

Control of Quadcopters for Collaborative Interaction

C. (Cees) Trouwborst
BSc Report

Committee:
Prof. dr. ir. S. Stramigioli
M. Fumagalli PhD
Dr.ir. R.G.K.M. Aarts

Juli 2014

Report nr. 006RAM2014
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Summary

Unmanned Aerial Vehicles have the potential to perform tasks that are normally dangerous or costly to be performed by humans. In this bachelor thesis the Parrot AR.Drone was analysed and utilised for performing a collaborative task containing interaction with the environment: moving a mass using a set of quadcopters.

The Parrot AR.Drone is a commercially available quadcopter, intended for recreational use. Control of the quadcopter is normally done through a smartphone or tablet. In this project, the Robot Operating System was used as a framework to implement new functionality.

As a starting point multiple ways of teleoperation were implemented, enabling the user to directly send velocity commands to the quadcopters. Then, setpoint control was implemented making use of the position feedback of the Natural Point OptiTrack system. The performance of the controller was adequate for the position of one quadcopter. When flying multiple quadcopters indoors at the same time the negative effects of the additional airflow could clearly be seen, resulting in larger deviations from the setpoint. The influence of the size of the room was another observation made related to aerodynamics.

This controller architecture was used to explore the possibilities of the Parrot AR.Drone to perform interactive tasks in a collaborative way. The scenario of two quadcopters transporting a mass was chosen for experiments. Semi-automated picking up and moving of a mass connected through cables was successfully performed at the end of the project. The internal integrating action of the Parrot AR.Drone rendered it unnecessary to implement additional controller actions.

It can be concluded that the Parrot AR.Drone is an easy to use and widely available research platform. Since the internal control software is not open and not fully documented, the platform is not useful for all research questions regarding quadcopters. As example: connecting two quadcopters through a rotational joint was not possible because unpredictable internal logic.

Multiple executables have been developed during this project. Two of the most important applications (the controller and an application automating the connection sequence) feature a graphical user interface to enable intuitive control. These applications are connected in a modular way, so that other UAVs or position feedback systems can easily be implemented.

Contents

1	Introduction	1
2	Background	2
2.1	Modelling of Quadcopters	2
2.2	Connection between UAVs and loads	3
2.3	Control Theory	3
2.4	The Robot Operating System	4
2.5	Natural Point OptiTrack	4
3	Realizing position control of the Parrot AR.Drone	5
3.1	The Parrot AR.Drone	5
3.2	Closing the control loop	5
3.3	Control interfaces	11
3.4	Intermediary conclusions	14
4	Position control of multiple quadcopters	15
4.1	Interfacing with multiple drones	15
4.2	Setpoint Stability	17
4.3	Flying in patterns	17
4.4	Intermediate conclusion	17
5	Moving a load	18
5.1	Possible approaches to picking up and moving a load	18
5.2	Connecting through cables	18
5.3	Connected quadcopters	22
5.4	Intermediary conclusions	25
6	Discussion	26
6.1	Flying robots	26
6.2	Practical nature	26
7	Conclusions and recommendations	27
A	Appendix 1 - Documentation for further use	28
	Bibliography	29

1 Introduction

Unmanned Aerial Vehicles have large potential for performing tasks that are dangerous or very costly for humans. Examples are the inspection of high structures, humanitarian purposes or search-and-rescue missions. One specific type of UAV is becoming increasingly more popular lately: the quadcopter (Figure 1.1). When visiting large events or parties, professional quadcopters can be seen that are used to capture video for promotional or surveillance purposes. Recreational use is increasing as well: for less than 50 Euros a small remote controlled quadcopter can be bought to fly around in your living room or garden. In these situations the quadcopter is usually in free flight. There is no physical contact between the surroundings and the quadcopter and no cooperation between the quadcopters

If UAVs would have the capabilities to collaborate the number of possibilities grows even further. For example, a group of UAVs would be able to efficiently and autonomously search a missing person in a large area by sharing data between. Or, the combined load capacity of a group of quadcopters can be used to deliver medicine in remote areas.

This bachelor thesis focuses on the use of a commercially available quadcopter platform, the Parrot AR.Drone, to perform a task that requires physical collaboration and interaction: moving a mass. In this way a clear interaction between the quadcopters and their surroundings is present. As preliminary step towards the view of collaborating aerial robots the choice was made to perform this task in an indoor scenario where position feedback is present. Starting off with position control, additional controller logic can be implemented to counteract the forces imposed by a mass connected to the quadcopter.

At the end of this research a group of Parrot A.R.Drones should be able to pick up, move and place a predefined mass. While the choice is made for the Parrot AR.Drone, a generalized approach is chosen where possible to encourage reuse of this research's outcome and deliverables.



Figure 1.1: Examples of recreational quadcopters

2 Background

This chapter introduces some of the main concepts and background knowledge related to this project. A generic model of a quadcopter will be introduced, as well as methods of connecting masses to UAVs and an introduction to controller actions. Some specific tools used in this project, like the Robot Operating System or NaturalPoint OptiTrack motion capture system, are shortly introduced as well.

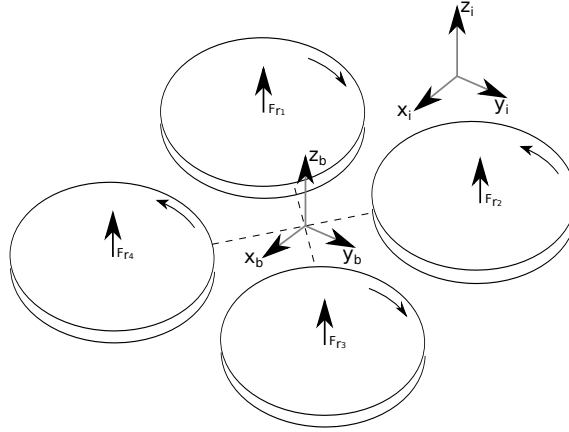


Figure 2.1: Schematic view of a quadcopter

2.1 Modelling of Quadcopters

A quadcopter is a type of (usually unmanned) aerial vehicle using four rotors for its actuation. A simplified quadcopter is shown in Figure 2.1. Four rotors are used to move the quadcopter in six degrees of freedom (three translations and three rotations) and can be considered inputs. The relation between the force from the propellers and the thrust and net torques around the center of mass is given in Equation 2.1. Those four inputs can be used to generate a net torque around the center of mass of the quadcopter in every direction, and a total thrust vector (adapted from [Fumagalli et al. (2012)]).

$$\begin{bmatrix} F_{zb} \\ M_{xb} \\ M_{yb} \\ M_{zb} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -d & d & d & -d \\ d & d & -d & -d \\ -c & c & c & -c \end{bmatrix} \begin{bmatrix} F_{r1} \\ F_{r2} \\ F_{r3} \\ F_{r4} \end{bmatrix} \quad (2.1)$$

where F_{zb} stands for the total upwards force, d is the distance from the center of mass along the axes until perpendicular to the motors, c is the constant ratio between rotor thrust and reaction torque, and M_{ib} are the torques around the body fixed axis indicated by the index. Equation 2.2 shows the dynamics relations for the rotations when the LHS of Equation 2.1 is considered the system input.

$$\begin{bmatrix} J_x \ddot{\phi} \\ J_y \ddot{\theta} \\ J_z \ddot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} M_{xb} \\ M_{yb} \\ M_{zb} \end{bmatrix} \quad (2.2)$$

J_i is the rotational moment of inertia around the axis indicated by the index, ϕ, θ and ψ are the positive rotations around x, y and z respectively. For the translations, the following dynamic relations can be found.

$$m \begin{bmatrix} \ddot{x}_i \\ \ddot{y}_i \\ \ddot{z}_i \end{bmatrix} = R_3(\phi, \theta, \psi) \begin{bmatrix} 0 \\ 0 \\ F_{zb} \end{bmatrix} - mg \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (2.3)$$

with m the mass of the quadcopter, g the gravitational acceleration and $R_3(\phi, \theta, \psi)$ the 3D rotation matrix as function of the current attitude. From Equation 2.3 can be concluded that a quadcopter is an underactuated aerial vehicle. Through the four inputs only four of the degrees of freedom can be manipulated directly. As example: to get an acceleration in the positive x-direction, the quadcopter should change its rotation around the y-axis. In order to obtain this attitude change the rotors on one side of the quadcopter should spin at a higher speed than the ones at the other side of the quadcopter.

This coupling of degrees of freedom immediately shows that the transfer function between torque and position in the plane of the quadcopter will be non-linear and of the fourth order. The torque has to be integrated over twice to get to an attitude. This attitude relates to a net force in a translational direction through the (non-linear) rotation matrix. One has to integrate twice over this force to get to position, resulting in a non-linear, fourth-order model. [Hernandez et al. (2013)]

2.2 Connection between UAVs and loads

In this research a specific type of UAV is connected to a load. For other types of UAVs this has been done before [Bernard et al. (2011)][Maza et al. (2010)]. To prevent confusion about the exact approach chosen in this research, a categorization can be made in the way the mass is connected to the UAV. [Parra-Vega et al. (2012)] gives definitions for the methods widely used. In this research a combination of *Load Carrying with Cables* and *Ground Grabbing* is used. Other well-known methods are *Aerial Manipulation*, where the pose of the mass is passively controlled in the air and *Load Carrying Rigidly*, where the mass is connected to the UAV rigidly from the take-off.

2.3 Control Theory

To be able to pick-up and move a mass in a semi-autonomous way, position control has to be implemented. In that way, the quadcopter can move to desired locations based on setpoints (coordinates). The control algorithms used are classical feedback controllers with a number of possible actions working on the error between the reference and feedback signal. This setup is schematically depicted in Figure 2.2. The system to be controlled is usually called the *plant*. A full description of this class of controllers can be found in specific literature on control theory. [Franklin et al. (2010)]

Actions that are considered are proportional, integral and derivative action (resulting in the well-known abbreviation 'PID-controllers'). Each of these actions has its purposes. Proportional action is the most basic action: as long as there is a non-zero error, contribute to actuation in that direction. Derivative action counteracts large overshoot by looking at the derivative of the error: if the error becomes smaller, contribute to the actuation for the opposite direction. Integral action sums over the error, increasing the actuation over time if the error is non-zero. This enables the controller to counteract position dependent forces (e.g. a spring) that result in steady-state errors.

