Control and image processing algorithms for quadrotor coverage applications

Catarina da Cruz Coelho da Silva

Thesis to obtain the Master of Science Degree in Electrical and Computer Engineering

Supervisor(s)
Prof. João Paulo Salgado Arriscado Costeira
Dr. Manuel Ricardo de Almeida Rodrigues Marques
Prof. Duarte José Guerreiro Tomé Antunes

Examination Committee
Chairperson: João Fernando Cardoso Silva Sequeira
Supervisor: Prof. Duarte José Guerreiro Tomé Antunes
Members of the Committee: Prof. Pedro Manuel Urbano de Almeida Lima
Prof. Rita Maria Mendes de Almeida Correia da Cunha

November 2016
Abstract

The main goal of this work is to study the exploration scenario in indoor environments for a low cost unmanned aerial vehicle. A new formulation is presented for this problem and two approximation algorithms are compared. The chosen platform is Parrot’s AR Drone 2.0 and the implementation is made in a bottom-up approach, from quadrotor stabilization, through image-based localization until the final coverage path planning study. Several satisfactory results are obtained from each of the modules, in particular the work achieved in image acquisition puts together an analysis of different methods that can be used in the chosen device, which was submitted to Mathworks.

Keywords: Exploration, Coverage Path Planning, Quadrotor control, Image based localization, Computer vision
I would first like to thank my thesis advisors, Dr. Manuel Marques and Prof. João Paulo Costeira from Instituto Superior Técnico at Universidade de Lisboa, and Prof. Duarte Antunes from Eindhoven University of Technology. They consistently allowed this paper to be my own work, but steered me in the right direction whenever they thought I needed it.

This thesis was supported by Instituto Superior Técnico and partially by Eindhoven University of Technology, and so I thank our colleagues from Control System Technology group of the Eindhoven University of Technology who provided a basis for the work, insights and expertise that greatly assisted the research.

Finally, I must express my very profound gratitude to all those who accompanied me in this process, for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Catarina Silva
# Contents

Abstract

Acknowledgments

Contents

List of figures

List of tables

Acronyms

Notation

1 Introduction
  1.1 Motivation
  1.2 Thesis Statement
  1.3 Related work
  1.4 Contributions
  1.5 Outline

2 Background
  2.1 Parrot AR Drone 2.0
  2.2 Quadrotor dynamics
  2.3 Control techniques
  2.4 Camera model
  2.5 Visual marker identification
  2.6 Vision-based position control
  2.7 Localization and mapping
  2.8 Exploration
  2.9 Optimization and complexity

3 Quadrotor control
  3.1 Sensor calibration
  3.2 PWM calibration
  3.3 State definition
  3.4 State estimation
  3.5 Control scheme
  3.6 Outer loop
  3.7 Take off and landing

4 Image processing
  4.1 Image acquisition
## Contents

4.2 Camera calibration .......................................................... 49  
4.3 ArUco visual markers ..................................................... 50  
4.4 Mapping ......................................................................... 52  
4.5 Pose extraction ............................................................... 53  
4.6 Code flow and organization ................................................. 54  

5 Exploration and coverage ...................................................... 55  
5.1 Mapping using confidence levels ......................................... 55  
5.2 Exploration and coverage scenarios ..................................... 57  
5.3 Offline coverage algorithms ............................................... 58  

6 Simulations and results .......................................................... 61  
6.1 Sensor calibration .............................................................. 61  
6.2 PWM modelation ............................................................. 61  
6.3 Kalman filter ................................................................. 63  
6.4 Quadrotor stabilization ...................................................... 64  
6.5 Image processing ............................................................. 70  
6.6 Vision outer loop ............................................................ 74  
6.7 Covering algorithms ........................................................ 75  

7 Conclusions ..................................................................... 79  
7.1 Discussion and future work ............................................... 79  

Bibliography .................................................................. 81  

