Multi-Agent Testbed development, modelling and control of Quadrotor UAVs

August 19, 2012

AXEL KLINGENSTEIN

Master’s Degree Project
Stockholm, Sweden 2012

XR-EE-RT 2012:019
January to August 2012

Multi-Agent Testbed development, modelling and control of Quadrotor UAVs
— Diploma Thesis —

Axel Klingenstein

Automatic Control Laboratory
School of Electrical Engineering, KTH Royal Institute of Technology, Sweden

Supervisor
Meng Guo and Jose Araujo
KTH Stockholm

Examiner
Dr. D. V. Dimarogonas
KTH Stockholm

Stockholm, August 19, 2012

1axel.klingenstein@gmail.com
Quadrotor Unmanned Aerial Vehicles (UAVs) have been an area of great interest for academic research at several universities around the world. They provide an interesting platform for research into various control related areas such as modelling, testing different control strategies or cooperative behaviour and possible real-world applications within search-and-rescue, traffic surveillance and many other areas with the potential to improve everyday life. In this thesis, the process of designing and implementing a testbed that allows for practical multi-agent control related experiments with Quadrotor UAVs is described. The testbed consists of a centralized controller PC, a positioning system for determining the quadrotors position in space and a bi-directional wireless link for communicating with the quadrotors. The controller PC software in LabVIEW and the various interfaces to the subsystems in C, C# and under TinyOS are implemented considering the main goal to have a robust platform that allows for quick testing and debugging of different control algorithms. To be able to perform simulation and understand the dynamic behaviour of the quadrotors, a mathematical model is derived. The model is then used to design PID and LQR controllers that allow the quadrotors to track a position in space. The functionality of the testbed, the performance of the controllers and the validity of the model is successfully evaluated by flight-testing the quadrotors using the testbed and verifying that simulation results are in accordance with the real-world results that are obtained.
Acknowledgments

I would like to thank Dimos Dimarogonas for giving me the possibility to conduct this interesting thesis and for all the advise and time he gave me in addition to his duties as an examiner. I would also like to thank Meng Guo for all the help on the theoretical part he has provided as my supervisor. Further, I would like to thank Jose Araujo for the inspiration to start this thesis, all the practical help he provided in the beginning and for supervising my work. Special thanks also to Patrik Almström of Qualisys AB for inviting us to Gothenburg and allowing us to work with the Qualisys system.

Axel Klingenstein
August 19, 2012
Contents

1 Introduction ................................................................. 1
  1.1 Background ......................................................... 1
    1.1.1 Quadrotor UAV applications ............................... 2
  1.2 Motivation ......................................................... 3
  1.3 Goals ............................................................. 3
  1.4 Outline .......................................................... 4

2 Quadrotor Platform ................................................... 5
  2.1 Introduction ....................................................... 5
  2.2 Technical description ........................................... 5
    2.2.1 Airframe ...................................................... 5
    2.2.2 Control inputs and modes of operation .................. 6
    2.2.3 Flight electronics .......................................... 6
    2.2.4 Mission Planner Software ................................. 7
  2.3 Principals of flight and manoeuvring ......................... 8
    2.3.1 Level, stationary flight and vertical movement ........ 8
    2.3.2 Pitch, roll and lateral movement ....................... 9
    2.3.3 Rotational or yawing movement .......................... 9

3 Quadrotor Testbed ..................................................... 11
  3.1 Introduction and specifications ................................ 11
    3.1.1 The closed-loop control system .......................... 11
    3.1.2 Agents - Quadrotors and other .......................... 12
  3.2 Controller PC .................................................... 13
    3.2.1 Software implementation .................................. 13
  3.3 Wireless communication link ................................... 15
    3.3.1 Tmotes technical description ............................ 15
    3.3.2 Tmotes firmware implementation ......................... 16
    3.3.3 Tmote Sky - Controller PC interface .................... 17
  3.4 Positioning systems ............................................. 17
    3.4.1 Ubisense system ........................................... 17
    3.4.2 Qualisys system .......................................... 19

4 Quadrotor dynamic Model .......................................... 21
  4.1 Modelling ....................................................... 21
Contents

4.2 Model derivation ................................................................. 21
  4.2.1 Non-linear model .......................................................... 24
  4.2.2 Linearized model .......................................................... 24

5 Controller design ................................................................. 27
  5.1 PID control ................................................................. 27
    5.1.1 Theory ........................................................................ 27
    5.1.2 Design ....................................................................... 28
  5.2 LQG control ................................................................. 29
    5.2.1 Theory ........................................................................ 29
    5.2.2 Design ....................................................................... 31

6 Evaluation and Results .......................................................... 33
  6.1 Simulation ................................................................. 33
    6.1.1 PID .......................................................................... 33
    6.1.2 LQR ......................................................................... 34
  6.2 Testbed experiments ......................................................... 36

7 Conclusion and further work ................................................. 39

References ................................................................. 41
1 Introduction

1.1 Background

Throughout history, mankind has always expressed a fascination for flying and aerial vehicles. First concepts for VTOL (Vertical Take-Off and Landing) vehicles were made as early as 1490 with Leonardo da Vinci’s drawings of his “Helix Pteron” or “Aerial Screw”, but it was not before 1936 that Prof. H. Focke designed and built what is considered as the first functioning prototype of a helicopter.

![Leonardo da Vinci’s “Helix Pteron” (1490); Prof. H. Focke’s first helicopter prototype (1936); (Source: http://commons.wikimedia.org/)](image)

**Figure 1.1**: Left: Leonardo da Vinci’s “Helix Pteron” (1490); Right: Prof. H. Focke’s first helicopter prototype (1936); (Source: http://commons.wikimedia.org/)

Dating back several decades, helicopters have become widely commercially available for a relatively low cost. In recent years, development has among other directions gone towards UAV, Unmanned Aerial Vehicles often referred to as drones, that allow for a mission to be performed where a human pilot is not desirable because of safety or other concerns. Furthermore, UAV’s allow for a much smaller vehicle construction, where a human pilot physically would not fit. Applications for UAV’s have from the beginning mainly been military, but recently drones have also been used in other areas such as civil security, search-and-rescue (SAR), fire fighting and aerial photography. A benefit of VTOL UAV’s, such as for example quadrotors that will be treated in this thesis, is that they can take-off in a very limited space and also hover at a fixed point in space, which makes them ideal for reconnaissance and surveillance missions. In the next section, some sample applications for (micro) VTOL UAV quadrotors will be described in detail.
1 Introduction

Figure 1.2: Left: Search-and-Rescue UAV used by U.S. Coastguard; Right: Surveillance UAV quadrotor used by German police force; (Source: http://commons.wikimedia.org/).

1.1.1 Quadrotor UAV applications

In this section, two examples for practical applications of UAV quadrotors will be given. The examples have, as it was considered when this thesis was written, not been implemented in reality, but the concepts that are described in this thesis could be modified to allow for the following scenarios.

Search-and-Rescue

If a person is lost within a large area of land, one or several quadrotors flying in formation can be used to search the area much faster than grounded SAR personnel could do. The quadrotors can be equipped with cameras, heat detectors or other sensors to locate the missing person. In the case of skiers or climbers that are buried under a snow avalanche, special sensors could be used to detect the signal that is emitted from beacons that are often fitted to mountain gear [1]. If a missing person is found, a quadrotor could hover over the area and signal the location by flash-lights and/or sound. If several quadrotors are used and the missing person is located a large distance from the SAR-crew, a cooperative scenario is possible where one quadrotor hovers above the person and another flies back to the rescuers and guides them to the location.