Each of the actions has a corresponding gain. The optimal gains for the system are dependent on the requirements and the system itself. When a full model of the system is available, the gains can be found through iteration in simulation. In practice a combination of simulation and experiments (trial and error) is used for tuning.

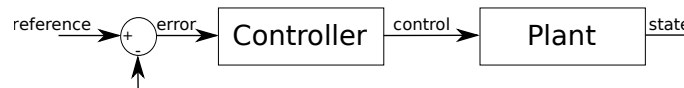
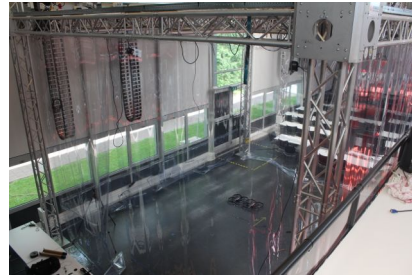


Figure 2.2: Schematic view of a feedback controller



(a) Flying Arena at Robotics and Mechatronics



(b) SmartXP Lab

Figure 2.3: Locations with position feedback

2.4 The Robot Operating System

In this project the Robot Operating System (ROS) is used as software framework for all code used in the control of the quadcopters.

ROS is a flexible framework for writing robot software. It is a collection of tools, libraries and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. [ROS.org (2014)]

ROS is designed to be language independent. Python and C++ are widely used to write *packages*. Packages are used to group executables, libraries, configuration files, etc. together. Executables can be *nodes*: processes in the ROS infrastructure responsible for communication and calculation. Communication works over *topics*: communication lines where nodes can subscribe or publish to. Besides topics there are also *services*, that use a request / reply structure.

ROS is actively used and developed by a easy-to-reach community. The primary source of information is ros.org, including a dedicated wiki and discussion board (ROS Answers). ROS is specifically designed to be used under Ubuntu Linux.

For this specific project the choice for ROS was made for three reasons:

1. ROS enables rapid development of new software for robotics by supplying the user with numerous tools and libraries.
2. ROS encourages a modular structure and therefore sharing and reusing code.
3. ROS is used by a large number of people (also within the Robotics and Mechatronics department), so a lot of information is readily available and questions are easily asked.

2.5 Natural Point OptiTrack

Part of the definition of the assignment is the use of quadcopters in an indoor scenario, where position information is present. At the University of Twente there are two places where this holds: the flying arena at the Robotics and Mechatronics Lab and the SmartXP Lab (Figure 2.3). In both area's an OptiTrack motion capture system is present, manufactured by Natural Point. [NaturalPoint (2014)]

This system uses an array of high-speed infrared cameras to track objects within the tracked volume. Each of the cameras has a ring of infrared LEDs in its body to sufficiently illuminate the objects. On the objects passive infrared reflectors (markers) are placed. The reflectors can be tracked by looking at what points clear infrared reflections are visible.

3 Realizing position control of the Parrot AR.Drone

The first step in getting a group of quadcopters to perform a useful task is to realize position control of the specific type quadcopter used in this project. The approach used for the control of one UAV can be extended for the use of multiple drones afterwards.



Figure 3.1: The used quadcopter

3.1 The Parrot AR.Drone

The AR.Drone V2 quadcopter (Figure 3.1) used in this project is manufactured by the French company Parrot [Parrot (2014)]. They intend to create a large group of potential users by making the quadcopter controllable from smartphones and tablets. The calculation power of embedded platforms, both in the quadcopter as well as in the controlling device, enabled Parrot to make a quadcopter that is easy to fly, even by inexperienced people. To do this, a large number of sensors is built-in in the body of the AR.Drone: a 3-axis gyroscope, a 3-axis accelerometer, a 3-axis magnetometer, a pressure sensor, an ultrasound sensor and two cameras (one HD camera pointing forwards and one QVGA camera pointing downwards to use for optical flow measurements relative to the floor).

Parrot supplies a Software Development Kit for developers interested in writing applications that use the AR.Drone [AR.DroneSDK (2014)]. An implementation of this SDK for ROS was developed in the Autonomy Lab of the Simon Fraser University. [Monajjemi (2014)]

As input the SDK (and its ROS implementation) expects a combination of attitude (desired angles) and a vertical speed, therefore the controller does not have to (and cannot) control the individual motor speeds. The embedded electronics of the AR.Drone contain a control loop that translates the commands to motor speeds. When it is assumed that this embedded inner loop is significantly faster than the outer loop, the planar motion of the system can be seen as a dominantly second-order system.

Later in the process, this assumption is validated by giving in a new setpoint and looking at the response of the attitude and position to the given commands (Figure 3.2). The resemblance between the control signal and attitude response can be seen (inversed and scaled), indicating a very low delay between giving the command and reaching the attitude reference¹.

3.2 Closing the control loop

As input the plant expects an attitude reference and desired vertical speed, based on the error between the current position and the desired setpoint. To close the loop the position inform-

¹There are differences between the two signals also. The relatively low peak around 3 seconds, for example. This might be due to an integrating action in the internal loop, as described in Section 5.2.1.

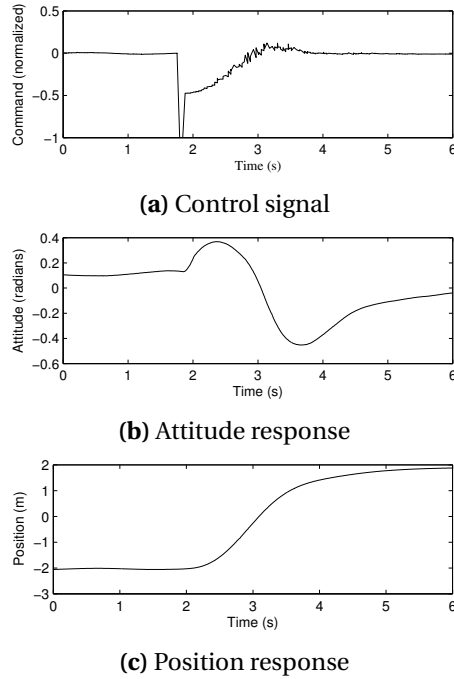


Figure 3.2: Comparison between inner and outer loop speed.

ation of the plant should therefore be measured and fed back into the controller, to be able to calculate the error.

3.2.1 Getting state information

The Natural Point software used with the OptiTrack system (Tracking Tools or Motive) is able to stream the position information of the tracked items over a network connection to other computers in the network using a UDP multicast message. This information is published as a *pose*-type message on specified ROS topics by a package developed by the Canada-based company Clearpath Robotics. [Purvis (2014)]

To track items within the software the quadcopter should be marked with infrared passive markers in a non-symmetric way (to accurately keep track of rotations as well). Every *trackable* in the OptiTrack software has a unique identifier, corresponding to a set of markers placed on the trackable's body (e.g. Figure 3.3).



Figure 3.3: The hull of the quadcopter uniquely marked

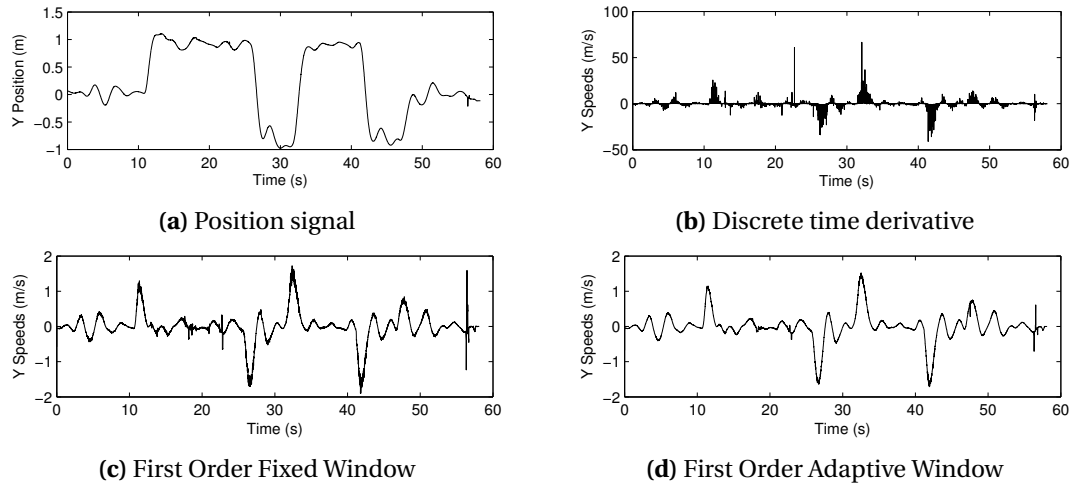


Figure 3.4: Comparison of derivative methods

3.2.2 Control algorithm

A classical PD controller was implemented in C++. Since there are no position dependent forces present in the system during free flight² an integral action is omitted (Section 2.3). The derivative action however is very important, since the low friction of the quadcopter easily results in overshoot.

The position information (and therefore the error signal) from the OptiTrack system is very stable and accurate (deviations smaller than 0.001 meter) no filtering or smoothing was applied. For the derivative of the error signal multiple approaches were tried. Naturally, smooth algebraic solutions were not available since a discrete time system is used.

1. **Standard discrete-time derivative:** The difference between the current sample and the previous sample divided by the time between them. This approach is used with and without an additional low-pass filter.
2. **First Order Fixed Window approach:** The difference between the current sample and a previous sample, a fixed number of steps back, divided by the time between them.
3. **First Order Adaptive Window approach:** A method proposed by [Janabi-Sharifi et al. (2000)] to find a smooth derivative in discrete systems. Equal to the First Order Fixed Window approach, except that the size of the window is dynamically adapted to the aggressiveness of the signal.

A comparison of these methods applied to a position signal can be seen in Figure 3.4. Analysis of these signals reveals that the standard deviations of the fixed window approach and adaptive window approach differ under 2%, indicating a high resemblance³. The choice was made to use the fixed window approach to reduce unnecessary complexity. The size of the window is ten samples, corresponding to approximately 0.1 seconds.

The description of the task ("picking up and moving a mass") does not lead to very accurate specifications for the controller performance. Keeping in mind that a pick-up has to take place in the end, it was determined that the controller should be able to get the quadcopter positioned within a circle with approximately 0.05 meter radius. Besides that, it should be possible

²Technically, gravity is present and an position dependent force. Since the input signal of the plant is a vertical speed however, realized by the on-board controller, an integral action is not necessary in the Z-direction.