Appendix A PWM measurements ............................................ A-1  
Appendix B Image acquisition measures ................................. B-1
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Parrot AR Drone 2.0 in flight</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Parrot AR Drone 2.0 sensor axis</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Quadrotor body frame for the plus configuration</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>Quadrotor body frame for the cross configuration</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>Inducing roll movement through rotor actuation</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Inducing pitch movement through rotor actuation</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>Inducing yaw movement through rotor actuation</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>Quadrotor frames</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>Planar calibration pattern undistorted</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>Different ArUco markers with different sizes identified</td>
<td>17</td>
</tr>
<tr>
<td>11</td>
<td>Aruco detection process - adaptive threshold</td>
<td>18</td>
</tr>
<tr>
<td>12</td>
<td>Aruco with draw axis computed from pose estimation</td>
<td>18</td>
</tr>
<tr>
<td>13</td>
<td>Relation between frames for PBVS if the object is an ArUco marker</td>
<td>21</td>
</tr>
<tr>
<td>14</td>
<td>Point cloud of a room viewed with PLC viewer</td>
<td>22</td>
</tr>
<tr>
<td>15</td>
<td>DFS result in path search</td>
<td>28</td>
</tr>
<tr>
<td>16</td>
<td>BFS result in path search</td>
<td>28</td>
</tr>
<tr>
<td>17</td>
<td>Dijkstra’s optimal path example</td>
<td>29</td>
</tr>
<tr>
<td>18</td>
<td>Best-first search example</td>
<td>30</td>
</tr>
<tr>
<td>19</td>
<td>A* search example in grids</td>
<td>31</td>
</tr>
<tr>
<td>20</td>
<td>Accelerometer calibration subsystem scheme</td>
<td>33</td>
</tr>
<tr>
<td>21</td>
<td>Calibration minimum set of positions</td>
<td>34</td>
</tr>
<tr>
<td>22</td>
<td>Roll and pitch computation from accelerometer in Simulink®</td>
<td>39</td>
</tr>
<tr>
<td>23</td>
<td>Phi LQR-tuned PD controller</td>
<td>42</td>
</tr>
<tr>
<td>24</td>
<td>Outer loop control</td>
<td>43</td>
</tr>
<tr>
<td>25</td>
<td>Position reception in the control scheme</td>
<td>43</td>
</tr>
<tr>
<td>26</td>
<td>Transforming the global position to body frame coordinates</td>
<td>43</td>
</tr>
<tr>
<td>27</td>
<td>Take off procedure for the Parrot AR Drone 2.0</td>
<td>44</td>
</tr>
<tr>
<td>28</td>
<td>Outer loop outputing angle references</td>
<td>44</td>
</tr>
<tr>
<td>29</td>
<td>Simulink® scheme to output and send image over TCP</td>
<td>45</td>
</tr>
<tr>
<td>30</td>
<td>Simulink® scheme to receive over TCP and save the image</td>
<td>46</td>
</tr>
<tr>
<td>31</td>
<td>Computing image acquisition performance results</td>
<td>46</td>
</tr>
<tr>
<td>32</td>
<td>Gstreamer pipeline elements</td>
<td>47</td>
</tr>
<tr>
<td>33</td>
<td>Rasperry Pi assembly on top of the indoor hull of Parrot AR Drone 2.0</td>
<td>48</td>
</tr>
<tr>
<td>34</td>
<td>Camera calibration camera and pattern poses</td>
<td>49</td>
</tr>
<tr>
<td>35</td>
<td>Camera calibration mean reprojection error</td>
<td>50</td>
</tr>
<tr>
<td>36</td>
<td>ArUco marker detection</td>
<td>50</td>
</tr>
<tr>
<td>37</td>
<td>ArUco map</td>
<td>52</td>
</tr>
<tr>
<td>38</td>
<td>Aruco map represented in MATLAB®</td>
<td>52</td>
</tr>
<tr>
<td>39</td>
<td>Reference frame transformations to obtain the quadrotor position</td>
<td>53</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>Code flow of the image algorithm implementation</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>Experimental Torque vs. PWM curve for the four quadrotor rotors</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>Experimental Torque vs. PWM curve for the four quadrotor rotors with raspberry on top</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>Results for $\phi$ and $\theta$ before (blue) and after (red) Kalman filter</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>Kalman filter results on the impact of moving $\theta$ in $\phi$ measurements</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>Phi and P for hover flight test</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>Theta and Q for hover flight test</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>Z and W for hover flight test</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>Forces