Traffic surveillance

A fleet of UAV quadrotors equipped with cameras could be used to monitor likely traffic congestion points within a city. The advantage compared to permanently installed cameras is, that if there is no congestion the quadrotor can quickly be moved to another location. Furthermore, on a long road it might not be viable to install cameras along the whole path. Mobile quadrotors could be used to monitor the situation before an accident, warn other cars that are coming from behind and possibly even to find alternate routes that are not blocked.

Another scenario is related to platooning of several heavy-duty or personal vehicles. One or several quadrotors could follow the platoon and provide vision of a much larger area. This data can be used by the platoons control system to consider the traffic situation ahead of the vehicles to calculate an optimal configuration, with respect to e.g. fuel economy, in which they should travel. The safety is increased because the UAVs can detect accidents/road blocks
from a distance. If several quadrotors fly in a circular motion around the platoon, they could also see what alternative routes are available that are not blocked in case of a congestion.

1.2 Motivation

Academic research within the (micro-) VTOL UAV area has been of great interest at several universities around the world. Challenging problems like modelling, control, trajectory generation and cooperative behaviour have been in the focus of researchers and have been dealt with both in theory and practice. Before this thesis had begun, the work within this area at the KTH Automatic Control Laboratory, Stockholm, was mainly of theoretical nature. There was a strong desire to practically implement some of the interesting concepts and to be able to see real-world vehicles perform various tasks that had been designed by researchers and professors. Furthermore, a quadrotor UAV is an ideal platform for control experiments, since its dynamic behaviour can be understood intuitively. The platform can be used by many undergrad or MSc students in the future.

The personal interest of the author in UAVs on the one hand and the work of previous universities on the other have also been large factors for starting this work. The thesis can be considered as a first step for the KTH Automatic Control into practical research with quadrotor UAVs, which is a unique chance to build a platform that others can expand upon in further work.

1.3 Goals

One of the primary goals for this thesis was to provide the KTH Automatic Control Laboratory with a testbed for performing control experiments with the quadrotor UAV agents. Furthermore, the testbed could be used to do the experiments that are a part of this thesis. It should allow for easy debugging and provide a robust and adaptable platform for multiple agents. The main three parts that have to be designed to make the testbed work are the interface with the positioning system, that provides the controller with the agents current position in space to enable closed-loop control, a wireless bi-directional link for sending actuation commands to the agents and receiving sensor data, and the main controller implementation that calculates a desired output for each of the agents based on current and past input signals.

The second series of targets for this thesis is related more directly to the quadrotors. At first, a detailed and faithful model has to be derived to be able to perform simulations and test possible controller implementations. The model has to take into account what inputs are available on the chosen quadrotor platform and accurately resemble the dynamic behaviour of the agent. When the model is finished, simulation can be performed and the results compared to measurements that are made on the real quadrotors using the testbed. This will help to verify that the model is correct. The final step will be to implement different control algorithms and to evaluate the performance in simulation and in real-world. In summary, the following goals should be achieved:
1 Introduction

- Quadrotor UAV Testbed
  - Develop an interface for the positioning system that allows to retrieve all necessary data and make it accessible to the controller implementation.
  - Implement a bi-directional wireless link between the agents for sending actuation commands and retrieving data from their sensors.
  - Build a framework that allows for easy implementation and debugging of new controllers.

- Quadrotor modelling
  - Derive a detailed and faithful quadrotor model for the specific quadrotor platform that was chosen.
  - Verify that the model is correct by comparing simulation and real-world measurements.

- Quadrotor control
  - Implement, test and compare the performance of different control algorithms in simulation and real-world.

1.4 Outline

This section will give a chapter-by-chapter outline of all topics covered in this thesis. This section is part of chapter 1, that gives some background and applications to the topic and summarizes the goals of the thesis. Chapter 2 deals with the quadrotor platform that was chosen to be worked with during this thesis. The technical details of the airframe, electronics and also architecture of the open-source software will be covered. The last section in this chapter is about principles of flight for the quadrotor, where basic manoeuvres such as pitching, rolling and yawing will be covered. In chapter 3, the development of the testbed is described. Design goals alongside with applications are stated and then a technical description of the three main parts is given, which are the controller PC implementation, the interface with the positioning system and the wireless bi-directional link to the agents. In chapter 4 of this thesis, a faithful dynamic model of the quadrotor will be derived. The non-linear equations governing the behaviour of the quadrotor will be stated and a linearized model based on Taylor estimation will be derived. Chapter 5 describes the theoretical background and the design of PID and LQR controllers for the quadrotor platform that can be used in simulation and real-world experiments. The final chapter 6 is about the results that where obtained from different experiments that where performed in simulation and using the testbed.
2.1 Introduction

When the work with this thesis started, researchers at the KTH Automatic Control Lab where beginning to discover the possibilities of the DIYDrones ArduCopter MultiCopter platform [2]. Because of the availability at the lab and the previous effort that had gone into testing, this platform was chosen to be worked. The platform consists of the flight electronics with a micro-controller and sensors and an open-source software architecture that can be used to control multi-rotor vehicles like tri-, quad- or hexarotors. The electronics can be used with several alternatives as airframes, but for this thesis a JDrones [3] Quad quadrotor platform was used, that consist of a aluminium frame, four electric motors with RPM controllers and propellers, power distribution board, landing gear and parts for mounting the electronics.

2.2 Technical description

2.2.1 Airframe

![Airframe Diagram]

Figure 2.1: The complete JDrones quadrotor airframe with ArduCopter flight electronics mounted (http://code.google.com/p/arducopter).
2 Quadrotor Platform

A picture of the assembled airframe is shown in figure 2.1, where the most important parts have been pointed out. The basis of the frame consist of aluminium bars (5) to which all other components are mounted. The span from motor to motor is about 60 cm. The motors (1) are of electrical, brushless type and have a RPM/Volt rating of 850. They can rotate at a maximum rotational velocity of about 11000 RPM and generate a thrust of almost 1 kg per motor using a standard 10x45 propeller [3]. The four motors are powered by a Lithium-Ion Polymer (LiPo) three-cell battery (3) with a nominal voltage of 11.2V and a rated capacity of 2200 mAh. One speed controller (4) is used for each motor to generate the switched AC current needed by the brushless motors and thereby control the motors rotational velocity proportional to the applied input signal.

2.2.2 Control inputs and modes of operation

The quadrotor platform is designed to be controlled by an standard hobby radio-control unit [4]. The receiver of the unit can be connected to the quadrotors electronics and will provide full manual control of pitch, roll and yaw angles plus throttle. For the manual control of these inputs, there are two main modes of operation available. For the first mode, for any input angle, the internal controller of the quadrotor will stabilize it at that angle. If the inputs are zero, the quadrotor will return to level flight. In the second mode of flight, a certain amount of input angle corresponds to a rotational velocity around that angle, meaning that a larger input will make the quadrotor rotate faster and no input will correspond to no rotation (not necessarily level). To control the quadrotor for the experiments performed during this thesis, mainly the stabilized mode of flight was used where an external controller took care of handling the manual inputs.

In addition to the manual control modes, several autonomous flight modes are available such as waypoint tracking, loitering at a fixed point in space, automatic landing and take-off. Predefined routes can be programmed through a special software graphical user interface that is supplied with the quadrotor. These modes however are designed for outside operation and rely on position data delivered by a GPS sensors.

2.2.3 Flight electronics

The ArduCopter open-source firmware is designed to be run on the ArduPilot Mega1 circuit board [5]. Here, an Atmel ATMega 2560 micro-controller running at 16MHz is used to run the controller software and provides analog and digital in-/outputs to connect to external sensors. The ArduPilot board also contains a PWM interface for decoding signals coming from a radio-control (RC) remote controller and for encoding signals going to the speed controllers or other servos. The firmware can be configured via an USB interface that allows to view sensor data, reprogram the micro-controller and to adjust parameters used by the quadrotors controller. It uses the Arduino library [6] to implement some of its functionality. Several additional libraries are supplied that allow the use of all sensors and additional hardware for own project.