³For the sake of completeness: the standard deviations of the adaptive window and the discrete time derivative differ 620%.

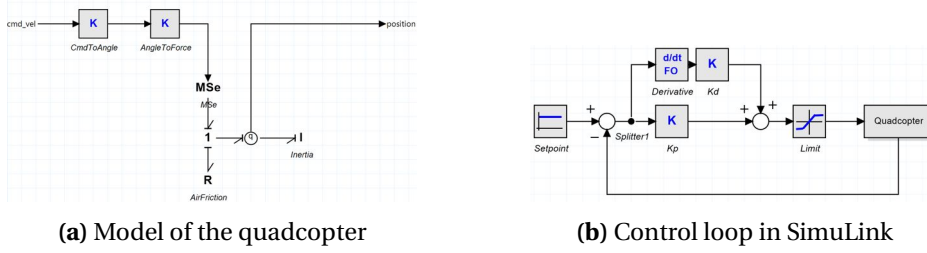


Figure 3.5: 1D 20-Sim for lateral motion

to damp out overshoot if the task at hand requires such. The tuning of the PD controller was done in two steps: a 20-Sim model (Figure 3.5) was created of the lateral motion in 1D to simulate the moving mass of the quadcopter. In this model, the assumption is made that (normalized) attitude commands are directly and in a linear fashion related to the force in lateral direction. The following system parameters are relevant:

- Quadcopter mass. Given by Parrot to be 420 grams when the hull is attached.
- Air friction coefficient. Air friction is modelled to be velocity dependent. Since this is a large simplification of the actual behaviour of the system, no values were found in literature. A constant coefficient of 0.1 Ns/m was estimated.
- Ratio between command and angle. This value is one of the settings of the AR.Drone SDK and is set to its maximum: 0.52 radians (approximately 30 degrees).
- Ratio between the angle and exerted force. This value is determined experimentally by measuring both attitude and accelerations over a certain time. Since the mass is known, an estimate can be given for this ratio. A ratio of 4.116 N per radian was used.

The linearised transfer function from normalized command to position (Equation 3.1) was then used as input for the Matlab PID tool to get to an initial set of controller parameters ($k_p = 0.05$ and $k_d = 0.13$). The response of this system in simulations can be seen in Figure 3.6. This set of parameters was used as starting point for further experimental tuning.

$$\text{position} = \frac{5.096}{s^2 + 0.2381s} \text{command} \quad (3.1)$$

For validation the control parameters used in majority of the experiments were tested in the simulation as well. The response of the system can be seen in Figure 3.7. When comparing this figure with an actual response (Figure 3.9) a large resemblance can be seen. Reasons for the deviations can be both the system parameters as well as the simplifications made in the model. Since the model was only used for initial tuning (successfully), no further effort was spent on expanding the model.

The inner control loop of the Parrot AR.Drone has functionality implemented to hover in one position. Sensor data from the gyroscope, magnetometer and accelerometer is combined with

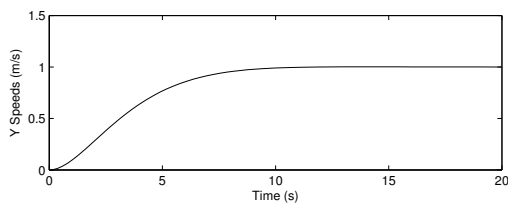


Figure 3.6: Controller performance in simulation (initial tuning)

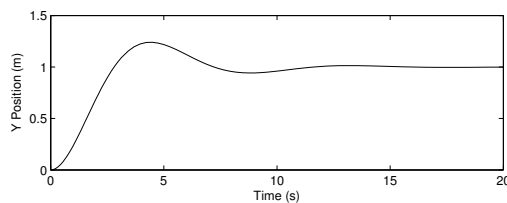


Figure 3.7: Controller performance in simulation (tuned as in experiments)

an optical flow measurement [Barron et al. (1994)] from the bottom camera. [AR.DroneSDK (2014)] This approach should reduce the drift that normally occurs when using only gyroscopes, magnetometers and accelerometers. A function was implemented to enable this on-board hover mode when the quadcopter was within a specified distance from the setpoint. The comparison between the on-board hover function and hovering using the implemented PD control can be seen in the next section on controller performance.

Another optional feature of the controller is the implementation of absolute velocity damping. This feature was implemented to reduce overshoot by counteracting large velocities. The aforementioned First Order Fixed Window approach was used to get the velocity information. Again, the influence of this velocity damping on performance can be seen in the next section.

3.2.3 Controller performance

Setpoint behaviour

To test the performance of the controller an experiment is performed where a single setpoint is given (Figure 3.8). It can clearly be seen that the used parameters result in a slightly over-damped system. The performance is well within the desired range.

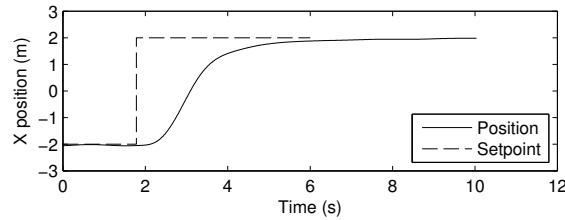


Figure 3.8: Single setpoint performance in experiment

For a next experiment, a series of setpoints is given, positioned in a square around the origin of the coordinate system. The gains of the controller are slightly altered to make the system faster, while allowing some overshoot. Both setpoints and response are given in Figure 3.9.

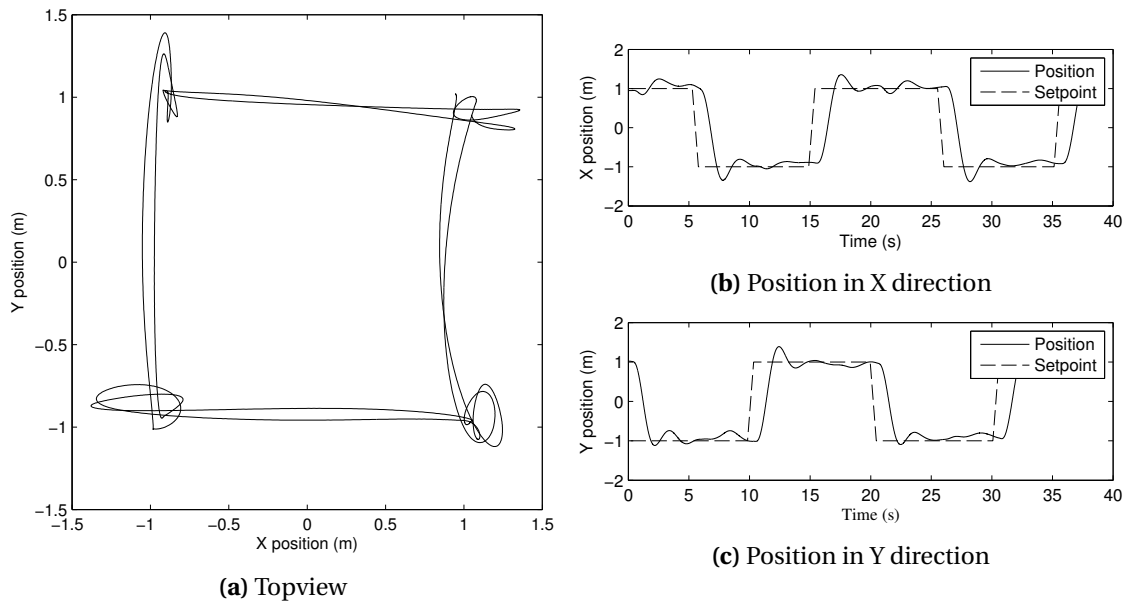


Figure 3.9: Series of setpoints

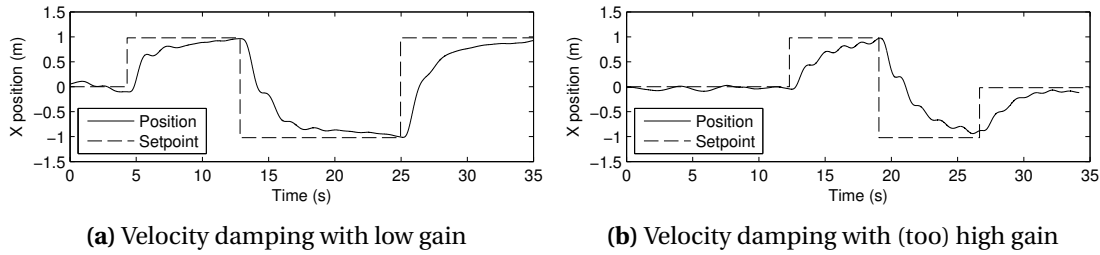


Figure 3.10: Velocity damping performance

Velocity damping

The velocity damping was implemented to lower overshoot in the system and increase stability. The D action in the PD controller has the same function in a specific direction towards a specific setpoint. Absolute velocity damping implements this behaviour in general. The results of the implementation can be seen in Figure 3.10.

The influence of the damping can easily be seen: the system becomes slower (as is expected). What really catches the attention when seeing this experiment is the "wiggly" motion of the quadcopter. This effect is amplified when the gain is increased. This motion is expected to be the result of a small delay, resulting in continuous overcompensation. Since this additional controller logic does not greatly enhance the performance and is not required for operation, velocity damping is disabled in other experiments.

Performance while hovering

To compare the performance of the internal hovermode and hovering using the implemented PD controller, multiple experiments were performed. Since the bottom facing camera utilises an optical flow algorithm, the experiments were done twice: with and without textured floor (Figure 3.11).

The influence of the texture of the floor on the performance of the optical flow algorithm can easily be seen. When the camera can depict motion from a clear texture on the floor, it will counteract all movement, resulting in oscillations around the threshold (since above the threshold the PD controller kicks in). If the floor is non-textured, the internal hovering algorithm is not able to counteract drift efficiently and keeps drifting, until the PD control reverses its direction.