and torques values for hover flight test</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>PWM values for hover flight test</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>Phi and P for take off and landing flight test</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>Theta and Q for take off and landing flight test</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>Z and W for take off and landing flight test</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>Forces and torques values for take off and landing flight test</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>PWM values for take off and landing flight test</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>Rasperry Pi's ArUco pose estimation accuracy for a still camera</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>Kalman filter results for the raspberry camera (1280 x 720) moving 14 cm in the x coordinate</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>Kalman filter results for the raspberry camera (640 x 480) moving 14 cm in the x coordinate</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>Camera localization results for larger map, with baseline integrated $\psi$</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>Body frame coordinate transformation and outer loop results</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>Ground coverage problem for a 6 x 6 grid</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>Ground coverage sub-optimal algorithms results for a 6 x 6 grid</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>Ground coverage random-based coverage results for a 6 x 6 grid</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>Ground coverage test in a 12 x 9 grid</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>Ground coverage best first in horizon result sequence for a 12 x 9 grid</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>Ground coverage test with different levels of confidence in the same map</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>Ground coverage results for different levels of confidence in the same map</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>Bottom-up approach</td>
<td></td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>PWM approximate modelation values</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>Kalman filter values for attitude control</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>Kalman filter values for altitude control</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>Control parameters for attitude control</td>
<td>42</td>
</tr>
<tr>
<td>5</td>
<td>Control parameters for altitude control</td>
<td>42</td>
</tr>
<tr>
<td>6</td>
<td>Best image acquisition parameters and results for Simulink® AR Drone Target</td>
<td>47</td>
</tr>
<tr>
<td>7</td>
<td>Best image acquisition parameters and results for gstreamer</td>
<td>48</td>
</tr>
<tr>
<td>8</td>
<td>Best image acquisition parameters and results for Raspberry Pi camera</td>
<td>48</td>
</tr>
<tr>
<td>9</td>
<td>Best combined image acquisition and processing results for Raspberry Pi camera</td>
<td>51</td>
</tr>
<tr>
<td>10</td>
<td>Best image acquisition parameters and results for Raspberry Pi camera</td>
<td>54</td>
</tr>
<tr>
<td>11</td>
<td>Rotor hovering point for the quadrotor without (case 1) and with a raspberry Pi 3 (case 2)</td>
<td>62</td>
</tr>
<tr>
<td>12</td>
<td>Raspberry Pi’s ArUco pose estimation accuracy for a still camera</td>
<td>70</td>
</tr>
<tr>
<td>13</td>
<td>Raspberry Pi’s ArUco pose estimation accuracy for a still moving camera</td>
<td>73</td>
</tr>
<tr>
<td>14</td>
<td>Algorithm path length and time results for ground coverage problem in small grids</td>
<td>75</td>
</tr>
<tr>
<td>15</td>
<td>Algorithm path length and time results for ground coverage problem with higher complexity</td>
<td>77</td>
</tr>
<tr>
<td>16</td>
<td>PWM calibration for quadrotor without payload (case 1)</td>
<td>A-2</td>
</tr>
<tr>
<td>17</td>
<td>PWM calibration for quadrotor with raspberry payload (case 2)</td>
<td>A-4</td>
</tr>
<tr>
<td>18</td>
<td>Image acquisition results for quadrotor front camera (case 1)</td>
<td>B-2</td>
</tr>
<tr>
<td>19</td>
<td>Image acquisition results for quadrotor front camera (case 2)</td>
<td>B-2</td>
</tr>
<tr>
<td>20</td>
<td>Image acquisition results for quadrotor front camera (case 3)</td>
<td>B-3</td>
</tr>
</tbody>
</table>
2D two-dimensional. 15, 18, 24, 49, 54, 55