Sensors

The quadrotor platform is equipped with several sensors that are used by the internal controller for various tasks. Since the platform is open-source, when developing own external controllers the sensors could be easily accessed and drivers/libraries were available that provided some abstraction and made the implementation of own applications easier. Most of
the sensors that are used for stabilized flight are mounted on a second circuit-board, called Inertia-Measurement-Unit or IMU, that is connected directly to the one where the main micro-controller is located. The IMU provides a 3-axis gyroscope and 3-axis accelerometer that are coupled to 12 bit analog-to-digital converters to get accurate readings. Furthermore, there is an absolute pressure sensor on the board that can be used to measure the flight height of the quadrotor when flying outdoors at high altitudes.

Several external sensors can be connected to the quadrotor’s electronics. A magnetometer can be used to compensate for the long-term drift of the gyroscope, a GPS sensor facilitates outdoor waypoint tracking and loitering and an ultrasonic range finder can be used to accurately measure the distance to the ground at relatively low altitudes below zero and seven meters.

### 2.2.4 Mission Planner Software

The quadrotor can be programmed and configured through an external Microsoft Windows application named “Mission Planner”. This software will graphically display the output of all sensors on the quadrotor, allowing to see if everything is working correctly. Additionally the internal quadrotor controller gains can be adjusted to allow for the use of different airframes, motors, etc; the firmware can be updated and several other parameters such as e.g. calibration data for different RC remote controllers can be adjusted.

![Figure 2.2: The Mission Planer software that is supplied with the ArduCopter platform.](image)
2.3 Principals of flight and manoeuvring

In this section, the principles of flight for the quadrotor will be explained and a description will be given of how it can perform different manoeuvres that are directly related to the manual control inputs (pitch, roll and yaw). In contrast to the mathematical model that is provided in chapter 4, this section will try to give a more intuitive understanding of how the quadrotor can fly and move about in space.

2.3.1 Level, stationary flight and vertical movement

![Figure 2.3: A quadrotor in level, stationary flight.](image)

Each of the four motors produces a torque \((M_1, \ldots, M_4)\) which spins the propeller and generates an upwards thrust \((F_1, \ldots, F_4)\). The torque from the propellers is also applied to the quadrotor body, so in order for the quadrotor to fly steadily and not spin out of control, it has to be balanced. This is achieved by making two of the four propellers, that are opposite to each other \((e.g. \ M_1 \ and \ M_3)\), spin in the other direction as the remaining two propellers. This way the torque that is applied to the quadrotor will be balanced out in the same way as for example a tail rotor is used to cancel out the torque from the main rotor of a traditional helicopter.

If all propellers spin at the same rate, they will generate the same amount of thrust and a total thrust vector \(F_{tot}\) will be incident at the center of gravity of the quadrotor that points directly upwards (positive z direction). This total thrust vector can be directly influenced by the quadrotors controller by making the propellers spin at different rates. Now, if the total...
2.3 Principals of flight and manoeuvring

thrust is equal to the gravitational force $F_g$, the quad will stay at the same height; if it is larger than the gravity move upwards and if it is smaller downwards.

2.3.2 Pitch, roll and lateral movement

![Figure 2.4: A pitching quadrotor with altered total force vector $F_{tot}$.

For the quadrotor to pitch and thereby move laterally, the rotational velocity of the propellers can be altered such that for example the thrust $F_2$ is greater and $F_4$ is less than that generated by the other two propellers (see figure 2.4). For the scenario in the figure, a torque around the axis through propeller 1 and 3 will be generated causing the quadrotor to tilt in the direction of propeller 4. The total force vector $F_{tot}$ is tilted by an equal amount with respect to the surrounding environment and this will cause the quadrotor to move laterally in that direction. By varying the rotational velocity of the corresponding propellers, the quad can move in all four lateral directions and of course also any viable combination of them.

2.3.3 Rotational or yawing movement

To perform a rotational or yaw-movement, the torque generated by the propellers can be altered so that the sum is no longer zero around the central quadrotor axis. This is achieved by lowering the rotational velocity of two propellers that are opposite of each other, as seen in figure 2.5. Here, by increasing the rotational velocity of propellers 1 and 3, the torques $M_1, M_2$ are also increased while at the same time the torques $M_2, M_4$ are decreased by the same amount. This will generate negative total torque $M_{tot}$, causing the quadrotor to rotate.
around its center axis. Because the propellers are increased and decreased by the same amount, the total force vector $F_{\text{tot}}$ is not altered and the quadrotor will either stay at the same altitude, or continue climbing/descending, while it is rotating.

Figure 2.5: A rotating or yawing quadrotor.
3

Quadrotor Testbed

3.1 Introduction and specifications

To be able to effectively develop closed-loop position feedback controllers for the quadrotors (referred to as agents), it was necessary to have a regulated environment that allows for quick implementation and easy debugging of control algorithms. Furthermore, the testbed should allow a user of the system to implement different controllers with minimal effort and provide functionality for conducting multi-agent control experiments. All of the experiments that were conducted as a part of this thesis utilized the testbed, but it was also designed to provide the KTH Automatic Control Laboratory with a platform for doing practical research in the multi-agent cooperative control area. To achieve this goal and as a major contribution to this thesis, a testbed for ground and aerial vehicles that is described in this chapter was developed. A software and hardware framework was implemented, consisting self-developed program code and applications, but also commercially available products that were integrated into the system.

As an overview, the testbed consist of three main subsystems plus the agents and the according interfaces. A positioning system is used to determine the agents position in space. A workstation computer running Microsoft Windows (referred to as ”controller PC”) is used to run the main controller application and to connect all subsystems together. Finally, a wireless link facilitates bi-directional communication between the agents and the controller for transmitting actuation commands to the agents, and telemetry data from the agents back to the controller. The following sections in this chapter will give a detailed description of all different parts functionality, performance and technical data of the quadrotor testbed and the process that was undergone whilst designing it.

3.1.1 The closed-loop control system

To be able to accurately control each agent’s position in space, the three subsystems described in the foregoing section where connected with the agents in a position-feedback control loop configuration. An overview of this loop and the parts that are involved is given in figure 3.1.

The controller PC is used to provide a platform for implementing different controller algorithms for cooperative control of the agents. To be able to calculate the desired actuation signals for each individual agent, an interface is provided on the controller PC to access both data coming from the positioning system and also from the agents on-board sensors directly via the wireless link. After retrieving all necessary data, an actuation output command can be calculated by the controller and transmitted to the agents. The wireless bi-directional link
serves this purpose of transmitting the commands to the agents while allowing them to freely manoeuver in space without being constrained by a cable. It is implemented using TMote Sky wireless modules and more details can be found in section 3.3.

### 3.1.2 Agents - Quadrotors and other

For the tests that where performed as a part of this thesis, the quadrotor platform as described in chapter 2 was used. However, the testbed can with minimal modification be used to control other types of airborne or also grounded vehicles like radio-controlled small-scale car or truck models. This will allow for further research into multi-agent cooperative control with both airborne and grounded agents.