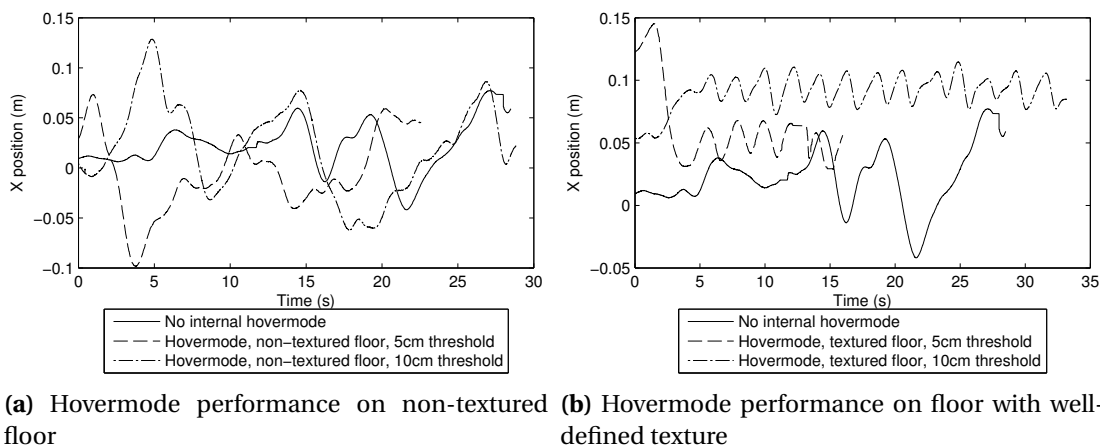


Figure 3.11: Hovermode performance comparison

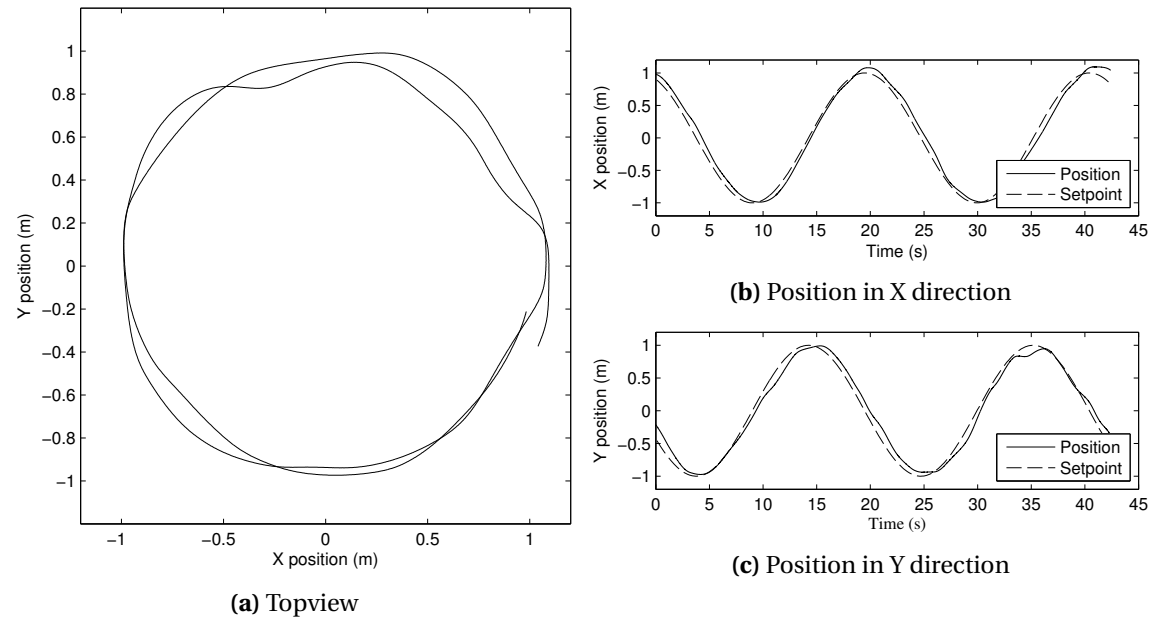


Figure 3.12: Reference tracking.

It can be concluded that the internal hovermode is, in some situations, a useful tool. Especially on very textured floors the algorithm works well and the implementation can be tuned to get stable and predictable behaviour around the setpoint. However, this would add demands for both the surroundings and the used quadcopter platform. As mentioned before, the goal is to create software that can easily be used on other platforms as well. Since the use of the internal hovermode is not a necessity, this controller option is switched off in further experiments.

Reference tracking

Utilizing the interfacing methods with Matlab an experiment was performed with a setpoint dependent on time. For both lateral components of the setpoint a periodic function was used, resulting in a circular motion with a radius of 1 meter. It can be seen that the reference tracking performance is within specifications, especially when the small delay in response is neglected (Figure 3.12).

3.3 Control interfaces

There are two ways the quadcopters can be controlled through ROS: by directly sending velocity commands and by sending setpoints. Using a smartphone for the control of the drone, as intended by Parrot, is an example of directly sending velocity commands. Using the position controller, only a setpoint is given in by the user. In both cases interfaces need to be designed to enable interaction between the end user and the software.

3.3.1 Directly sending velocity commands

Keyboard control

As a first attempt to control the quadcopter through ROS a keyboard interface was developed in C++. This ROS node changes the behaviour of a standard Ubuntu terminal to directly read the input from the keyboard per key touch and interprets the keys as commands for the drone. While this is a nice way to test the quadcopter, this implementation of keyboard control has two fundamental problems.

- Only one character can be interpreted at a moment in time, therefore smooth movement in diagonal directions is not possible.
- Holding a key results in unwanted behaviour. When you hold a key, you expect that the command to the quadcopter will be kept constant as well. In practice this does not work, because of the Auto Key Repeat Delay implemented in the BIOS of a computer.

These problems can be solved by using libraries that directly read the keyboard output instead of the characters from the terminal, or using another input device. The latter option was chosen, since other input devices are a better fit for this task.

Joystick / gamepad control

Using C++ a ROS node was developed to interpret the commands from a joystick or game controller. ROS implements a joystick driver for generic Linux joystick input devices. The main task of the implemented node is to translate input from specific axes and buttons to commands towards the quadcopter. Two devices were used for testing: a Logitech joystick and a Speedlink PS3 controller clone.

3.3.2 Position control

When using the position controller, the reference signal (the setpoint) can be considered as the input from the user. Multiple interfaces were implemented to input this reference information.

From ROS

The controller node is subscribed to a ROS topic where a setpoint can be published to. The controller node will remember the latest setpoint given and navigate towards it. The message type of the setpoint topic is *pose*. This results in the ability to set a setpoint directly through a command line interface in the following way:

```
rostopic pub -1 /setpoint geometry_msgs/Pose -- '[1.0, 2.0, 3.0]'
          '[0, 0, 0, 1]'
```

In this command */setpoint* is the topic to publish the command to, *geometry_msgs/Pose* refers to the type to be used and the arrays of numbers depict a setpoint in $x = 1$, $y = 2$, $z = 3$ while maintaining a zero yaw⁴

While this works and gives insight in the controller structure, a more user-friendly way of inputting new setpoints was desired. For that reason a user interface was developed using Python and Glade. This user interface not only enables users to set the current setpoint, but also implements buttons for the take-off, landing and resetting of the drone. Additional buttons were added through the process to be able to configure control parameters while flying. A screenshot of the control interface can be seen in Figure 3.13.

Matlab / Simulink

Both Matlab and its simulation toolkit SimuLink are tools that are widely used within the University of Twente and many other universities and companies. Being able to control the quadcopter from Matlab by publishing setpoints can lead to more projects using the quadcopter platform. In the beginning of 2014 MathWorks (the company behind Matlab and Simulink) released an I/O package to communicate between Matlab and ROS, making it very easy to write ROS nodes in Matlab. Internally the I/O package uses the Java implementation of ROS to integrate with the ROS infrastructure.

⁴A quaternion representation of the rotation is used in ROS.



Figure 3.13: Graphical User Interface of Controller

The reference tracking performance from Section 3.2.3 used a Matlab script to generate and publish the setpoints, using just the following lines of code.

```
% Core and topic information
core = 'http://ramflight-arena:11311';
topic = 'John/setpoint';

% Initialize node, publisher and message
node = rosmatlab.node('matlab_circle',core);
pose_pub = rosmatlab.publisher(topic, 'geometry_msgs/Pose', node);
pose = rosmatlab.message('geometry_msgs/Pose',node);

% Init timer
tic;

% Paramers
r = 1; % Setpoint coord.
w = 3; % wait in the beginning
z = 1.5; % Setpoint z

% Prepare plot
figure;

while true
    pose.getPosition().setX(0);
    pose.getPosition().setY(0);
    if toc() > w
        pose.getPosition().setX(cos(-toc()*(0.3))*r);
        pose.getPosition().setY(sin(-toc()*(0.3))*r);
    end
    pose.getPosition().setZ(z);
    pose.getOrientation().setW(1);

    % Live update of setpoint radius
    plot(pose.getPosition().getX(),pose.getPosition().getY(),'rd');
    axis([-1.5 1.5 -1.5 1.5]);

    % Pub frequency approx. 10 hz.
    pause(0.1);
    pose_pub.publish(pose);
end
```

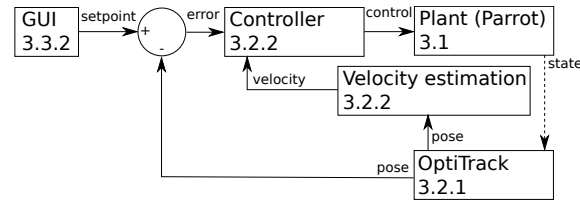


Figure 3.14: Schematic overview of controller architecture

3.4 Intermediary conclusions

Using a custom-made PD controller position control through ROS for one Parrot AR.Drone was successfully implemented. The performance of the controller is adequate for the task at hand (moving a mass, requiring an estimated precision of ± 0.05 meter). Reflecting on Section 2.3 the control architecture is depicted schematically in Figure 3.14, including references to the relevant sections of this chapter.

Users can control the drone through directly sending velocity commands or giving in setpoints. For direct control the keyboard or a joystick can be used. For giving in setpoints the custom graphical user interface is recommended, since all control parameters can be altered through this application as well.

4 Position control of multiple quadcopters

With control of one Parrot AR.Drone working and tested thoroughly, the next step is to scale up to the use of multiple drones. This chapter describes the required changes to the network infrastructure and the consequences for the control of the performance.