3D three-dimensional. 1, 15, 18, 20, 23, 24, 49, 52, 55

AGV automated guided vehicle. 3

API application program interface. 33, 34, 45, 80

ATSP asymmetric travelling salesman problem. 26

BFS breadth first search. 28, 29

CPP coverage path planning. 2, 4, 23, 24

CSP covering salesman problem. 23, 26, 58

DFS depth first search. 28

DOF degrees of freedom. 1, 20, 24

DP dynamic programming. 24, 56

FIFO first-in-first-out. 28

FPS frames per second. 7, 46, 48

FTP File Transfer Protocol. 7

GCSP generalized covering salesman problem. 26

HD High-definition. 7

IBVS Image Based Visual Servo. 19, 21

IC integrated circuit. 6

IMU Inertial Measurement Unit. 5, 6

LIFO last-in-first-out. 28

LQR linear quadratic regulator. 12, 13, 14, 41, 42, 79

MEMS microelectromechanical system. 5, 6, 33, 34

MIMO multiple input multiple output. 11, 13

MTU maximum transmission unit. 48
Acronyms

**PBVS**  Position Based Visual Servo. 20, 21

**PWM**  pulse width modulation. 5, 34, 35, 41, 48, 61, 62, 66, 69

**QVGA**  Quarter Video Graphics Array. 7

**RTP**  Real-time Transport Protocol. 7, 46, 47

**SISO**  single input single output. 11, 13

**SLAM**  simultaneous localization and mapping. 22

**SLAM++**  simultaneous localization and mapping ++. 1, 3, 22

**STSP**  symmetric travelling salesman problem. 26

**TCP**  Transmission Control Protocol. 7, 46

**TSP**  traveling salesman problem. 23, 24, 26, 58

**UAV**  unmanned aerial vehicle. 1, 2, 3, 33

**UDP**  User Datagram Protocol. 7, 43, 46, 47
General definitions:

\[ s_\theta = \sin(\theta) \]
\[ c_\theta = \cos(\theta) \]
\[ t_\theta = \tan(\theta) \]

\[ {}^A R_B \quad \text{Rotation matrix from } \{B\} \text{ reference frame to } \{A\} \text{ reference frame} \]
\[ {}^A T_B \quad \text{Translation vector from } \{B\} \text{ reference frame to } \{A\} \text{ reference frame} \]

Reference frames:

\{A\} - inertal reference frame
\{B\} - quadrotor reference frame
\{V\} - ArUco reference frame

Angles in inertial reference frame:

\( \phi \) - roll
\( \theta \) - pitch
\( \psi \) - yaw

Velocities in body frame:

\( P = \dot{\phi} \in \{B\} \)
\( Q = \dot{\theta} \in \{B\} \)
\( R = \dot{\psi} \in \{B\} \)

Used constants:

\( g = 9.81 \quad \text{acceleration of gravity} \)
1

Introduction

1.1 Motivation

**Unmanned aerial vehicles (UAVs)**, in particular quadrotors, are becoming more and more relevant in several kinds of applications, such as infrastructure inspection and precision farming. This is a result of their aerial mobility and easy control scheme, combined with the ability of hovering at a given position, unlike other aerial platforms.

In indoor scenarios quadrotors are also a common research subject. These scenarios imply the need to assure safety for the people moving in the environment, as well as the robot’s integrity. There is also a need for indoor localization, since it is not possible to use GPS, as in most outdoor applications. This is a challenge that increases difficulty without the use of expensive sensing units.

Many applications rely also on the use of image processing, and a camera is usually part of the UAV-system. Image applications tend to be rather demanding in terms of processing, but the use of object maps, a concept that is often referred to as **simultaneous localization and mapping ++ (SLAM++)**, mitigates the need for memory and processing.

Another important subject addressed by a considerable body of research in the literature is **exploration**. It is the task of guiding a robot in such a way that it covers the entire environment with its sensors. This is a very interesting problem for quadrotors, since they can explore a three-dimensional (3D) space easily with their 6 degrees of freedom (DOF).

The problem has applications in aerial robotics that span over a wide range of areas, for example search and rescue. This subject is also interesting for terrestrial vehicles, from cleaning robots and painter robots to lawn mowers and even aquatic vehicles.

This thesis tackles the image-based position control problem, using simple control techniques and combining them with image based localization. Moreover it addresses the exploration problem, focusing on the study of coverage algorithms.
Chapter 1. Introduction

1.2 Thesis Statement

The present thesis investigates control and computer vision algorithms for autonomous UAV exploration in indoors environments. It presents an implementation in a commercially available low-cost UAV, the AR Drone 2.0 from Parrot, based on a MATLAB® toolbox - AR Drone Simulink®.

The final goal is to address the indoor exploration problem for aerial vehicles, while showing that the image-based localization techniques can be implemented without the need for highly complex and expensive hardware. The work is divided in two main topics:

1. Control and image algorithms for indoor localization and navigation of a low cost UAV
2. Coverage algorithms for exploration tasks in known or semi-unknown environments

The first topic addresses the quadrotor vision-based control, balancing the existing trade-offs when working with low cost technology, and aims to present and implement a simple modular control approach, showing its performance and the trade-offs when using image-based control with constrained resources.

The second topic tackles the coverage path planning (CPP) problem for exploration in indoor environments. Most of the previous work in the literature consider that the goal is to explore the entire space. Instead, this thesis presents a formulation where a confidence measure is defined for each area. This represents a measure of utility in the environment, in such a way that the vehicle focusses on mapping specific areas assigned more significance.

The final goal is to study and compare some existing state of the art approximation algorithms in the literature and apply them to this formulated problem for exploring an indoor unknown or partially known environment.