For the agents to work with the testbed without further modifications, they have to provide direct access to their actuation interface through a standard 3-pin radio-control servo connector or a serial interface that allows to directly control these signals. Furthermore, their size and maximum velocity should be suited to the indoor space were the positioning system is installed and they have to allow for the tags or reflective markers of the positioning system to be installed on them.
3.2 Controller PC

The role of the controller PC in the testbed is to provide a platform for implementing and testing controller algorithms for all agents in the system. During the implementation of the testbed software on this PC, the goal was to have a setup that would allow for quick testing and debugging of different control algorithms. Furthermore, the robustness of the system was an important consideration.

The software on the controller PC consists of different modules that are shown in figure 3.3 and explained in detail in the following sections. The controller task allows for different algorithms for multi-agent control to be implemented. The graphical user interface serves as the user’s access point to the testbed and allows for starting/stopping experiments, adjusting controller parameters and monitoring test related data. Finally, the interface tasks provide functionality for accessing the actuation and telemetry bi-directional wireless link and the data coming from the positioning system.

The controller task and the GUI were implemented in National Instruments LabVIEW 2011 development environment. The task that handles the communication with the Tmote Sky wireless modules is programmed in a combination of C and C++ and the interface to the positioning system in C# .Net.

3.2.1 Software implementation

As mentioned before, the main function of the testbed is to have a platform for easily testing and debugging different algorithms for multi-agent control. An overview of the software architecture on the controller PC is given in figure 3.3. The controller task serves the purpose of handling the main implementation of the controller that will affect the agents positions in space. Data acquisition is handled through reading the current values from the positioning system interface task and the Tmote Sky wireless link interface task. Data from both sources is retrieved over a TCP/IP connection using different ports. After the control output signal is calculated, a second instance of the Tmote Sky interface task is used to send the actuation commands to the agents, also using a TCP/IP connection.
The graphical user interface (GUI) provides a user of the system with the possibility to quickly adjust different parameters that are relevant for the cooperative control experiments, such as varying the gain of the controllers that are used. Alongside this, the measurements taken from the positioning system and the agents on-board sensors and are displayed, debugging functionality is provided and finally the possibility to manually control the actuation inputs of the agents.

For implementing the GUI and controller task, National Instruments LabVIEW 2011 integrated development environment was chosen. LabVIEW (short for “Laboratory Virtual Instrumentation Engineering Workbench”) provides the possibility for programming in graphical language named “G”. An extensive library of preprogrammed subroutines is included in the distribution which simplifies tasks such as accessing the serial port or TCP/IP communication. Further reasons for choosing LabVIEW were that development in the graphical programming language is quick and that it inherently supports parallel execution of multiple tasks, which is very useful when designing controllers for several vehicles at the same time.

A sample implementation of the GUI for controlling a quadrotor with four PID controller for pitch, roll, yaw and throttle is provided in figure [3.4]. When different control strategies such as LQG or MPC are used, the parameters that need to be adjusted vary and the GUI will have to be adapted accordingly. With the sample implementation that is provided as a reference framework and considering the easily learned user-friendly nature of the LabVIEW development environment, this will be an easy task for researchers working with the testbed in the future.
3.3 Wireless communication link

3.3.1 Tmotes technical description

The MoteIV Tmote Sky \[8\] is a wireless interface featuring a 2.4 GHz IEEE 802.15.4 Chipcon wireless transceiver and an 8 MHz Texas Instruments MSP430 microcontroller on the same circuit board. The microcontroller features USB, USART, SPI, I^2C and GPIO interfaces for connecting itself and the radio to other external components, such as the agents or the controller PC in case of the testbed.

For the testbed application, the TinyOS operating system \[9\] was chosen to be run on the Tmote Sky module. TinyOS provides task scheduling, shared resource handling and several layers of abstraction for accessing the underlying hardware components of the Tmote Sky modules. To create user-defined applications, they can be written in proprietary programming language named NesC. The syntax is similar to the C programming language, but the execution model is event-driven instead of procedural as in C.
3 Quadrotor Testbed

![The MoteIV Tmote Sky wireless module.](image)

**Figure 3.5:** The MoteIV Tmote Sky wireless module.

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>MoteIV Cooperation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Tmote Sky</td>
</tr>
<tr>
<td>Radio</td>
<td>2.4 Ghz IEEE 802.15.4 Chipcon</td>
</tr>
<tr>
<td>Radio throughput</td>
<td>250kbps</td>
</tr>
<tr>
<td>Radio range</td>
<td>50m indoors; 125m outdoors</td>
</tr>
<tr>
<td>Microcontroller</td>
<td>TI MSP430 8MHz</td>
</tr>
<tr>
<td>Interfaces</td>
<td>USB, SPI, USART, I²C, GPIO</td>
</tr>
<tr>
<td>Supply voltage</td>
<td>2.1 – 3.6 V</td>
</tr>
<tr>
<td>Dimensions</td>
<td>32 x 66 x 6 mm³</td>
</tr>
<tr>
<td>Weight</td>
<td>20 g</td>
</tr>
</tbody>
</table>

**Table 3.1:** Technical data for the Tmote Sky module.

### 3.3.2 Tmotes firmware implementation

To establish bi-directional communication between an agent and the controller computer, two Tmote modules were used on both the agent and on the computer (four in total). One of the modules on each side was used for sending and the other for receiving, facilitating bi-directional communication. The reason for this is that the wireless radio module on the Tmote Sky cannot receive and send at the same time, so with this configuration it was not necessary to develop a scheduling algorithm, which would have been outside the scope of this thesis.

For each of the four different Tmote Sky modules pictured in 3.6, a specific firmware was developed and run on top of the underlying TinyOS operating system layer on the module. One of the Tmote modules that is connected to the controller PC is programmed to receive data over the wireless radio module. The data is then forwarder to the PC via virtual serial port from where it can be read from the serial forwarder application. The second Tmote module that is connected to the PC receives data from the controller task via another instance of the serial forwarder and passes it on to the radio module. All data that is transmitted wirelessly is sent utilising the IEEE 802.15.4 protocol.

The first Tmote module that is located on the agent is programmed to receive data via the radio link and then pass it on to the UART serial output port, from where it can be fetched by the agent’s control system. The second Tmote module receives data from its serial port and sends it back to the controller PC via the radio module. On the Tmote modules, the
3.4 Positioning systems

The positioning systems were used to measure the agents position in space. Two different systems are described in the following sections, where the Ubisense system was available during the whole time the thesis was performed, while the Qualisys system was only available for a limited period of time during a visit to the company’s headquarters in Gothenburg.

3.4.1 Ubisense system

The system is manufactured by Ubisense Ltd., United Kingdom [10] and consist of several wall-mounted sensors and mobile tags that can be placed on the agents, as show in the figure below. The sensors are mounted to the walls of the area in which the experiments are performed and the agents to be located. All of the sensors are connected to each other...
through an Ethernet timing cable. One of the sensors will act as a master timing source for
to which the other sensors can synchronize themselves.
A personal computer is in contact with all the sensors through a wired network connection.
This computer gathers the calculated position data and provides a TCP/IP interface for other
clients (in this case the controller PC) that need to access the data.

![Figure 3.7: Overview of the Ubisense positioning system.](image)

To determine a tag’s position, the tag continuously emits an omnidirectional ultra-wideband
pulse which is registered by the sensors. Upon receiving this signal, the sensor will determine
the angle-of-arrival and the time-of-arrival for this pulse. Since all sensors position is known,
this data can then be used to calculate the three-dimensional position of the tag in space. The
position data is given as x, y and z values in a right-handed Cartesian coordinate system. In
addition to the ultra-wideband channel, a 2.4GHz telemetry channel is used to identify which
tag sent the pulse. Some technical data for the system in the given installation is provided in
table 3.2 below.