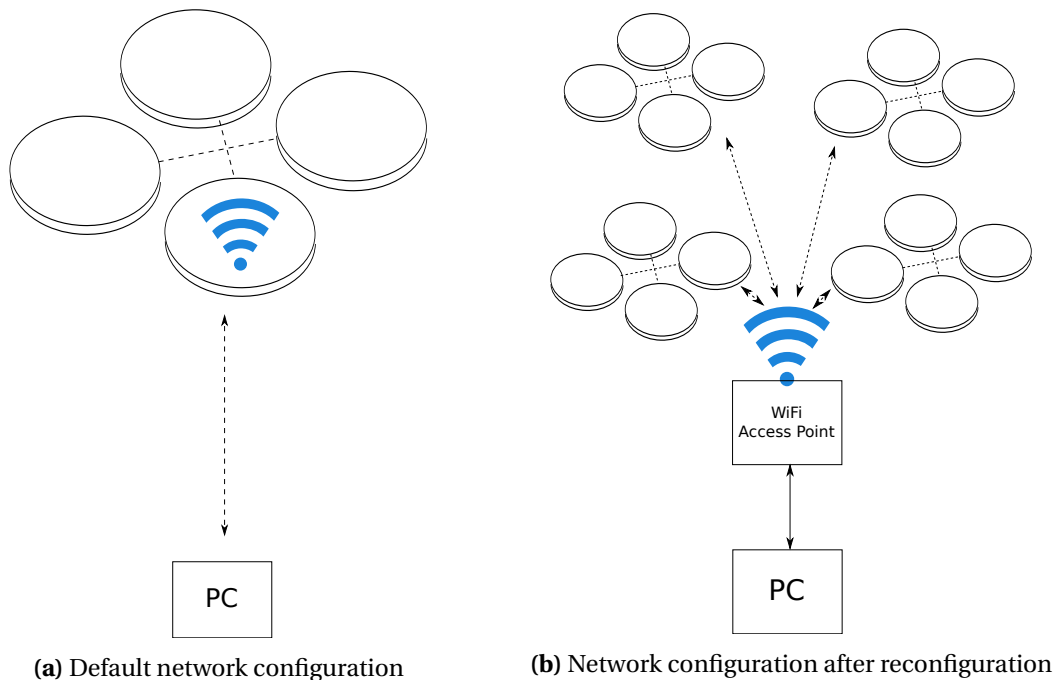


Figure 4.1: Network infrastructure

4.1 Interfacing with multiple drones

The way the user usually connects to a Parrot AR.Drone is by connecting directly to the Wi-Fi network of the drone. When using the on-board access point of the drone the user's computer or telephone will get an IP address automatically using DHCP and will be able to reach the drone functioning as gateway on IP address 192.168.1.1. When using multiple drones this approach is impossible¹. In order to use multiple drones simultaneously one can however use *telnet* to make the drone connect to a specified WLAN network using a fixed IP address, by running a batch script on the Linux-based embedded software of the Parrot AR.Drone. This changes the network structure, as shown in Figure 4.1. This is a task that has to be performed every time the drone is powered on. Other implementations are also possible, but using this approach reduces the risk of doing irreversible damage and keeps the original behaviour intact.

Through the dedicated network it is possible to connect to all quadcopters. Within ROS however a structure has to be implemented to make sure drones can be separated from each other. This can be done by using *namespaces*. Every namespace contains a quadcopter driver, a controller node and other nodes required for the functioning of the quadcopter.

The task of configuring the drones to connect to the network and implementing the namespaces is to be performed every time the quadcopters are switched on, or when the amount of the drones that is flying changes. For this reason an application was written in Python (including Glade user interface) to perform the following tasks:

¹This holds when using only one network interface. In theory, one can use multiple network interfaces and give all drones different static IP addresses and still connect directly.

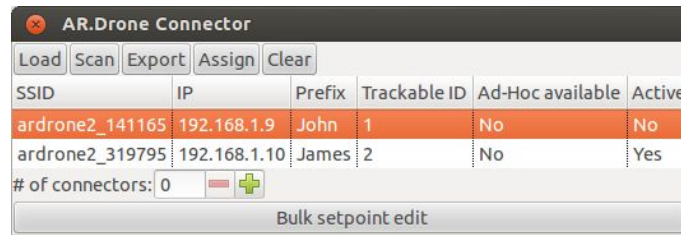


Figure 4.2: Graphical User Interface of Connector

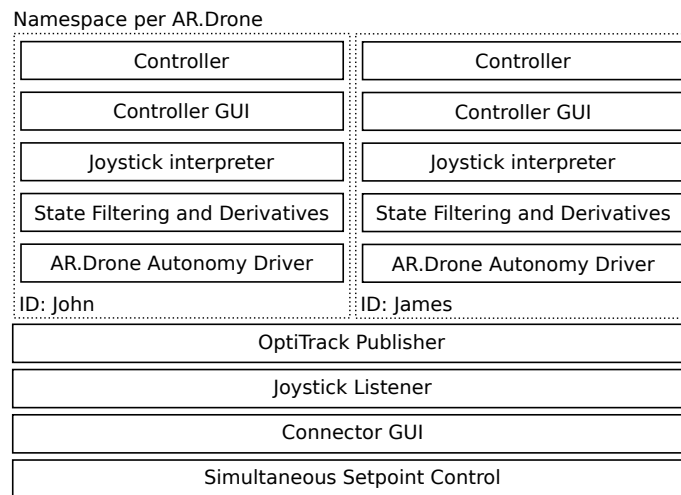


Figure 4.3: Schematic overview of the system

1. Scan for available Parrot AR.Drones (based on SSID).
2. Assign a name (namespace prefix) to each detected drone.
3. Assign a trackable ID to each detected drone.
4. Use telnet to configure the quadcopters to connect to the dedicated network.
5. Configure and start the OptiTrack package using the trackable IDs from user input.
6. Start a joystick node to be used during tele-operation.
7. Export known drones with their name and trackable ID for reuse.
8. Import known drones.
9. On double-click: start a controller user interface.

The user interface can be seen in Figure 4.2. An overview of the total system is schematically depicted in Figure 4.3.

The original AR.Drone SDK was not intended for the use of multiple quadcopters at the same time. The UDP ports specified for communication with the quadcopter can only be used by one driver at the same time, resulting in conflicts when multiple drones were used. A developer from the University of Georgia was able to change the AR.Drone SDK to fix this problem, by iterating over available ports.

The importance of a stable and fast wireless connection cannot be overstressed. If connection is lost during flight, the quadcopter will fly around without the ability to be landed or reset. The only way of stopping the quadcopter is waiting for a crash.

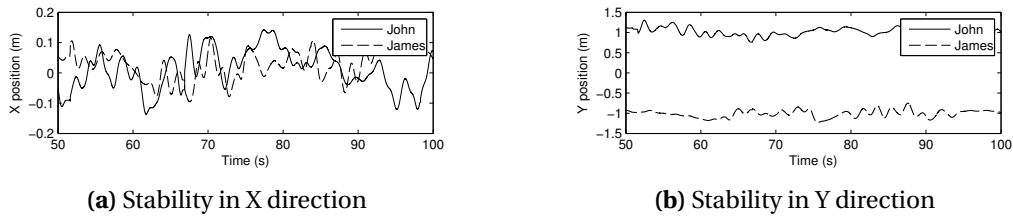


Figure 4.4: Setpoint performance with multiple drones

4.2 Setpoint Stability

The use of multiple setpoints is tested by using two quadcopters connected through the aforementioned dedicated wireless network. From Figure 4.4 can be concluded that both drones reach their setpoints, but that the airflow results in less stable behaviour compared to single drone operation: an average standard deviation of 0.0723 can be observed, opposed to 0.0241 (disabled hovermode signal from Figure 3.11).

4.3 Flying in patterns

Using the Matlab script to publish setpoints to two specific drones a pattern can be followed. In Figure 4.5 the behaviour over time is shown of two drones flying in a circle. When comparing this to Figure 3.12 the same conclusion can be drawn: flying multiple drones in proximity of each other has negative effects on the performance.

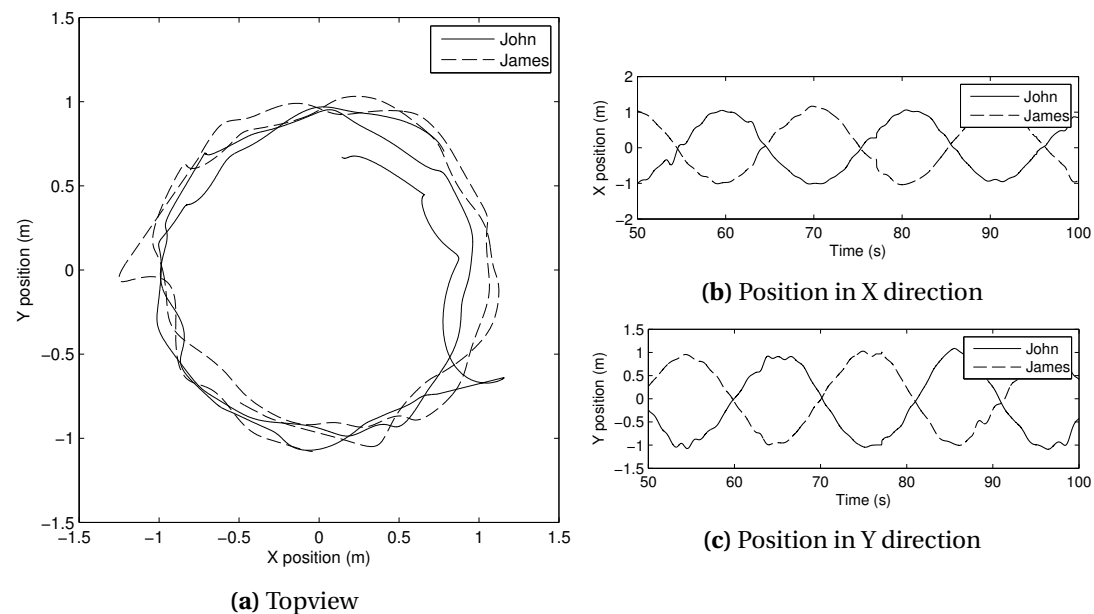


Figure 4.5: Reference tracking.