1.2.1 Implementation context

There are innate limitations of the chosen platform, Parrot’s AR Drone 2.0, that are important to mention at this point since they constrain what can be achieved or expected from the present work, at the implementation level:

**Motor power** The AR Drone’s motor have limited power, as such the supported weight is very limited;

**Sensor precision** As a low cost quadrotor, the sensors are not very precise and are very noisy;

**Vision range** The maximum camera resolution will affect the range seen by the quadrotor, constraining the maximum distance allowed when assessing a goal target surface/object;

**Software interface** The cameras are not available and most of its access has to be redone.

Given these limitations, the following options were taken:

**Off-board processing** Most processing is done off-board, making it mandatory to have some host station in a given radius of the robot;

**Obstacles** No obstacles are considered in the environment and no obstacle avoidance was implemented.
Chapter 1. Introduction

1.3 Related work

Multirotor aircraft, and particularly quadrotors, have been rapidly growing in popularity as a platform for robotics research. This work builds upon the dynamic models used by many others in literature, such as [19]. The practical implementation is built on top a particular control technique, the PID tuned by LQR [20] control scheme, described in 2.3.4.

For the purpose of localization and mapping, image processing algorithms are used. Since the focus of the thesis is not on object recognition a solution with visual markers is used, namely ArUco markers [14]. In most of the present work, the algorithm proposed in [27] is implemented. In particular, an implementation of SLAM++ is used, which is a closed form solution for obtaining a global map from partial observations with pairs of objects. Although the presented algorithm was shown only for a terrestrial automated guided vehicle (AGV), its implementation is equally advantageous in an UAV.

The exploration study presented in the thesis relies on the recent survey on coverage path planning [13], which presents an extensive overview of the state of the art coverage algorithms present in the literature. It also covers common applications and describe some of the challenges that remain to address in this field. One of the focused algorithms in this scope is a random based coverage [12] which is shown to work rather well in offline planning.

The initial implementation is based on the AR Drone Simulink ® libraries, that facilitate the communication with the target and its components, and the implementation was largely supported by a collaborative work with the Control System Technology group of the Eindhoven University of Technology.

1.4 Contributions

The present work has two main contributions. The first contribution is the construction of a bottom-up modular approach to implement exploration in a quadrotor. This thesis presents a strategy for each module and demonstrates its satisfactory performance in the experimental setup.

One of the main challenges is the replacement of the external cameras by a single camera. As a consequence, the work presents several image acquisition methods for the drone cameras. One of the experimented methods consists on the implementation of Simulink ® blocks that allowed image retrieving, with three phases:

- Expand Mathworks rudimentary video library with TCP communication to extract the image to the host
- Combine the video and communication blocks to increase speed
- Expand these processes with threads and allow control of internal buffers for better tuning

These implementations were submitted to Mathworks, as well as the results obtained through testing, that gave a significant insight on the drone capabilities.
Chapter 1. Introduction

A second contribution is the proposed formulation of the CPP problem, which differs from the ones found in the literature. The CPP typical formulations present a generic approach where the entire space should be mapped. This thesis presents a different formulation where a measure of utility and believe is presented as part of the problem formulation. Moreover, the world map to be covered is represented by beliefs in the interval $[0, 1]$ instead of a binary representation (covered/not covered).

This new formulation introduces two new features. First, it allows the introduction of new scenarios, where the focus is not to explore the entire world, but to explore particular areas of interest. Second, the problem increases in complexity, thus this work tries to study possible approximation algorithms and its performance on a simple scenario.

1.5 Outline

Chapter 2 presents the basic knowledge that supports the research. It starts by describing the hardware used with 2.1. A description of the quadrotor’s dynamics is then presented in 2.2, followed by an overview of used control techniques explained in 2.3. Section 2.6 explores the state of the art algorithms for vision-based control and 2.8 then introduces a high level view of the proposed work in the context of exploration. Lastly 2.9 presents the basic concepts of optimization and complexity, in the scope of high-level decision algorithms.

The following chapters, 3, 4 and 5, describe the implementation process of quadrotor stabilization, the image acquisition and vision control and the coverage problem, respectively. The implementation models, and the used settings are also presented here, building all the support and context for the results presented in chapter 6. This chapter thus presents the results drawn from each implementation and discusses them.