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Ubisense Ltd, UK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor type</td>
<td>Series 7000 IP</td>
</tr>
<tr>
<td>Tag type</td>
<td>Series 7000 compact tag</td>
</tr>
<tr>
<td>UWB frequency</td>
<td>6 – 4 GHz</td>
</tr>
<tr>
<td>Telemetry channel</td>
<td>2.4 GHz</td>
</tr>
<tr>
<td>Positioning precision</td>
<td>10 cm</td>
</tr>
<tr>
<td>Maximum update rate</td>
<td>10 Hz</td>
</tr>
<tr>
<td>Tag dimensions</td>
<td>38 x 39 x 16.5 mm³</td>
</tr>
<tr>
<td>Tag weight</td>
<td>25 g</td>
</tr>
</tbody>
</table>

**Table 3.2:** Technical data for the Ubisense positioning system.

**Ubisense interface**

All of the Ubisense systems sensors are connected to a dedicated PC running proprietary
software, as described in section [*3.4.1*](#). This PC is connected to the controller computer via
3.4 Positioning systems

A TCP/IP wired Ethernet connection. The interface task is run on the controller computer and will make the data gathered by the Ubisense system accessible from within a LabVIEW environment. A dynamic link library (DLL) was implemented in the C# .Net framework and programming language. The DLL makes extensive use of the application programming interface (API) provided by the Ubisense Ltd. company. The Ubisense system in its nature is event-driven, which means that the tag’s position is only updated when a tag is in motion and otherwise there will be no new position updates.

Because of this constraints, the interface task was designed to implement a buffer that will always provide LabVIEW with the latest known position of a tag. Whenever a tag position update is registered, the data will be written to an internal storage space from where it can be read from other programs that need access to the data. Each tag is identified by a unique, 12-digit number. From within the LabVIEW environment, the position data can be retrieved through calling the appropriate functions inside the DLL, which takes the tags identification number as an argument.

3.4.2 Qualisys system

The Qualisys Motion Capturing system is a commercially available system designed for vision-based motion capturing for post-processing and data analyses but also for real-time positioning. The system is manufactured by the Swedish company Qualisys AB based in Gothenburg. The system consists of a set of infra-red cameras that are placed around the area where the experiments with the agents are performed, so that the whole area is in their line-of-sight. The cameras continuously emit a series of infra-red flashes, that allows them to detect several retro-reflective ball shaped markers with a diameter of about 1 cm that reflect the infra-red light back to the cameras. When the cameras detect a marker, the image of several cameras placed at different positions can be compared in order to recreate the three-dimensional position of the marker in space.

![Figure 3.8: Overview of the Qualisys motion capturing system (http://www.qualisys.com).](image)

A dedicated LabVIEW application was provided by the manufacturer of the Qualisys system. The application runs periodically from within the main controller application and will on each iteration of the loop provide the controller with updated values for the agents position and orientation. For the LabVIEW interface to work, the Qualisys Track Manager application has to be run in the background on either the controller PC itself or a separate PC that is connected over a TCP/IP connection. This application communicates with the cameras and processes the data so that the agent’s positions can be determined.
### 3 Quadrotor Testbed

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Qualisys AB, Gothenburg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera type</td>
<td>Oqus 3</td>
</tr>
<tr>
<td>Marker type</td>
<td>Retro-reflective ball-shaped marker</td>
</tr>
<tr>
<td>Positioning precision</td>
<td>&lt; 1 mm</td>
</tr>
<tr>
<td>Maximum update rate</td>
<td>500 Hz</td>
</tr>
<tr>
<td>Marker dimensions</td>
<td>D = 1 cm</td>
</tr>
<tr>
<td>Marker weight</td>
<td>3 - 5 g</td>
</tr>
</tbody>
</table>

**Table 3.3:** Technical data for the Qualisys positioning system.

### Six degrees-of-freedom bodies

Alongside the ability to detect a single markers position in a world coordinate frame, the Qualisys system can also be used to bundle four or more markers that are attached to an agent to form a rigid body. Within the Qualisys software package, a local coordinate system can be assigned to the bundled markers and the system can then also determine the three orientation angles (pitch, roll, and yaw) for the rigid body in real-time. This is a very useful feature since a lot of extra states of the agent can be measured without using additional sensors, allowing for more precise control.

![Figure 3.9](image-url)  
*Figure 3.9:* Screenshot from Qualisys Track Manger software highlighting the global (bottom) and local coordinate system of a 6 DoF body (upper right) alongside the orientation angles.
4

Quadrotor dynamic Model

4.1 Modelling

In this chapter, a dynamic model for the quadrotor platform described in chapter 2 is derived. A non-linear model will be given and a linearization performed that allow for linear controller design in the following chapters. The derivation follows a method proposed by N. Michael, D. Mellinger et al. in [11] at University of Pennsylvania, however it is modified to account for the fact that in this case the desired value for the three orientation angles are used as an input instead of using the rotational speed of the four motors as inputs. These desired values of the angles are fed to the internal controller of the quadrotor which will stabilize it at the desired value.

4.2 Model derivation

Let us consider a world coordinate frame $W$ and a body coordinate frame $B$ that is attached to the center of gravity of the quadrotor platform, as described in chapter 2. The two coordinate frames are shown in 4.1. Here, the positive direction of the body frame $x$-axis is chosen so that it will point towards the forward flight direction of the quadrotor, while the positive $y$-axis will point to the left and the $z$-axis directly upwards if the quad is level ($z$-axis of world and body frame parallel). The orientation of the quadrotor in the world frame is modelled by using Z-X-Y Euler angles which are given by $\psi$, the rotation around the $z$-axis, $\theta$ for the rotation around the $x$-axis and $\phi$ for the $y$-axis rotation.

For transforming from body to world coordinate frame, the following rotational matrix is used ($c\theta = \cos \theta, s\phi = \sin \phi$ and similarly for all other angles):

$$
R = \begin{bmatrix}
 c\psi c\theta - s\phi s\psi s\theta & -c\phi s\psi & c\psi s\theta + c\theta s\phi s\psi \\
 c\theta s\psi + c\psi s\phi s\theta & c\phi c\psi & s\psi s\theta + c\psi c\theta s\phi \\
 -c\phi s\theta & s\phi & c\phi c\theta
\end{bmatrix}
$$

The forces acting on the center of mass of the quadrotor, which are the thrust generated by the propellers ($F_1, \ldots, F_4$) and gravity, can be modelled with the following equation:

$$
m \begin{pmatrix}
 \ddot{x} \\
 \ddot{y} \\
 \ddot{z}
\end{pmatrix} = \begin{pmatrix}
 0 \\
 0 \\
 -mg
\end{pmatrix} + R \begin{pmatrix}
 0 \\
 0 \\
 \sum F_i
\end{pmatrix} \quad (4.1)
$$
After multiplication with $R$:

$$
\ddot{x} = \sum \frac{F_i}{m} (\cos \psi \sin \theta + \cos \theta \sin \phi \sin \psi) \quad (4.2)
$$

$$
\ddot{y} = \sum \frac{F_i}{m} (\sin \psi \sin \theta - \cos \psi \cos \theta \sin \phi) \quad (4.3)
$$

$$
\ddot{z} = \sum \frac{F_i}{m} \cos \phi \cos \theta - g \quad (4.4)
$$

Our inputs are the desired values of the three orientation angles. The dynamics between the desired value and the actual value of an orientation are estimated as a first-order transfer function.

$$
\dot{\theta} = k_{\theta} (\theta^{des} - \theta) \quad (4.5)
$$

$$
\dot{\phi} = k_{\phi} (\phi^{des} - \phi) \quad (4.6)
$$

$$
\dot{\psi} = k_{\psi} (\psi^{des} - \psi) \quad (4.7)
$$

The thrust $F_i$ generated by a propeller depends on its rotational velocity $\omega_i$ and is governed by the following equation

$$
F_i = k_F \omega_i^2 \quad (4.8)
$$

The total thrust is given by:

$$
\sum F_i = k_F (\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) \quad (4.9)
$$

When a change in the desired value for $\theta^{des}, \phi^{des}$ occurs, the quadrotor embedded control system will change the rotational velocity of two opposing propellers (e.g. $\omega_1, \omega_3$), where it
increase one and decrease the other by equal amounts ($\Delta \omega_1 + \Delta \omega_3 = 0$). This increase and decrease in $\omega_i$ of the opposing propellers will be very small compared to the total rotational velocity. Thus, we can approximate that [4.8] and also [4.9] are linear for this small change. In conclusion, the total thrust will remain unchanged when manoeuvring (altering $\phi, \theta, \psi$) the quadcopter.

