused by the connectors and the cable diameter. The current cable diameter of the ribbon cable is not large enough to maintain a stable voltage on motorcontroller E at the end of the cable. The added bypass wires reduce the internal resistance of the power lines and therefore the supply voltage of motorcontroller E remains longer stable when current increases.

C LPC2148 USB Bootloader

C.1 What is a bootloader

A bootloader is a small piece of code that runs before the operating system starts running. In this project the bootloader is the code that runs before the pirate firmware starts up. Typically a bootloader is used because the system memory is too small to contain the entire program, and so the bootloader uses a set of routines to call the program from a different part of memory.

C.2 Why using LPC2148 USB bootloader

Since the ARM architecture allows for such large flash space, loading code onto the LPC2148 ARM7 over the serial port was painfully slow. On the mainboard there is a SD card slot available which is connected to the SPI1 port of the LPC2148 and is used for data logging purposes. Furthermore the LPC2148 has a USB 2.0 full speed compliant device controller onboard which can be used for running the mass storage profile. The bootloader brings all these techniques together to create an easy to use and very fast development system.

The current programming method for the LPC2148 uses the RS232 port and a flash programming tool called 'Flashmagic' which is shown in figure C.1a. When using such a serial programmer to load code onto the LPC2148, it can take several minutes to get the code up and running (around 45 seconds for a 200kB file at programmed at 38400 baud).

This can be a major setback when one is writing very large programs, and starts debugging. If for example only two lines of code are changed, and it takes several minutes to load these changes onto the mainboard, it is very time consuming and a waste of expensive time. Using the LPC2148 USB bootloader allows the user to load code in seconds without the need of any flash programming tool (see figure C.1b).

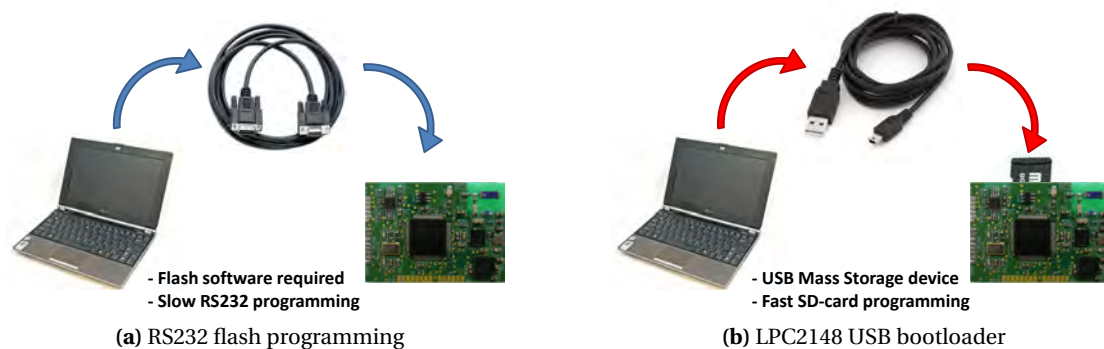


Figure C.1: Programming methods for the LPC2148

C.3 Bootloader performance

The LPC2148 USB bootloader performs three steps:

C.3.1 USB connection

The first task the bootloader performs is checking to see if a USB cable has been plugged in. If the LPC2148 detects the presence of a USB cable then it initiates a USB Mass Storage system. This will cause the mainboard of the pirate robot to appear on any computer platform as a removable flash drive. The user can then seamlessly transfer files to the flash drive. In the background, the LPC2148 moves the user's files onto the SD card using the FAT16 file system.

C.3.2 Firmware file

After a USB connection is made with the PC the bootloader looks for a firmware file on the SD-card. In this project, the bootloader looks for a file named FW.SFE. This file contains the desired operating firmware (in a binary file format) for the LPC2148 microprocessor. If the bootloader finds this file on the FAT16 system then it programs the contents of this file to the flash memory of the LPC2148. In this fashion, the bootloader acts as a programmer for the LPC2148 and a upgrade of the firmware on the LPC2148 can simply be done by loading a new file onto the micro SD card.

C.3.3 Calling Firmware

After performing the above two checks, the bootloader calls the main firmware. The main code should not even know that the bootloader was used and will run normally.

C.4 Using the bootloader

In order to start working with the bootloader first the bootloader itself needs to be programmed into the LPC2148. This can be done with the existing programming method (see figure C.1a). Once the bootloader is programmed it can be used by drag and drop the before mentioned firmware file FW.SFE to the SD-card. After unplugging the USB cable and resetting the mainboard the bootloader will look for the file named FW.SFE. If the file is found the bootloader will program the new code onto the LPC2148 and is running.

C.5 Conclusions

Since the LPC2148 supports two independent SPI ports, the SD-card slot can be attached to one of these. In the mainboard design there is chosen for the SPI1 port which can be seen in appendix G. Unfortunately, the bootloader is not working properly with the SPI1 port. Further investigation in finding the cause of this problem does not has led to a working solution. However, testing the bootloader with the SPI0 port works conveniently. In order to use the bootloader feature the mainboard should be redesigned in such a way the SD card slot is connected to SPI0 port.

D Docking station Installation manual

In this appendix the required installation instructions for using the docking station on a Windows XP machine are described. When plugging in the docking station USB cable into the PC the station will be recognized as USBserial device (see figure D.1) and the 'Found New Hardware Wizard' will be prompted (see figure D.2).