4.4 Intermediate conclusion

It is possible to position control multiple AR.Drones using multiple PD controllers through a dedicated WiFi network. The performance of the control however is less than when using only one drone. The additional airflow created by the drone results in larger deviations from the given reference signals.

Specialised software was developed to automate the process of connecting to multiple drones, to decrease the amount of time spent on repetitive tasks. The number of drones used is dynamic and can change during runtime, when other quadcopters are still flying.

5 Moving a load

With simultaneous setpoint control of multiple quadcopters in place the possibilities for performing tasks in a collaborative way can be explored. The specific task considered in this project is moving a load, with a focus on gaining knowledge on what is possible within the control architecture implemented in the previous chapter.

5.1 Possible approaches to picking up and moving a load

As can be read in the section on background information, there are multiple ways of moving a mass. Since the used quadcopter platform is essentially a closed platform, it is hard to extend with additional actuators. Active grasping of a mass is therefore not possible. When considering a group of AR.Drones instead of a single drone, a distinction can be made between three approaches:

1. Connection of every quadcopter to the mass through cables
2. Rigid or flexible connection of every quadcopter to the mass, essentially resulting in a new aerial vehicle
3. Rigid or flexible connection between quadcopters and one or multiple cables to the mass, essentially resulting in a new aerial vehicle with cable-attached mass

When a requirement is to perform a aerial or ground pick-up, two of the three options remain: option one, picking up ends of the cables connected to a mass per quadcopter, and option two, connecting multiple drones and let them perform a common pick-up. Those two options are considered in detail in the following sections.

5.2 Connecting through cables

By attaching a hook under the quadcopter (Figure 5.1) and connecting the mass through cables with a specifically designed pick-up connector (Figure 5.2), a set of quadcopters can perform a pick-up without using additional actuators.

When considering the movement of the mass underneath the quadcopters it can be stated intuitively that using at least three quadcopters positioned symmetrically around the mass will result in a very stable position of the mass, since the lateral translational degrees of freedom are limited. Dependent on the position of the cables on the mass, rotation can be blocked as well.

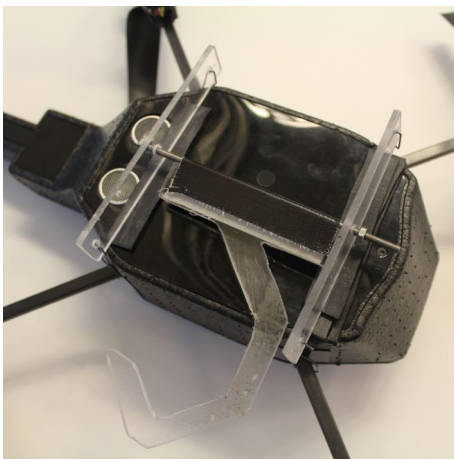


Figure 5.1: Hook attached under AR.Drone

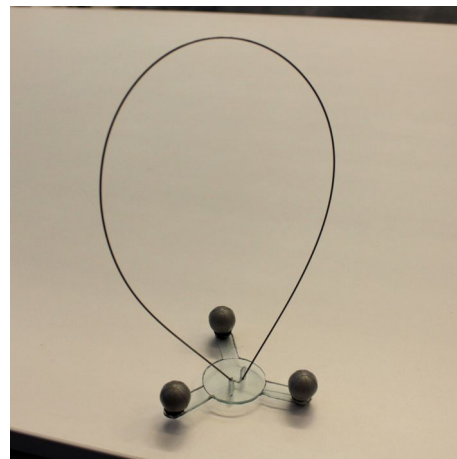


Figure 5.2: Connector with reflective markers

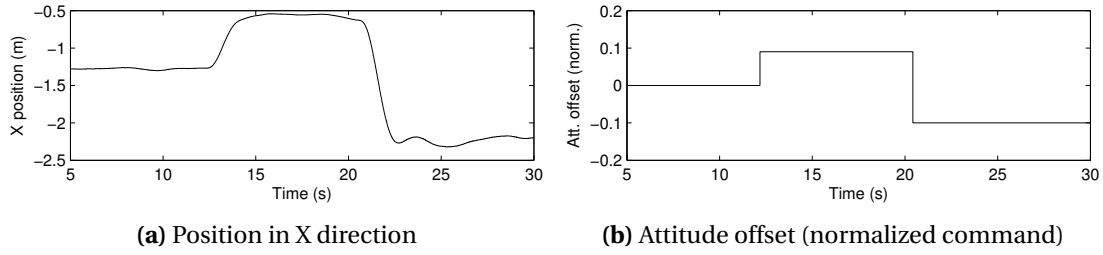


Figure 5.3: Attitude offset performance

Using two quadcopters movement (and thus oscillations) will still be possible perpendicular to the plane intersecting both quadcopters and the mass.

When the pick-up connectors are equipped with reflective markers, the pick-up action can be performed per drone in a semi-automatic way. The "Prepare" button in the interface makes the quadcopter hover in front of the connector, while the "Pick up" button performs the actual pick-up.

5.2.1 Control logic

The connection of the mass results in an additional force working on the body of the quadcopter. Since the controller was not designed with this additional force in mind, the controller structure has to be changed. Again, multiple options were considered and integrated for testing.

1. A fixed offset of the attitude command.
2. An additional integral action in the controller.

It should be stated that there are numerous other ways to adapt the controller logic to the dynamics introduced by the mass. An elegant way would be to use a model of the connection and mass and be able to predict and control the position and pose of the mass directly [Brescianini et al. (2013)]. Within this project the methods above were chosen for their ease of implementation.

Testing attitude offset

If the position of the mass relative to the body fixed frame of the quadcopter is known, the direction of the additional force is known as well. If the mass m and the angle between the quadcopter bottom and the cable (α) is known, the size of the force can be calculated. To get the angle, the position difference of the quadcopter and mass in height (h) and lateral distance (d) can be used. Both relations are given in Equation 5.1.

$$F_{add} = \frac{mg}{2 \tan \alpha} = \frac{mgd}{h} \quad (5.1)$$

This force relates to an additional attitude that has to be imposed, in order to converge to the original setpoint. To test this qualitatively, the line of thought was reversed: if there is no additional force present, but an attitude offset is implemented the quadcopter should hover at a fixed point in space as well, shifted from the setpoint. From the results (Figure 5.3) can be concluded that the attitude offset works as expected. It is good to keep in mind the limitations of the current implementation of this approach.

- Offsets are given in the world frame, allowing small rotations of the quadcopters. This means however that the position difference between the quadcopters and the mass should remain constant in the world frame.

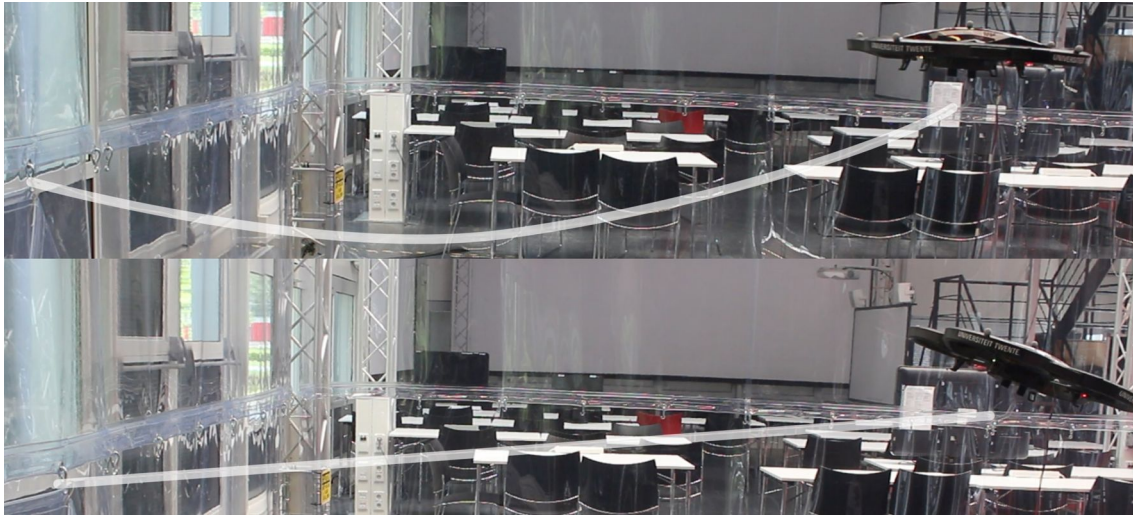


Figure 5.4: Experiment setup for testing integral action

- This approach is not dynamic. Since the relation between force and attitude is not linear the additional attitude might be too much or too less during flight.

Testing integral action

The integral action is a more general approach to a position dependent force in a system. Over time, the integral action will ensure an additional attitude command counteracting the influence of the mass. An integral action was implemented (with, as a safety measure, a reset on receiving new setpoints). To test the integral action, the quadcopter was connected to one of the protective sheets surrounding the flying area through a cable (Figure 5.4) and given new setpoints, too far from the sheet. The attitude of the quadcopter should increase over time if the integral action works.

While the results (Figure 5.5) look promising, the integral action was **not** enabled during this experiment. Apparently, the internal controller of the Parrot does more than just controlling the attitude: it seems that it incorporates the data from the on-board sensors to conclude that additional action is necessary.

The integrative action of the controller is not tested thoroughly, since the on-board I action cannot be switched off and therefore the difference in performance cannot be measured.

5.2.2 Experiments

Since the total process of moving the mass consists of multiple phases (pick-up, flight, drop-off), the experiments can be divided into multiple categories as well. The procedure for the drop-off is not very elegant and consists of the "shaking off" of the connection. The other phases are considered in more detail.