The final input variable for the model is chosen as $\omega_{\text{tot}}^{\text{des}}$, which is the desired rotational velocity for all four propellers at level flight ($\theta, \phi, \frac{d\psi}{dt} = 0$):

$$\omega_{\text{tot}} = \omega_1 = \omega_2 = \omega_3 = \omega_4$$  \hspace{1cm} (4.10)

This corresponds to a certain total thrust amount which is given by:

$$\sum F_i = 4k_F\omega_{\text{tot}}^2$$ \hspace{1cm} (4.11)

By following the conclusions above, it was shown that whilst both a change in $\theta^{\text{des}}, \phi^{\text{des}}$ and $\omega_{\text{tot}}^{\text{des}}$ will cause the propellers to rotate at different velocities, $\omega_{\text{tot}}^{\text{des}}$ can be used as an independent input parameter if it is seen as the rotational velocity of all four propellers that corresponds to a certain total thrust amount at level flight. As was shown, this total thrust amount will not change if $\theta, \phi, \psi$ change.

Finally, the dynamics between the desired and the actual rotational velocity are also estimated by a first order system [11]

$$\dot{\omega}_{\text{tot}} = k_\omega(\omega_{\text{tot}}^{\text{des}} - \omega_{\text{tot}})$$ \hspace{1cm} (4.12)

In summary, these are states that govern the quadrotor, where $\theta^{\text{des}}, \phi^{\text{des}}, \psi^{\text{des}}, \omega^{\text{des}}$ are considered as inputs:

$$\ddot{x} = \frac{4k_F\omega_{\text{tot}}^2 m}{m} (\cos \psi \sin \theta + \cos \theta \sin \phi \sin \psi)$$

$$\ddot{y} = \frac{4k_F\omega_{\text{tot}}^2 m}{m} (\sin \psi \sin \theta - \cos \psi \cos \theta \sin \phi)$$

$$\ddot{z} = \frac{4k_F\omega_{\text{tot}}^2 m}{m} \cos \phi \cos \theta - g$$

$$\dot{\theta} = k_\theta(\theta^{\text{des}} - \theta)$$

$$\dot{\phi} = k_\phi(\phi^{\text{des}} - \phi)$$

$$\dot{\psi} = k_\psi(\psi^{\text{des}} - \psi)$$

$$\dot{\omega}_{\text{tot}} = k_\omega(\omega_{\text{tot}}^{\text{des}} - \omega_{\text{tot}})$$
4.2.1 Non-linear model

The derivation presented in the foregoing section can be summarized in the following non-linear state-space equations, with \( \mathbf{u} = (\theta_{\text{des}}, \phi_{\text{des}}, \psi_{\text{des}}, \omega_{\text{des}}) \) as the input vector:

\[
\dot{\mathbf{x}} = \begin{pmatrix}
\dot{x} \\
\dot{v}_x \\
\dot{y} \\
\dot{v}_y \\
\dot{z} \\
\dot{v}_z \\
\dot{\theta} \\
\dot{\phi} \\
\dot{\psi} \\
\dot{\omega}_{\text{tot}}
\end{pmatrix} = \begin{pmatrix}
v_x \\
\frac{4k_F\omega_{\text{tot}}^2}{m} (\cos \psi \sin \theta + \cos \theta \sin \phi \sin \psi) \\
v_y \\
\frac{4k_F\omega_{\text{tot}}^2}{m} (\sin \psi \sin \theta - \cos \psi \cos \theta \sin \phi) \\
v_z \\
\frac{4k_F\omega_{\text{tot}}^2}{m} \cos \phi \cos \theta - g \\
k_\theta (\theta_{\text{des}} - \theta) \\
k_\phi (\phi_{\text{des}} - \phi) \\
k_\psi (\psi_{\text{des}} - \psi) \\
k_\omega (\omega_{\text{des}} - \omega_{\text{tot}})
\end{pmatrix}
\]

4.2.2 Linearized model

To linearize the model, at first the equilibrium points of the system corresponding to \( \dot{\mathbf{x}} = 0 \) have to be found.

\[
\dot{\mathbf{x}} = 0 = \begin{pmatrix}
v_x \\
\frac{4k_F\omega_{\text{tot}}^2}{m} (\cos \psi \sin \theta + \cos \theta \sin \phi \sin \psi) \\
v_y \\
\frac{4k_F\omega_{\text{tot}}^2}{m} (\sin \psi \sin \theta - \cos \psi \cos \theta \sin \phi) \\
v_z \\
\frac{4k_F\omega_{\text{tot}}^2}{m} \cos \phi \cos \theta - g \\
k_\theta (\theta_{\text{des}} - \theta) \\
k_\phi (\phi_{\text{des}} - \phi) \\
k_\psi (\psi_{\text{des}} - \psi) \\
k_\omega (\omega_{\text{des}} - \omega_{\text{tot}})
\end{pmatrix}
\]

The equilibrium points were chosen corresponding to the quadrotor being in level flight at a certain fixed altitude. Note that \( \theta, \phi \) have to be zero for the quadrotor to be in an equilibrium
4.2 Model derivation

state, while \( \psi \) can have any value (\( \psi = 0 \) was chosen for convenience).

\[
x_0 = \begin{pmatrix}
x^0_x \\
y^0_x \\
z^0_x \\
v^0_y \\
v^0_z \\
\theta^0 \\
\phi^0 \\
\psi^0 \\
\omega^0_{\text{tot}}
\end{pmatrix} = \begin{pmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{pmatrix}, \quad u_0 = \begin{pmatrix}
\theta^0_{\text{ref}} \\
\phi^0_{\text{ref}} \\
\psi^0_{\text{ref}} \\
\omega^0_{\text{des},0}
\end{pmatrix} = \begin{pmatrix}
0 \\
0 \\
0 \\
0
\end{pmatrix}
\]

To get the linearised system, the partial derivative matrices \( A, B \) have to be derived, where the partial derivatives are taken with respect to \( x, u \), respectively.

\[
A = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

Evaluated at \( x_0 \), \( A = \)

\[
\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \rho & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{\frac{gm}{4k_F}} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -k_\theta & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -k_\phi & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -k_\psi & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -k_\omega & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \rho
\end{pmatrix}
\]
4 Quadrotor dynamic Model

\[ B = \begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
k_\theta & 0 & 0 & 0 \\
0 & k_\phi & 0 & 0 \\
0 & 0 & k_\psi & 0 \\
0 & 0 & 0 & k_\omega
\end{pmatrix} \]

The linearized state-space system is now represented by the following equations:

\[ \dot{\tilde{x}} = x - x^0 \quad (4.13) \]
\[ \dot{\tilde{u}} = u - u^0 \quad (4.14) \]
\[ \dot{\tilde{x}} = A\tilde{x} + B\tilde{u} \quad (4.15) \]
In this chapter, the first section will deal with the synthesis and design of PID and LQ optimal controllers. A review of the general theory behind these control strategies will be given and then the application to the quadrotors will be described in detail. In the second section of this chapter, the controllers will be evaluated by performing simulation in Matlab and Simulink, and by studying real-world results from using the quadrotor testbed.