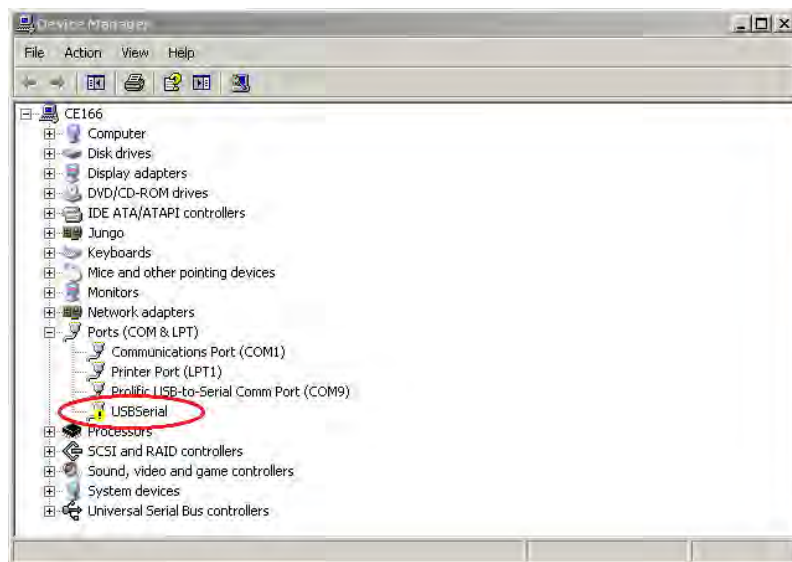


Figure D.1

The wizard will ask to connect to Windows Update to search for software drivers. Select 'No, not this time' and hit Next.



Figure D.2

The next page of the installation wizard will appear on the screen (see figure D.3). Here one can select for manually or automatically install the required device drivers. Since the drivers are available on a CD (ask gerben te Riet/Scholten) choose here 'Install from a list or specific location'.

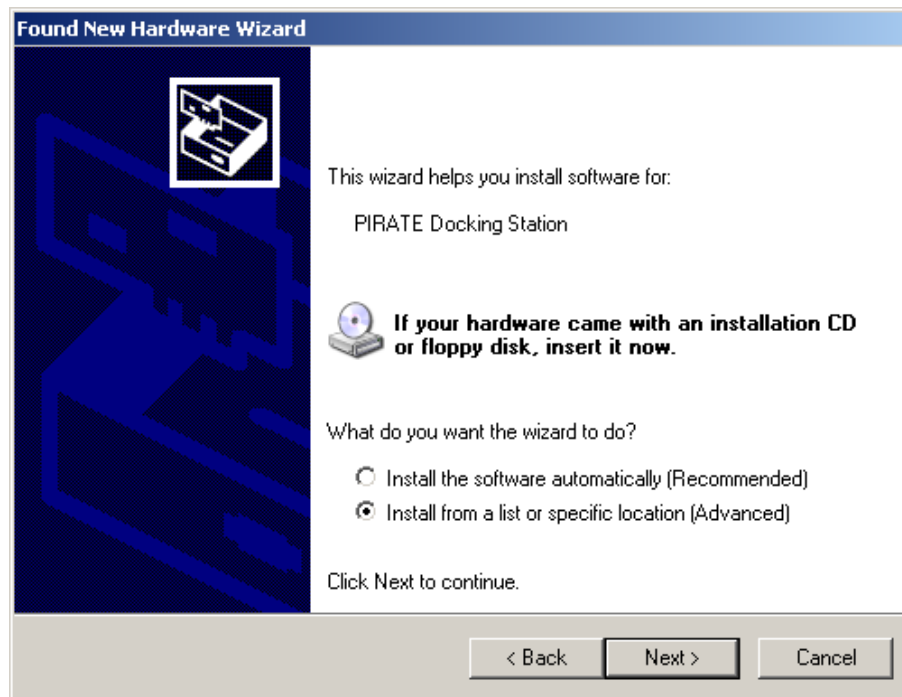


Figure D.3

Browse to the CDrom drive and select the folder "/Docking station/USBserial device drivers for windows/"

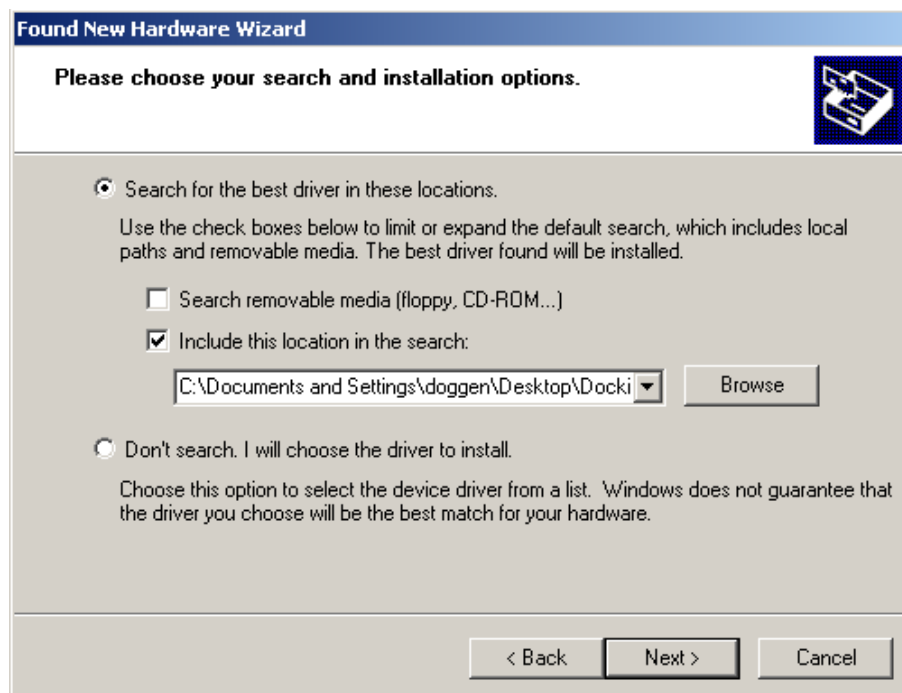


Figure D.4

The wizard installs the drivers now on the system.