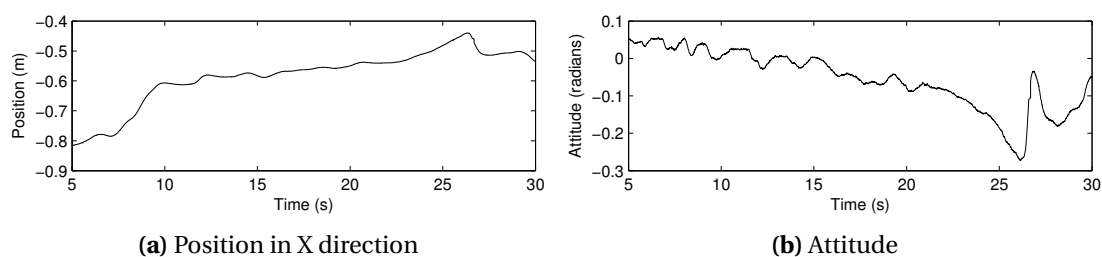


Figure 5.5: Integral action

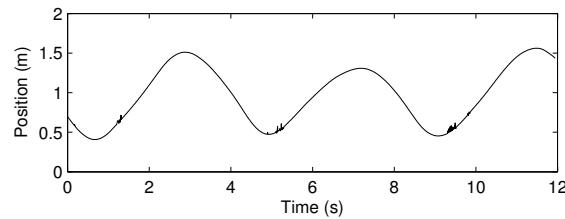


Figure 5.6: Large oscillations in Z direction after connection of hook

Pick-up and connector

When connecting the hook and the connector by hand it can be observed that the behaviour of the drones changes (Figure 5.6). This unpredicted behaviour can be explained by the internal loop of the AR.Drone: commands in the Z direction are relative speeds. The internal loop uses the signal from the ultrasound sensor as a factor in the vertical speed calculation. [AR.DroneSDK (2014)] Hanging objects underneath the drone gives jumps in the ultrasound sensor, interpreted as speed by the internal hardware. Since no speed command has been given, the hardware counteracts. This observation requires changes to the design of the hook and connector. The sensors of the AR.Drone should not be covered. The altered design for the hook can be seen in Figure 5.7. The new design of the connector consists of the three reflective markers connected to a cardboard tube of approximately 0.5 meter. On this tube an elastic slid is mounted, where the ring is pushed in. In this way, the tube and markers stay on the floor when the ring is picked up. Therefore the ring can only be picked up once: the markers do not move with the ring. The elevated mass connector makes that the quadcopter can hover at a higher altitude when picking up the mass, resulting in a more stable setpoint behaviour through the absence of ground effects. Using the new setup, it is possible to pick up the rings (and attached wires) with the quadcopters simultaneously.

Flight

The integrated I action should result in the quadcopters reaching their setpoints when a mass is attached. Since the exact amount the AR.Drone can carry is not specified, some experiments were performed. From the results in Figure 5.9 the following conclusion can be drawn: when a mass is connected that is too large to carry (more than approximately 80 grams), the AR.Drone will periodically decrease the rotor speed (easy to hear). This could be a measure to counteract overheating.

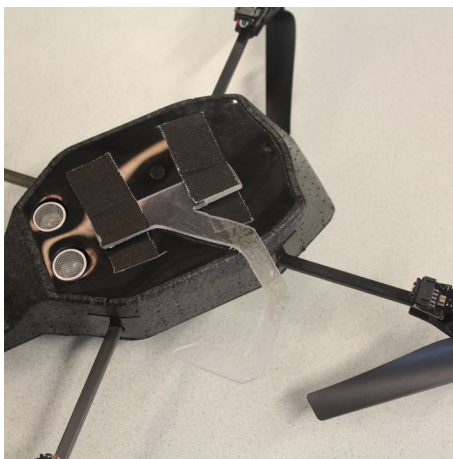


Figure 5.7: New hook design

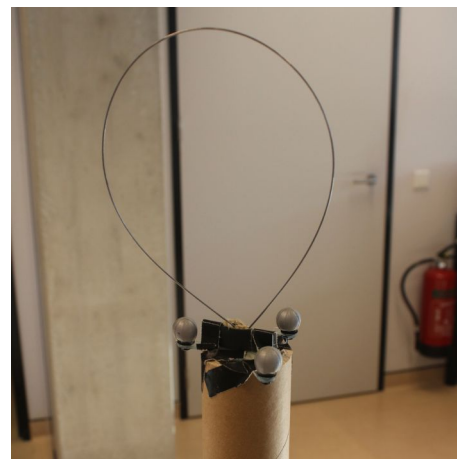


Figure 5.8: New connector design

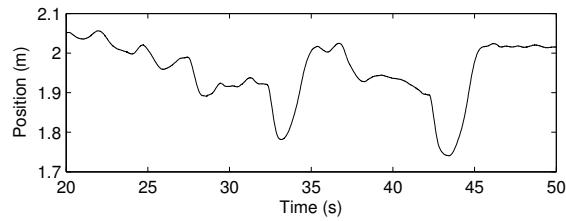


Figure 5.9: Overload results in dips in altitude (at 33 and 43 seconds)

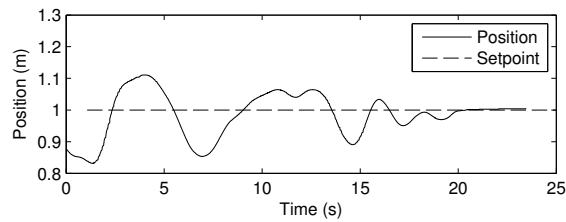


Figure 5.10: Setpoint performance when mass is attached (in line with cable)

Figure 5.10 shows the setpoint performance of a quadcopter while moving a (45 gram) mass collaboratively. It can indeed be concluded that the internal integral action results in a position around the setpoint.

5.3 Connected quadcopters

Moving a mass using cables is not the most efficient way of moving a mass. To deliver enough force to counteract gravity while remaining in a setpoint requires the quadcopter to have a constant attitude, resulting in a higher required rotation speed for the rotors to deliver the upward force. The closer the quadcopters are to each other, the more efficient this configuration becomes. With that in mind experiments were thought of using two, connected quadcopters.

When connecting two quadcopters, choices have to be made for the kind of connection (rigid, flexible, rotational, etc.). With the idea in mind that lifting is most efficient when the quadcopter is leveled, the idea emerged to connect two quadcopters through a rotational joint. In that way, one of the two can do the lifting efficiently while the other can push the combination of mass and quadcopter around through the air. This type of connection has not been tested before with UAVs, in contrast to for example a rigid [Naldi and Ricco (2013)] connection.

5.3.1 Rotational joint

To create a rotational joint without adding too much mass there was chosen to use multiple bands in a crossed orientation (Figure 5.11 and Figure 5.12).

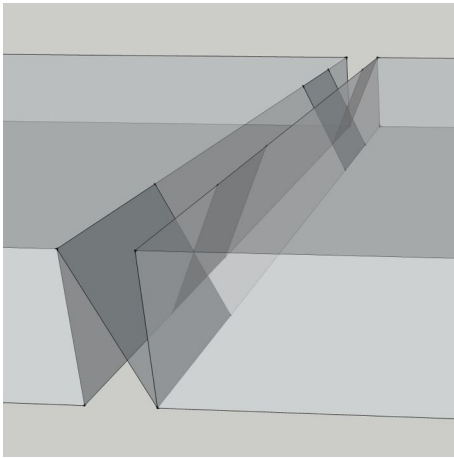


Figure 5.11: Rotational joint (schematic)



Figure 5.12: Rotational joint (implementation)

5.3.2 Experiments

As learned in the previous sections, there is quite a lot of logic inside the Parrot AR.Drone that is not fully described in the documentation (in comparison to the first version of the drone [Bristeau et al. (2011)]). One that is important in this context is the result of sending a "zero" command. It is possible to send a zero command to the drone without the drone entering the hovermode, and thus not using the optical flow measurement. However: sending a zero valued command does not mean the drone will not counteract movement: the velocity estimates are used and counteracted (as can be seen in Section 5.2.1 and Section 5.2.2). There is no way to disable these features (e.g. send a "just stay at this height" or "just stay at this attitude" command). Connecting two drones will therefore, most likely, result in a conflict between the two. As Figure 5.14 shows, this is indeed the case. Sometimes, a situation occurs where the quadcopters actually stay in the air for a short period (Figure 5.13), but over time the integrating action will lead to a crash.