The goal of the controllers described in the following sections is to stabilize the quadrotor at a given coordinate reference in space \((x = x_r, y = y_r, z = z_r)\) and also to move it there from an arbitrary zero state. Since these are the first experiments with this quadrotor platform, no "hard" goals in terms of performance specifications will be given, but instead safe and rather slow-moving flight dynamics with conservative parameter tuning will be favoured. This will allow to avoid crashes and provide a baseline for further work with the intention to improve control performance.

5.1 PID control

5.1.1 Theory

In PID or proportional-integral-derivative control the process variable \(y(t)\) that is to be regulated and the desired value or setpoint \(r(t)\) are subtracted to form the control error \(e(t) = y(t) - r(t)\). The PID controller uses this error as its input and calculates the output \(u(t)\) by applying proportional, integral and derivative terms to the error and summing the results.

\[
u(t) = K_pe(r) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt}e(t) \tag{5.1}\]

The proportional term will produce an output that is proportional to the error. However, pure proportional control will create a steady-state error or droop that can be countered through using an integral action. This term will sum the error over time and thereby eliminate the steady-state error. The derivative term is used to slow the rate of change of the controller output and thereby reduce the magnitude of an overshoot caused by the integral term. The parameters \(K_p, K_i\) and \(K_d\) are used to tune the controller and adapt it to different processes. If a derivative or integral term is not desired, the appropriate constants can be set to zero and pure PD or PI controllers can be achieved. Different methods can be used to find suitable values for these parameters, such as for example Ziegler-Nichols [12].
5 Controller design

5.1.2 Design

As mentioned in the foregoing section, the goal was to control the quadrotors position in space using the four inputs specified in chapter 4 (desired values for pitch, roll, yaw and rotational velocity of the four propellers). If the yaw angle $\psi$ is kept zero, so that the $x_B$- and $y_B$-axis of the quadrotor body frame is aligned with the corresponding axis of the global coordinate frame ($x_W$, $y_W$, figure 4.1), the desired value for the pitch angle $\theta_{des}$ can be used to control the quadrotors displacement on the $x_W$-axis and similarly the roll angle $\phi_{des}$ to control the position on the $y_W$-axis. This can be verified by comparing with the model given in chapter 4. If there are only small deviations from level flight (such as when manoeuvring slowly), the linearized model from section 4.2.2 can be assumed to be valid and there will be no cross-coupling between either one of the input angles $\theta_{des}$, $\phi_{des}$ and the displacement in corresponding output axis.

To fully control the quadrotor, four independent PID controllers are used: one for the height, one for yaw angle and one for x- and y displacement, respectively. The yaw angle $\psi$ is governed by a simple first-order differential equation that already contains an integral term (from the non-linear model in chapter 4).

$$\dot{\psi} = k_{\psi}(\psi_{ref} - \psi) \quad (5.2)$$

Since there is already a integral term, a proportional controller of the following form is used to control the yaw.

$$\psi_{ref}(t) = u_1(t) = K_1 e_\psi(t) \quad (5.3)$$

The flight height or z-axis displacement of the quadrotor is governed by the following equations ($\theta, \phi = 0$).

$$\ddot{z} = \frac{4k_F \omega_{tot}^2}{m} - g \quad (5.4)$$

$$\dot{\omega}_{tot} = k_\omega (\omega_{des}^{tot} - \omega_{tot}) \quad (5.5)$$

Here, a full PID controller with a constant additive term is used.

$$\omega_{tot}^{des}(t) = u_2(t) = \omega_0 + K_{p,\omega} e_z(t) + K_{i,\omega} \int_0^t e_z(\tau)d\tau + K_{d,\omega} \frac{de_z(t)}{dt} \quad (5.6)$$

The additive term $\omega_0$ is chosen so that it will cancel out the gravity $g$ in equation 5.4. The integral term is used to account for long-term fluctuations in battery supply voltage of the
quadrotor and the derivative term to smoothen the response to set-point changes and to reduce a possible overshoot.

The x- and y-axis dynamics of the quadrotor are given as follows ($\psi = 0$).

\[
\ddot{x} = \frac{4kF_\omega^2}{m} \sin \theta \\
\ddot{y} = -\frac{4kF_\omega^2}{m} \cos \theta \sin \phi \\
\dot{\theta} = k_\theta (\theta_{\text{des}} - \theta) \\
\dot{\phi} = k_\phi (\phi_{\text{des}} - \phi)
\] (5.7) (5.8) (5.9) (5.10)

Since an (dual-)integrator term is already present in differential function that connects input and output, two PD controllers are used to control the movement of the quadrotor in the x- and y-directions, respectively. Again, the derivative term will ensure a smoother action. The proportional part of the controller will make the quadrotor tilt more in the respective direction the further it is away from the setpoint. Since it is not desirable for the quadrotor to tilt too much (it would move very fast or even crash if $|\theta, \phi| > \frac{\pi}{2}$ since there is no upwards lifting force any more), a saturation for $\theta_{\text{des}}, \phi_{\text{des}}$ was implemented when programming the controller. The value of the upper and lower boundary is also a design parameter.

\[
\theta_{\text{des}}(t) = u_3(t) = K_{p,\theta} e_x(t) + K_{d,\theta} \frac{d}{dt} e_x(t) \\
\phi_{\text{des}}(t) = u_4(t) = K_{p,\phi} e_y(t) + K_{d,\phi} \frac{d}{dt} e_y(t)
\] (5.11) (5.12)

($e_\psi, e_x, e_y, e_z$ are the errors in $\psi, x, y, z$, respectively)

## 5.2 LQG control

### 5.2.1 Theory

Linear-Quadratic-Gaussian or LQG control is the combination of Linear-Quadratic-Regulator (LQR) and a Kalman filter to estimate the full state from the measurable outputs. Both the theory and design of the controller follow the derivation presented in [13]. To design a controller, consider the following linear state-space representation of a plant where all states can be measured.

\[
\dot{x} = Ax + Bu; \quad (5.13) \\
y = x; \quad (5.14) \\
z = Gx + Hu; \quad (5.15)
\]

Here, $\dot{x}$ is the full state of the plant, $y$ is the output, i.e. in this case all states can be measured and $z$ corresponds to the controlled output that controller should minimize. Now, the optimal controller can be found by minimizing the criteria as follows.

\[
J_{\text{LQR}} = \int_0^\infty z'(t)Qz(t) + \rho u'(t)Ru(t)dt
\] (5.16)
5 Controller design

Now, since the whole state is known, an optimal state-feedback LQR controller is of the form

$$u = -Kx. \quad (5.17)$$

Here, $K$ is given by

$$K = (H'QH + \rho R)^{-1}(B'P + H'QG), \quad (5.18)$$

where $P$ is the positive-definite solution to the Algebraic Riccati Equation

$$A'P + PA + G'QG - (PB + G'QH)(H'QH + \rho R)^{-1}(B'P + H'QG) = 0. \quad (5.19)$$

Since not all plants allow direct measurements of all states, let us now consider one with a state space model

$$\dot{x} = Ax + Bu + \bar{B}d; \quad (5.20)$$
$$y = Cx + n. \quad (5.21)$$

Here, only the output $y$ can be directly measured and the system is also affected by a disturbance $d$ and the measurement noise $n$. To overcome this limitation, the non-measurable states can be estimated using a Kalman filter and the optimal feedback matrix gain $K$ can be applied to the estimated state $\hat{x}$, forming an LQG controller.