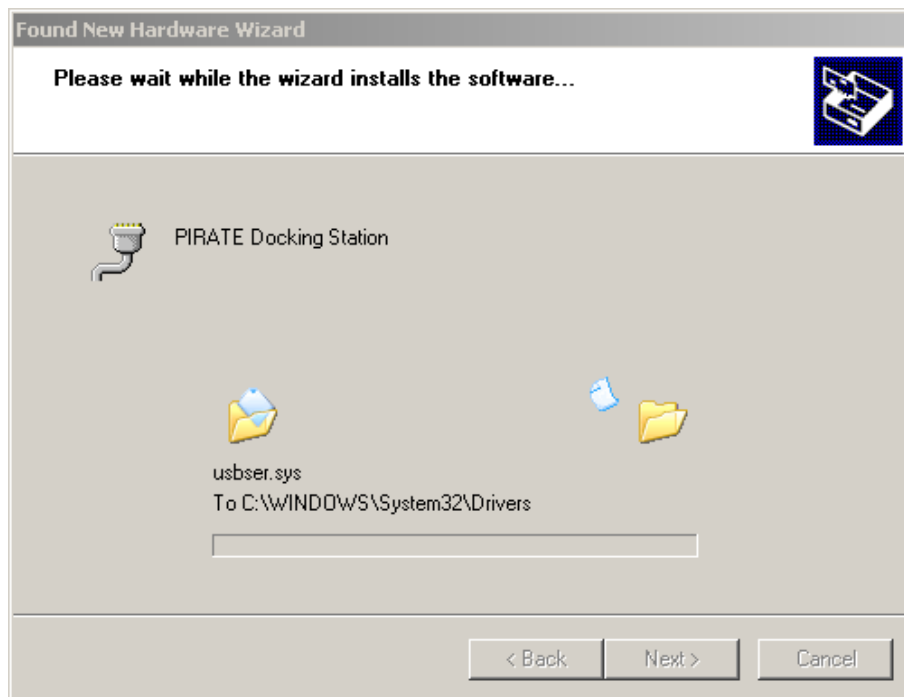


Figure D.5

After a while the user will be prompted with a hardware installation warning. This is nothing to worry about and just click on 'Continue Anyway' to continue with the installation (see figure D.6).

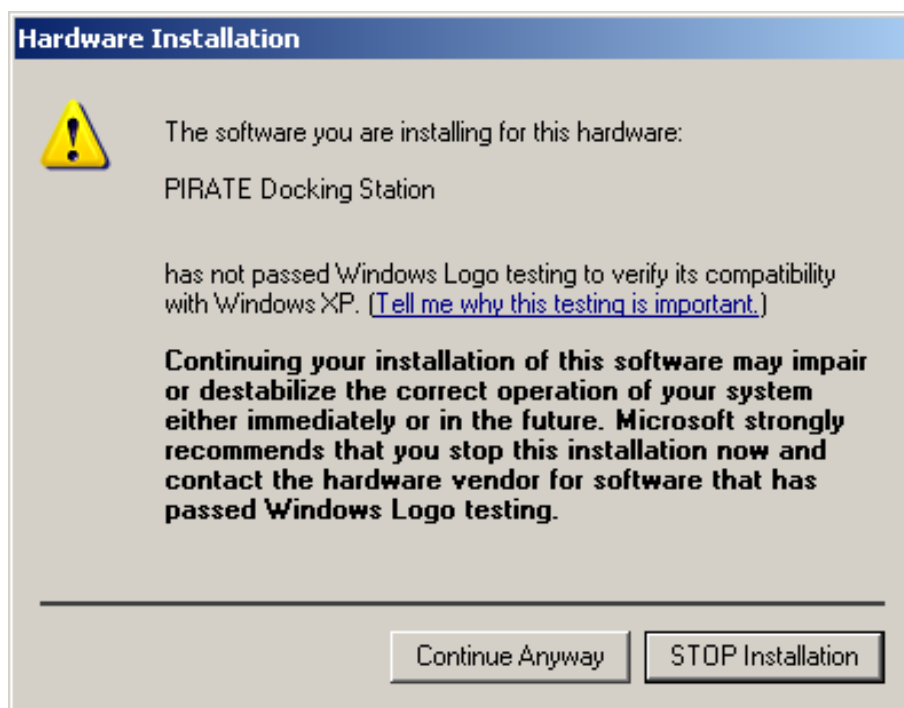


Figure D.6

The next page shows the installation is completed successfully if everything went fine during setup.

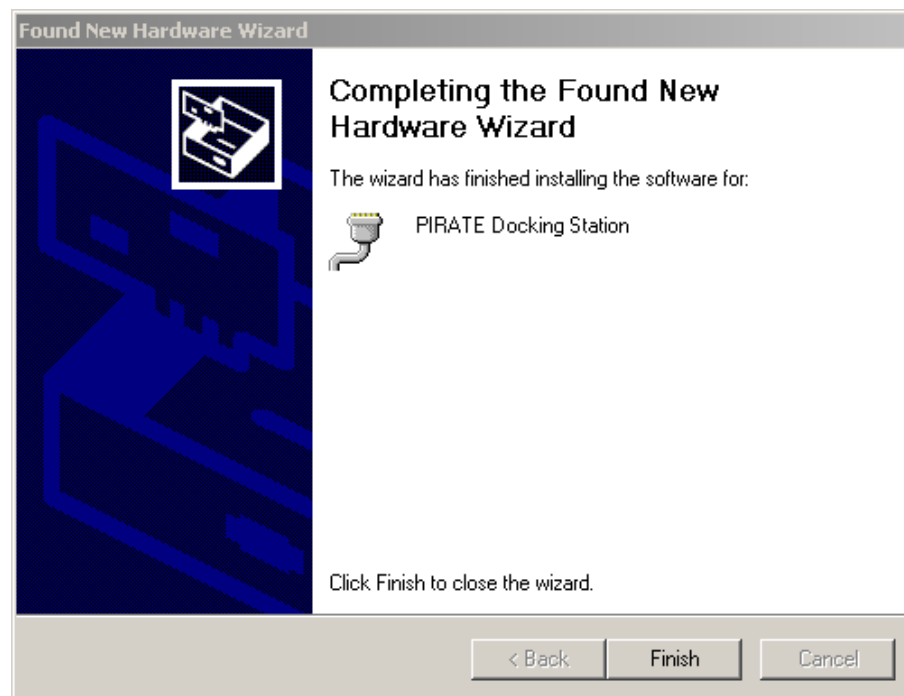


Figure D.7

Finally, in the device manager the docking station can be seen installed successfully since the exclamation mark is vanished. The docking station is now ready for use. Enjoy!