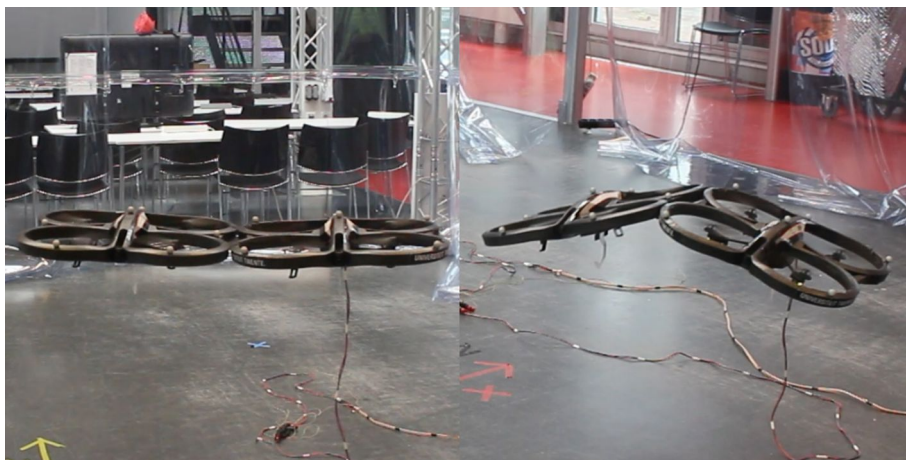


Figure 5.13: Quadcopters connected

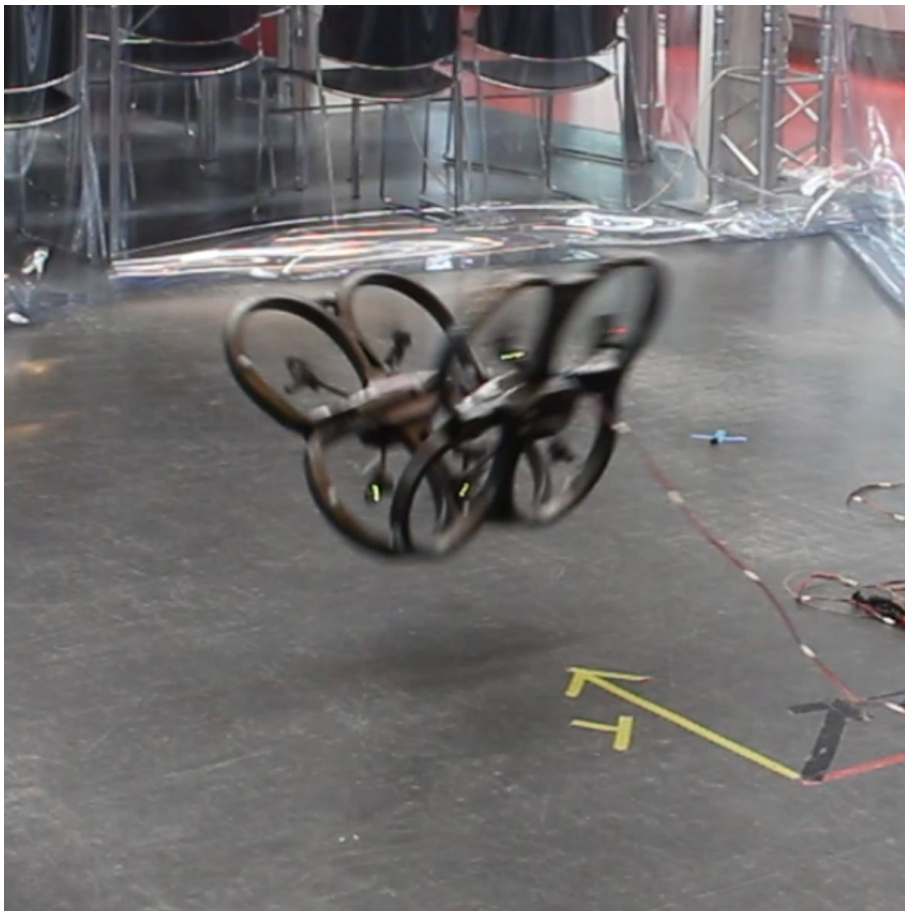


Figure 5.14: Quadcopter during crash

5.4 Intermediary conclusions

A set of Parrot AR.Drones was used to successfully move a load connected through cables. The pick-up of the connections with the cables can be implemented to be performed on the push of a button. The internal logic of the AR.Drone contains an integrating action, making it unnecessary to implement ways to counteract the added forces. Two approaches were implemented however, and partly tested.

This intelligence of the AR.Drone does have negative consequences as well. The fixed fusion of sensor data required a redesign of the hook and connectors. In this new design, repetitive use of the connectors impossible. Another possibility that was rendered impossible was the idea to physically connect two quadcopters: the integrating action caused great instability (and therefore many crashes).

6 Discussion

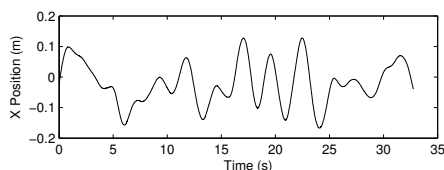
In this chapter reflection takes place, on both the process as well as the contents of this research. Matters that could have been improved or have to be taken into account in future projects are listed in the following section.

6.1 Flying robots

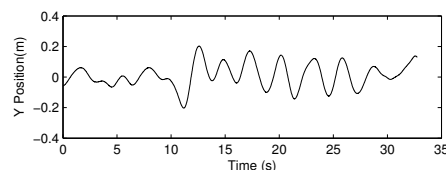
In this projects flying robots were used. This requires a optimized location within the laboratory to operate such robots. It was found that the size of the room had enormous influence on the performance of the designed systems. The setpoint behaviour of a quadcopter in the RAM Flying Arena (Figure 6.1) features very large deflections compared to other experiments in this thesis (standard deviation of 0.0737 for a single drone). These effects of airflow were greatly underestimated in the beginning of the project, resulting in delays. From the moment the influence was known, all experiments were done in the larger SmartXP Lab. This room however is not always available and requires approximately 45 minutes for conversion from or to a regular lecture room.

6.2 Practical nature

The practical nature of this assignment results in very practical deliverables: applications, reusable code, documentation of experiments (see Appendix A). Together with the closed software of the Parrot AR.Drone, there was not much room to dive into new territory for quadcopter research.



(a) Setpoint performance in x direction



(b) Setpoint performance in y direction

Figure 6.1: Oscillatory motion in flying arena

7 Conclusions and recommendations

From this project can be concluded that the Parrot AR.Drone is a great research tool. It is easy to work with and can withstand a large number of crashes without breaking down. Through ROS and the AR.Drone SDK the quadcopter can be controlled through practically every input interface.

With state feedback present through the OptiTrack system, PD position control was implemented and tested successfully (accuracy of approximately 0.05 meter). Through applications developed in this project, the simultaneous control of multiple drones was made easy and intuitive.

Flying multiple drones at once however resulted in a large additional airflow, negatively influencing the setpoint behaviour. Performance was still adequate to perform experiments on the main subject of this project: moving a mass using a set of quadcopters. For this cause a specific connector and hook were designed.

The downside of working with a commercially available platform like the Parrot AR.Drone is its closed nature. The exact control algorithms are not known and therefore unpredictable behaviour can occur. In this project, the intelligence of the quadcopter had both positive and negative effects: the built-in I action resulted in the quadcopters reaching their setpoints even when a mass was connected; connecting two quadcopters through a rotational joint however was made impossible by this same I action. Future researchers looking into quadcopters should consider carefully if the Parrot AR.Drone is the right platform for their research.

In general, this research contributed to the idea of quadcopters collaborating to perform a common task. The setup developed in this project can be seen as a proof of concept for using quadcopters for delivering small packages. The software architecture developed in this project can be easily adapted to be used in other projects, featuring other position feedback systems or UAV platforms.

Code written for this project can easily be reused for other tasks and/or quadcopters and will therefore be shared publicly. Appendix A contains references to available documentation on the use of the software.

A Appendix 1 - Documentation for further use

All code in this project is inspired by other projects and programmers. For that reason it is publicly shared and documented through GitHub: http://github.com/ceesietopc/ram_ba_package.

Information specific for the flying areas at the University of Twente or available tools can be found on the Wiki of the Control Engineering group: <http://wiki.ce.utwente.nl>

Bibliography

- AR.DroneSDK (2014), Parrot AR.Drone 2.0 Developer Zone.
<http://ardrone2.parrot.com/developer-zone/>
- Barron, J., D. Fleet and S. Beauchemin (1994), Performance of optical flow techniques, *International Journal of Computer Vision*, **vol. 12-1**, pp. 43–77, ISSN 0920-5691, doi:10.1007/BF01420984.
- Bernard, M., K. Kondak, I. Maza and A. Ollero (2011), Autonomous transportation and deployment with aerial robots for search and rescue missions, *Journal of Field Robotics*, **vol. 28**, pp. 914–931, ISSN 1556-4967, doi:10.1002/rob.20401.
<http://dx.doi.org/10.1002/rob.20401>
- Brescianini, D., M. Hehn and R. D’Andrea (2013), Quadcopter pole acrobatics, in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 3472–3479, ISSN 2153-0858, doi:10.1109/IROS.2013.6696851.
- Bristeau, P., F. Callou, D. Vissière and N. Petit (2011), The navigation and control technology inside the ar. drone micro uav, *18th IFAC world congress*, pp. 1477–1484.
<http://cas.enscm.fr/~petit/papers/ifac11/PJB.pdf>
- Franklin, G., J. Powell and A. Emami-Naeini (2010), *Feedback Control of Dynamic Systems*, Pearson Education, Limited, ISBN 9780135001509.
http://books.google.nl/books?id=_-cVQgAACAAJ
- Fumagalli, M., R. Naldi, a. Macchelli, R. Carloni, S. Stramigioli and L. Marconi (2012), Modeling and control of a flying robot for contact inspection, *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3532–3537, doi:10.1109/IROS.2012.6385917.
<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6385917>
- Hernandez, A., C. Copot, R. De Keyser, T. Vlas and I. Nascu (2013), Identification and path following control of an AR.Drone quadrotor, in *2013 17th International Conference on System Theory, Control and Computing (ICSTCC)*, IEEE, pp. 583–588, ISBN 978-1-4799-2228-4, doi:10.1109/ICSTCC.2013.6689022.
<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6689022>
- Janabi-Sharifi, F., V. Hayward and C.-S. Chen (2000), Discrete-time adaptive windowing for velocity estimation, *Control Systems Technology, IEEE Transactions on*, **vol. 8-6**, pp. 1003–1009, ISSN 1063-6536, doi:10.1109/87.880606.
- Maza, I., K. Kondak, M. Bernard and A. Ollero (2010), Multi-UAV Cooperation and Control for Load Transportation and Deployment, *Journal of Intelligent and Robotic Systems*, **vol. 57**, pp. 417–449, ISSN 0921-0296, doi:10.1007/s10846-009-9352-8.
<http://dx.doi.org/10.1007/s10846-009-9352-8>
- Monajjemi, M. (2014), AutonomyLab / ardrone_autonomy.
https://github.com/AutonomyLab/ardrone_autonomy
- Naldi, R. and A. Ricco (2013), A modular aerial vehicle with redundant actuation, *Intelligent Robots and ...*, pp. 1393–1398.
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6696531
- NaturalPoint (2014), NaturalPoint OptiTrack Optical Tracking Systems.
<http://www.naturalpoint.com/optitrack/>
- Parra-Vega, V., A. Sanchez, C. Izaguirre, O. Garcia and F. Ruiz-Sanchez (2012), Toward Aerial Grasping and Manipulation with Multiple UAVs, *Journal of Intelligent & Robotic Systems*,

vol. 70, 1-4, pp. 575–593, ISSN 0921-0296, doi:10.1007/s10846-012-9743-0.

[http://www.scopus.com/inward/record.url?eid=2-s2.](http://www.scopus.com/inward/record.url?eid=2-s2.0-84871640766&partnerID=tZ0tx3y1)

[0-84871640766&partnerID=tZ0tx3y1](http://www.scopus.com/inward/record.url?eid=2-s2.0-84871640766&partnerID=tZ0tx3y1)

Parrot (2014), Parrot AR.Drone 2.0.

<http://ardrone2.parrot.com/>

Purvis, M. (2014), Clearpath Robotics / mocap_optitrack.

https://github.com/clearpathrobotics/mocap_optitrack

ROS.org (2014), About ROS.

<http://www.ros.org/about-ros/>