$$\dot{\hat{x}} = (A - LC - BK)\hat{x} + Ly \quad (5.22)$$
$$u = -K\hat{x} \quad (5.23)$$

To find the optimal state estimator, the matrix gain $L$ that minimizes the expected value of the estimation error $e = x - \hat{x}$ has to be found:

$$J_{LQG} = \lim_{t \to \infty} E[||e(t)||^2] \quad (5.24)$$

If $d(t)$ and $n(t)$ are zero-mean Gaussian noise processes with the following power spectrum

$$S_d(\omega) = Q_N, \quad S_n(w) = R_N, \quad \forall \omega,$$

then the optimal estimator gain $L$ is given by

$$L = PC'R_N^{-1}. \quad (5.25)$$

To determine $P$, the positive-definite solution to the Algebraic Riccati Equation

$$AP + PA' + \bar{B}Q_N\bar{B}' - PC'R_N^{-1}CP = 0 \quad (5.26)$$

has to be found.
5.2.2 Design

To design an LQG controller, the linearized model that is presented in section 4.2.2 is used.

\[ \dot{x} = Ax + Bu \]  \hspace{1cm} (5.27)

The inputs are the desired values of three orientation angles and the throttle at the equilibrium. The measurable output is chosen as the three position coordinates of the quadrotor’s center of gravity in the world coordinate frame plus the three orientation angles.

\[ y = Cx = \begin{pmatrix} x \\ y \\ z \\ \theta \\ \phi \\ \psi \end{pmatrix} \]

In practice this implicates that the orientation angles have to be measured, which can be achieved by either using the 6-Degrees-of-Freedom measuring functionality of the Qualisys positioning system (section 3.4.2) or by retrieving the output of the quadrotors internal gyroscope over the bi-directional wireless link (section 3.3).

To implement the controller and calculate the estimator and optimal feedback matrices, Matlab R2011 by MathWorks was used. The linearized model was programmed into Matlab and used as a parameter for the `kalman()` and `lqr()` functions that calculate the $L$ and $K$ matrices, respectively.
6 Evaluation and Results

In this chapter, an evaluation of the controllers and the testbed will be presented. Both the PID and LQR controllers introduced in chapter 5 were tested using the simulation software MathWorks Simulink [14]. The PID controller was also tested in a real-world environment using the actual quadrotor and testbed, but the LQR controller could not be tested due to limited availability of the Qualisys system. All results that were obtained will be presented in this chapter.

6.1 Simulation

6.1.1 PID

The PID controllers described in section 5.1 were implemented in Simulink together with the non-linear model as seen in section 4.2.1. A block-diagram of the Simulink implementation is shown in figure 6.2. In the figure, the four independent PID controllers acting on each of the inputs can be seen. For lateral movement, the references are given in $x$ and $y$ coordinates, while the controller acts upon the corresponding orientation angels ($\theta$ and $\phi$, respectively) that directly influences the desired output.

![Figure 6.1: Simulink block diagram of the PID controllers and the non-linear quadrotor model.](image-url)
Figure 6.2: PID step-response simulation results for a quadrotor.

The system was simulated for 15 seconds with the default numerical solver and parameters in the Simulinks settings. The resulting step-response are plotted in figure 6.2 for a step input in \( x, y \) and \( z \) references. The responses in \( x \) and \( y \) direction (upper two graphs), which both show the results of a set-point change from 0 to 2m, are exactly the same because of the symmetry of the quadrotor. The lower graph shows the response for a change in the \( z \) setpoint from 0 to 1.5m.

6.1.2 LQR

For the LQR controller, the simulation was also performed in Simulink according to the controller given in 5.2. A block-diagram of the implementation is given in figure 6.3 where the LQR controller with state estimation, the non-linear quadrotor model and the feedback connection can be seen. The constant that is added after the controller (labelled "Constant" in the figure) contains the a vector with all input values when the state of the quadrotor has reached the setpoint (see [13]). The simulation was again performed for 15 seconds using standard Simulink simulation settings and numerical solver. The resulting graphs are shown in figure 6.4. For \( x \) and \( y \) direction they represent the response to a step in the reference from 0 to 2m. Here, again this two responses are similar to each other because of the symmetry of the quadrotor. For the \( z \) response, the step was from 0 to 1.5m and this can be seen in the last graph of the figure.
6.1 Simulation

Figure 6.3: Simulink block diagram of the PID controllers and the non-linear quadrotor model.

Figure 6.4: LQR step-response simulation results for a quadrotor.
6.2 Testbed experiments

In this section, the results from performing experiments with the actual quadrotor platform (chapter 2) and testbed (chapter 3) will be presented. The control algorithm that was implemented was based on four PID controllers for each of the inputs of the quadrotor, as described in section 5.1. The PID controllers where implemented in LabVIEW and the block-diagram of the code is given in [6.5]. Three PID controllers for altitude and position control and one pure proportional controller for the yaw can be seen. The output of the PIDs for later controller have been limited by a saturation because otherwise the commanded input angle might get too large, which will cause the quadrotor to tilt very far and thereby move too fast for safe operation. For these experiments, the Qualisys positioning system (section 3.4.2) was used that allowed for direct measurement of the yaw angle. The position and angle data was processed by the PID controllers and the actuation commands where sent over the wireless link (section 3.3) to the quadrotors. The resulting graphs that are directly captured from the Qualisys data and imported into Matlab are shown in figure 6.6.

![LabVIEW block diagram of the data acquisition from the Qualisys positioning system, PID controllers and the wireless actuation system interface.](image)

Figure 6.5: LabVIEW block diagram of the data acquisition from the Qualisys positioning system, PID controllers and the wireless actuation system interface.
The three graphs shown in the figure represent the result of a step change in the set-point of the reference value. For the upper graph, the reference in $x$ was changed from 0 to 1m. For the middle graph, the $y$ set-point was changed from $-0.5m$ to 1m and the last graph shows a take-off procedure, where the quadrotor was sitting on the ground and the reference was increased from 0 to 1m, causing it to lift.

Figure 6.6: Quadrotor testbed step-response experiment results.
Conclusion and further work

In this thesis, a testbed for performing multi-agent control experiments with quadrotors was designed. Parts of the testbed where a controller PC, a bi-directional wireless link and an interface to a positioning system. The software on the controller PC that was implemented allows to access all data that is necessary for own control algorithms to implemented and thereby provides a framework for other people that want to perform practical experiments. It’s functionality was thoroughly tested and results from performing actual experiments are presented as a part of this thesis. Furthermore, a model of the quadrotor was derived. This model was implemented into simulation and the results show that it is a viable model that allows to predict the behaviour of a the quadrotor in the testbed. LQR and PID controller where designed based on the model that was derived. This controller where test using both simulation and the testbed, thereby further proving the functionality of the testbed.

For further work in this area and with the testbed, there are several recommendations to be made:

- Cooperative control: Use the testbed to implement cooperative control between ground and aerial vehicles.
- Parameter estimation: Estimation of the parameters that depend on the actual quadrotor platform that is used in the model.
- LQR controller: Test the implementation of the LQR controller with the testbed.
- Increase autonomicity: To make the quadrotor truly autonomous, the controllers for position could be implemented directly on the quadrotor (instead of on the controller PC) and only the desired set-point transmitted to them.
References


[9] *TinyOS operating system*: http://www.tinyos.net/


[11] "*The GRASP Multiple Micro UAV Testbed*": N.Michael, D.Mellinger, Q. Lindsey, V. Kumar, University of Pennsylvania 2005