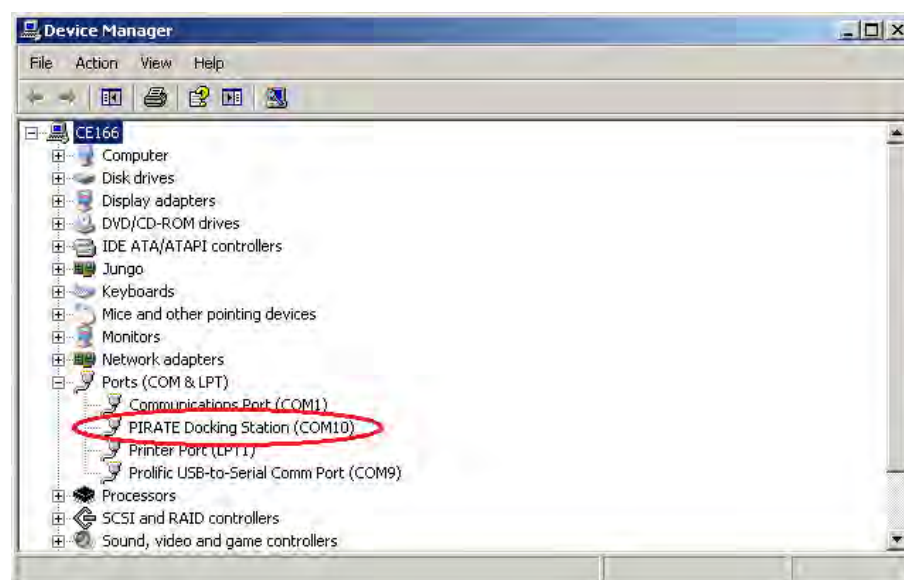


Figure D.8

E Control commands

Command	Commandline function	Arg 1	Arg 2
?m	cmdlineGetPIDmode()	slaveAddress	pidNum
?p	cmdlineGetPgain()	slaveAddress	pidNum
?i	cmdlineGetIgain()	slaveAddress	pidNum
?d	cmdlineGetDgain()	slaveAddress	pidNum
?u	cmdlineGetSetpoint()	slaveAddress	pidNum
?j	cmdlineGetIntegralError()	slaveAddress	pidNum
?w	cmdlineGetIntegralAntiwindup()	slaveAddress	pidNum
?h	cmdlineGetPWMoutputLimit()	slaveAddress	pidNum
?F	cmdlineGetControllerFeedback()	slaveAddress	pidNum
?E	cmdlineGetPositionError()	slaveAddress	pidNum
?D	cmdlineGetDifferentialError()	slaveAddress	pidNum
?Z	cmdlineGetControllerOutputSignal()	slaveAddress	pidNum
?x	cmdlineGetEncoderPosition()	slaveAddress	encNum
?v	cmdlineGetEncoderVelocity()	slaveAddress	encNum
?l	cmdlineGetCurrentLimit()	slaveAddress	
?J	cmdlineGetPosition0()	slaveAddress	
?K	cmdlineGetTorque0()	slaveAddress	
?I	cmdlineGetCurrent()	slaveAddress	
?L	cmdlineGetPosition1()	slaveAddress	
?M	cmdlineGetTorque1()	slaveAddress	
?S	cmdlineGetStatus()	slaveAddress	

Table E.1: Read commands

Command	Commandline function	Arg 1	Arg 2	Arg 3
!m	cmdlineSetPIDmode()	slaveAddress	pidNum	mode
!p	cmdlineSetPgain()	slaveAddress	pidNum	Kp
!i	cmdlineSetIgain()	slaveAddress	pidNum	Ki
!d	cmdlineSetDgain()	slaveAddress	pidNum	Kd
!u	cmdlineSetSetpoint()	slaveAddress	pidNum	setpoint
!w	cmdlineSetIntegralAntiWindup()	slaveAddress	pidNum	windup
!h	cmdlineSetPWMoutputLimit()	slaveAddress	pidNum	pwm
!x	cmdlineEncoderSetPosition()	slaveAddress	encNum	position
!v	cmdlineEncoderSetVelocity()	slaveAddress	encNum	velocity
!s	cmdlineEnablePID()	slaveAddress	pidNum	
!r	cmdlineDisablePID()	slaveAddress	pidNum	
!j	cmdlineSetIntegralError()	slaveAddress	pidNum	
!l	cmdlineSetCurrentLimit()	slaveAddress	currentLimit	
!e	cmdlineBridgeEnable()	slaveAddress		
!q	cmdlineBridgeDisable()	slaveAddress		

Table E.2: Write commands

F List of files

In this section, list of files contained in the CD accompanying this report is given.

F.1 20sim DLL

In the folder "/20sim threaded DLL/20simdll/" the Visual Studio C++ files can be found. In this project Visual Studio 10 is used for development of the DLL. The folder contains various project files as well as the source file 20simdll.cpp and the project file 20simdll.dsp.

F.2 20sim world modeling

The world modeling in 20sim related files can be found in the folder "/World modeling/". The files in this folder are listed in F.1.

File	Description
20sim.bmp	Bitmap of pirate
20simdll.dll	Multithreaded RS232 DLL
animation.emx	20sim project file
*.STL	Files to create 3D model

Table F.1: Files belonging to the 20sim world modeling

F.3 Pirate firmware

The pirate firmware can be found in the folder "/PIRATE/". The folder contains a 'inc' and a 'src' folder which contain the header- and c-source files respectively. In the 'src' folder the nrf24l01.c file is added in this project to support wireless data transmission using the Nordic nRF24L01 module. The header file for this source file is nrf24l01.h and is located in the 'inc' folder.

The "/system.pnproj"-file is the project file of the pirate firmware and is opened in the Programmers notepad 2 environment. In the programmers notepad environment the WINARM toolchain is setup and therefore used as compiler for this project (M. Thomas, 2006).

F.4 Docking station firmware

The docking station firmware can be found in the folder "/Docking Station/". In this folder the 'sources' folder contain all the c-source files which are used for creating the firmware. Mainly there the sources can be divided into two sections namely: USB related files, and Wireless communication files. The USB related files can be recognized by usb*.*.

F.5 LPC2148 USB Bootloader

The bootloader source files are located in the folder "/USB Bootloader/". The folder contains two other folders named: 'source' and 'example'. In the 'source' folder the required firmware is found which needs to be uploaded to the controller by a RS232 connection. After that the controller is ready to run the bootloader. The 'example' folder contains a simple example of a flashing led. Compiling the project outputs a file FW.SFE which is the firmware file.



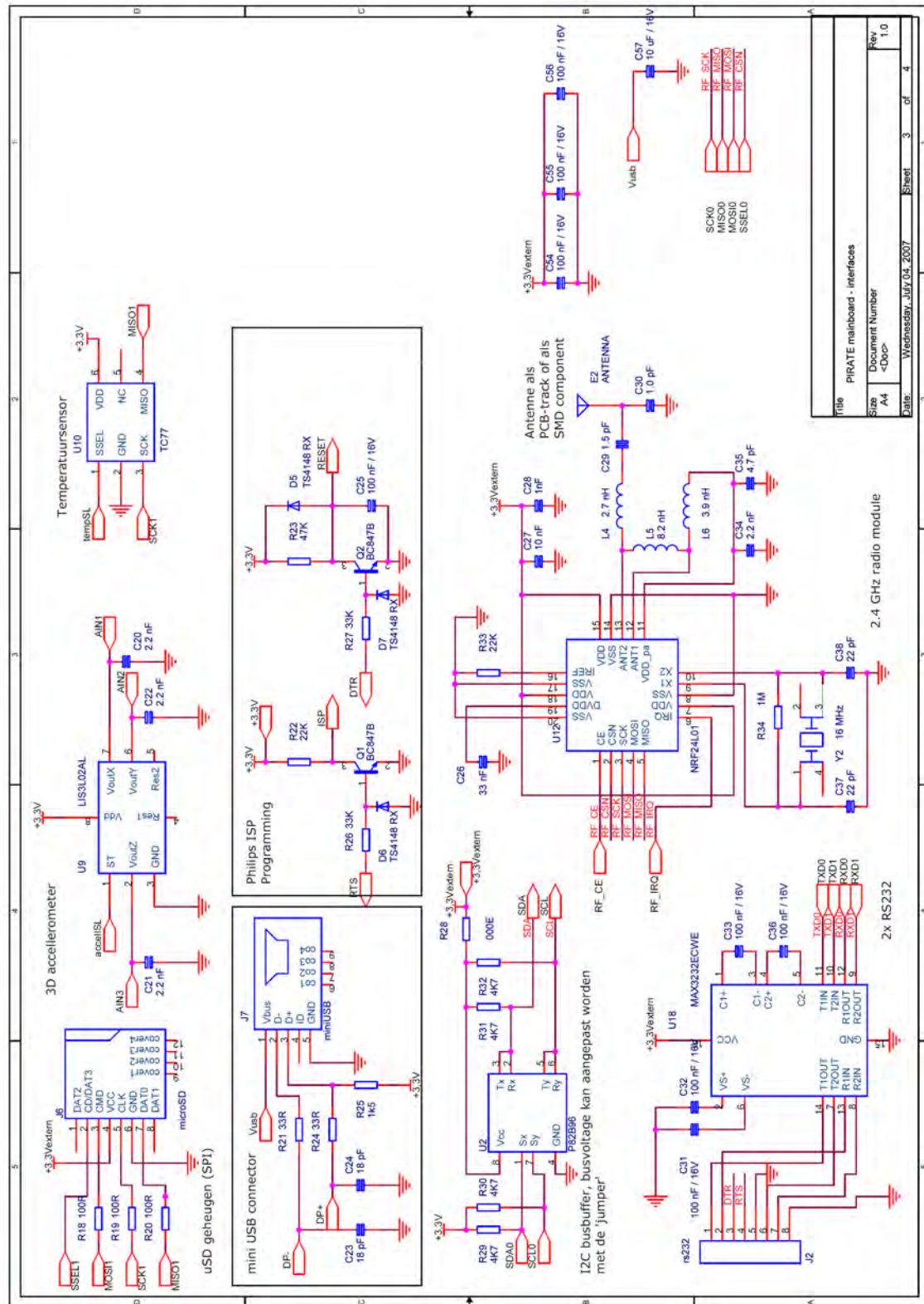


Figure G.3: Used interfaces

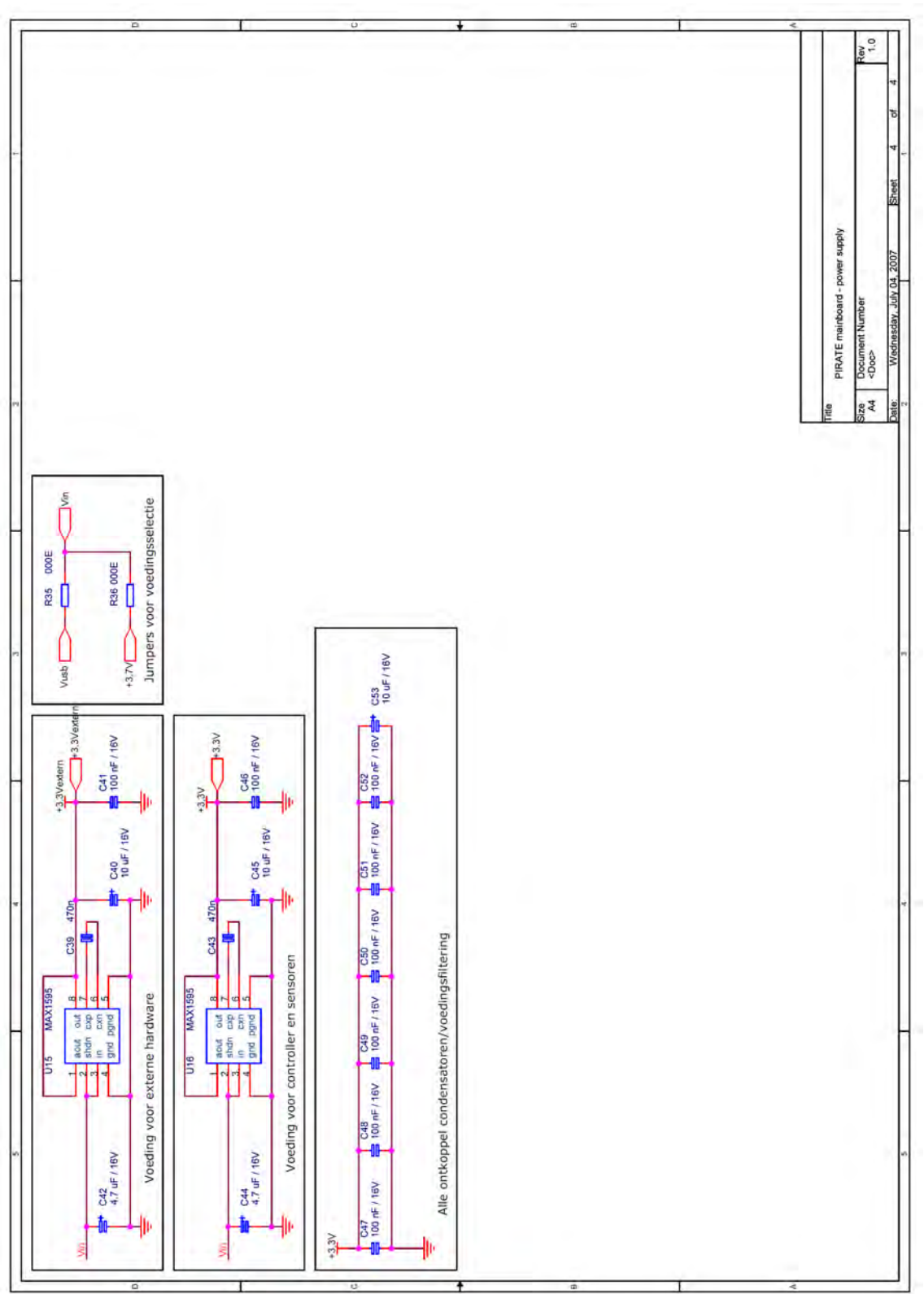


Figure G.4: Power supply

G.2 Motor controller

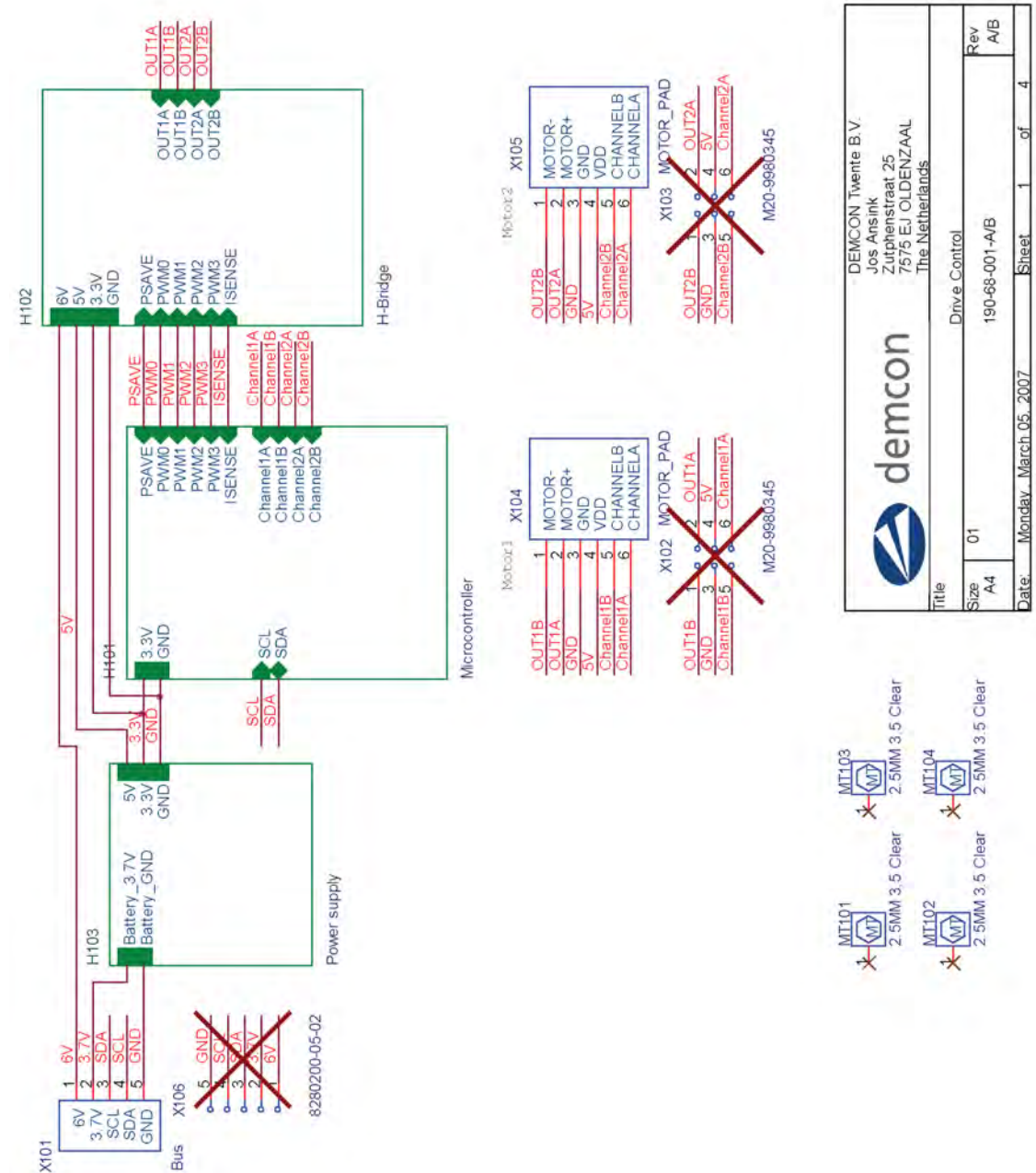


Figure G.5: Block diagram

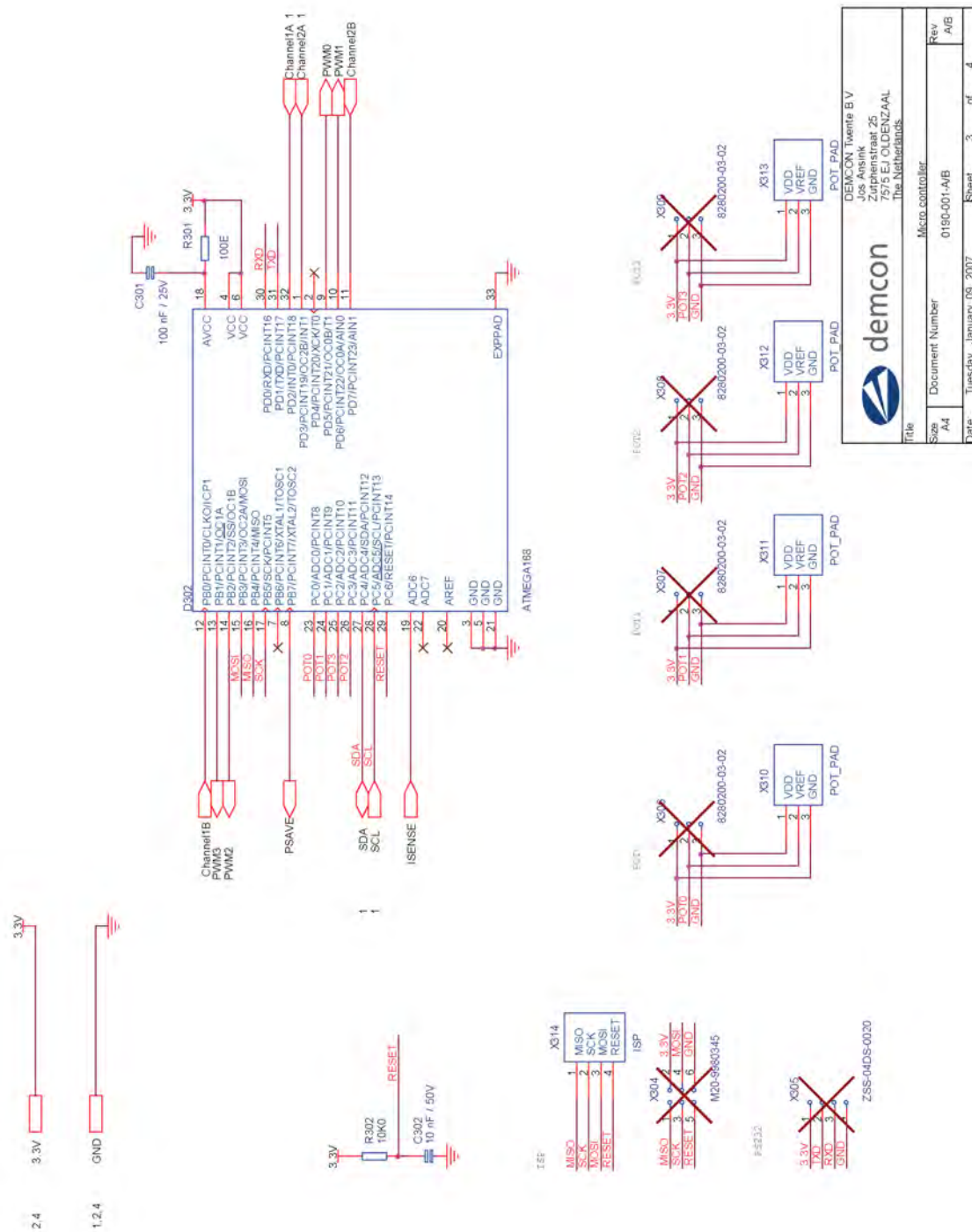


Figure G.6: Microcontroller Atmel ATmega168

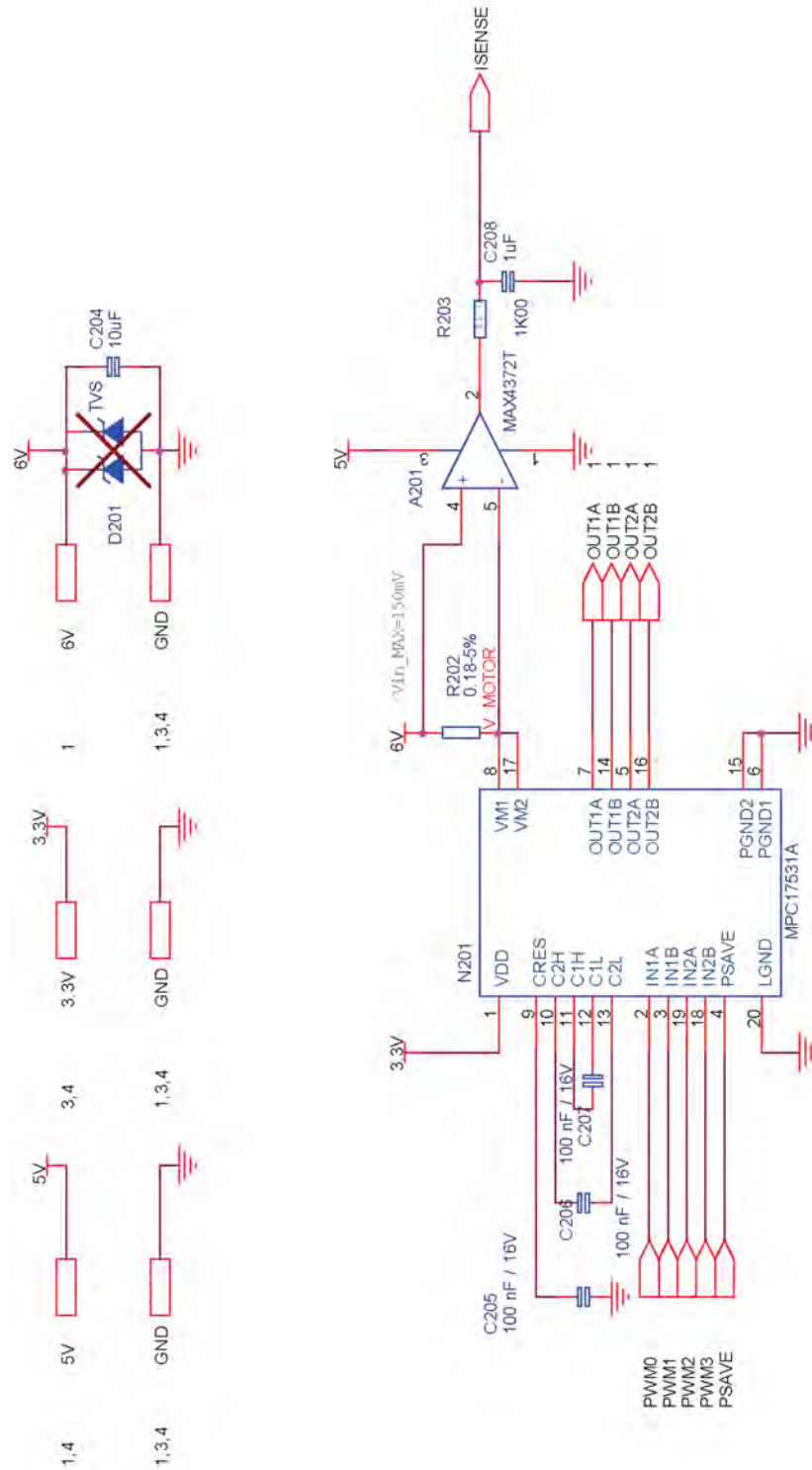



Figure G.7: H-bridge

		DEMCON Twente B.V. Jos Ansink Zuiphenstraat 25 7575 EJ OLDENZAAL The Netherlands	
		Title: H-Bridge	
Size: A4	Document Number: 0190-88-001	Rev: A/B	
Date: Wednesday, January 10, 2007	Sheet: 2	of: 4	

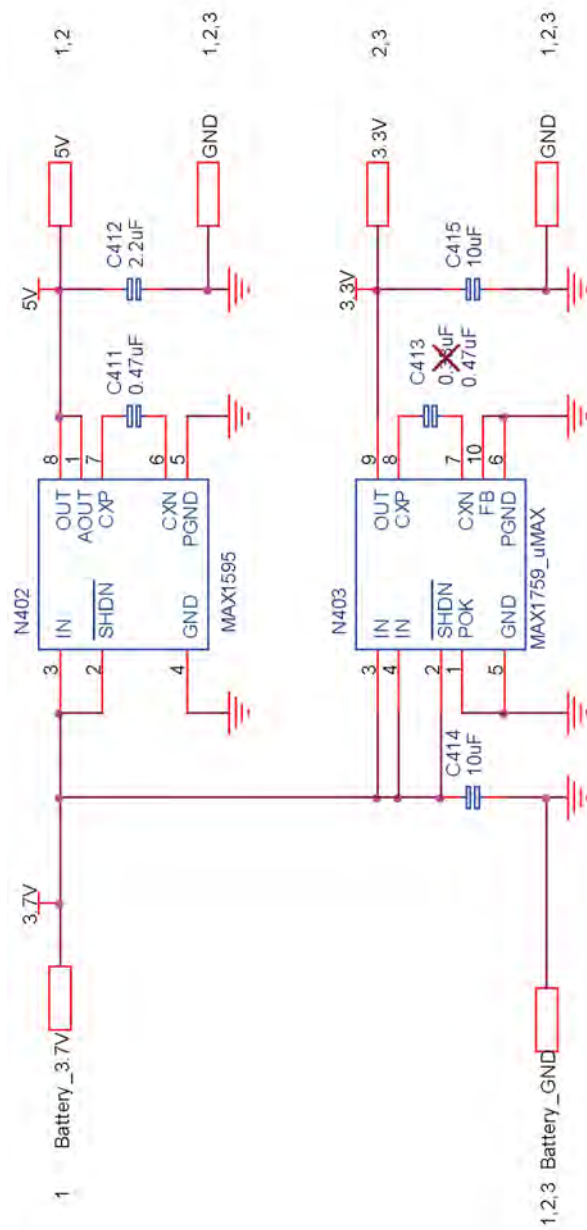



Figure G.8: Power supply

 DEMCON Twente B.V. Jos Ansink Zutphenstraat 25 7575 EJ OLDENZAAL The Netherlands			
Title		Power supply	
Size	Document number	Rev	
A4	0190-68-001	A/B	
Date:	Tuesday, January 09, 2007	Sheet	4 of 4

Bibliography

- Ansink, J. (2007), Electronic design for a gas pipe inspection robot.
- B. Sikken (2006), LPCUSB, an USB device driver for LPC microcontrollers.
<http://sourceforge.net/projects/lpcusb/>
- Burkink, B. (2009), PIRATE Propulsion Unit.
- Compu Phase (2009), Termite: a simple RS232 terminal.
http://www.compuphase.com/software_termite.htm/
- Dertien, E. (2006), System specifications for PIRATE.
- Dertien, E. (2007), Design of PIRATE Mainboard.
- Drost, E. (2009), *Measurement system for pipe profiling*, Master's thesis, University of Twente.
- K. Tuck (2007), Tilt sensing using linear accelerometers.
http://cache.freescale.com/files/sensors/doc/app_note/AN3461.pdf/
- M. Thomas (2006), WinARM toolchain to develop software for the ARM-family of controllers/processors on MS-Windows-hosts, Version 20060606.
http://gandalf.arubi.uni-kl.de/avr_projects/arm_projects/
- Mennink, T. (2010), *Self Localization of PIRATE Inside a Partially Structured Environment*, Master's thesis, University of Twente.
- Nordic Semiconductor (2007), nRF24L01 Single Chip 2.4GHz Transceiver, (Product Specification Revision 2.0).
<http://www.nordicsemi.com/>
- Reemeijer, H. (2010), *Control of a pipe inspection robot*, Master's thesis, University of Twente.
- Vennegoor op Nijhuis, J. (2007), *Development of a pipe inspection robot*, Master's thesis, University of Twente.