Finally, the work presented is concluded in chapter 7 by summarizing the problems tackled and discussing the many aspects of this thesis, with a critical analysis of the results and future work to be researched and implemented.
Background

2.1 Parrot AR Drone 2.0

The chosen platform for the implementation and test of the proposed research is Parrot’s AR Drone 2.0 [6][21]. This section presents an overview of its hardware, an introduction to the software and the communication protocols available to interact with a host station. This background supports the implementation described in chapters 3, 4 and 5.

AR Drone 2.0 has a carbon fiber airframe that can be protected by two different hulls in foam. Its system includes a microelectromechanical system (MEMS) Inertial Measurement Unit (IMU), a sonar, two cameras and two processors. Its four brushless inrunner motors of 14.5 Watts and 28500 RPM can be controlled by pulse width modulation (PWM) through a micro-controller.

PWM is used for controlling the amplitude of digital signals. It is a signal switched between on and off that simulates voltages in between the signal range. The portion of the time the signal is on versus the time that the signal is off defines the analog voltage given. The duration of ”on time” is called the pulse width.

\[
\omega_i \approx a_i \, PWM_i^2 + b_i \, PWM_i + c_i \tag{2.1}
\]

The angular velocity is related to the PWM given to the motor by approximately a quadratic equation, as shown in equation (2.1). The constants \(a_i\), \(b_i\) and \(c_i\) vary with each rotor. For a given angular velocity, the value of PWM is computed by the inverse transformation represented in equation (2.2).

\[
PWM_i = \frac{-b_i + \sqrt{b_i^2 - 4(a_i - \omega_i) \times c_i}}{2 \times c_i} \tag{2.2}
\]
AR Drone 2.0 has an **ultrasound sensor** that can be used for ground altitude measurement, through the reflection of sound waves with high frequency. The transmitter emits a short burst of sound in a particular direction, which bounces off the ground and returns to the receiver after a time interval $t$. The distance travelled $r$ based on the speed of sound $c$ is given by (2.3).

$$r = c \times t$$ \hspace{1cm} (2.3)

An important aspect of this sensor is its dead-zone: typically the sensor can not detect objects or measure distances immediately close, therefore it only starts to function properly at a given altitude.

The IMU has a **three axis gyroscope** with 2000 degrees per second precision. This sensor is small, between 1 to 100 micrometers, and when it is rotated, a small resonating mass is shifted as the angular velocity changes. This shift is converted into a low-current, a signal that can be amplified and read by a micro-controller. The sensor measures rotation around $x$, $y$, and $z$, respectively $\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}$, with its axis defined in figure 2a.

The main factor that affects the gyroscope measurements is the existing bias in each axis direction. It is the value the sensor outputs when no rotation exists and it has to be taken out of the normal output. The main challenge pertaining the bias is that it varies in time, for example with temperature differences caused by battery heating. Therefore it cannot be fully compensated for by canceling its initial value.

Another sensor available in the IMU is a **three axis accelerometer** with 50mg precision and axis as presented in figure 2b. The basic principle of operation behind a MEMS accelerometer uses the displacement of a small mass etched into the silicon surface of the integrated circuit (IC) and suspended by small beams. According to Newton’s second law of motion, $F = ma$, as an acceleration is applied to the device, a force develops which displaces the mass.

Since the value measured by the accelerometer also captures the gravity force, it is often used to calculate angles, for example, pitch and roll. Because accelerometers are more sensitive when perpendicular to gravity, the three axis accelerometer increase precision against two or one axis accelerometers. Equations (2.4) and (2.5) allow angle computation from the accelerometer, assuming a state near stillness.

$$\phi = \arctan\left(\frac{y}{\sqrt{x^2 + z^2}}\right)$$ \hspace{1cm} (2.4)

$$\theta = \arctan\left(\frac{x}{\sqrt{y^2 + z^2}}\right)$$ \hspace{1cm} (2.5)
Lastly, AR Drone 2.0 has two cameras: the first is a **forward facing** High-definition (HD) camera that can reach 30 frames per second (FPS); the second is a **bottom facing** Quarter Video Graphics Array (QVGA) camera that reaches 60 FPS. The device also supports several communication protocols that will be used throughout this thesis, namely User Datagram Protocol (UDP) [22], Transmission Control Protocol (TCP) [23], Real-time Transport Protocol (RTP) [29], File Transfer Protocol (FTP) [25] and Telnet [24]. Some security concerns are linked to FTP and telnet use [21].

### 2.2 Quadrotor dynamics

A quadrotor is a simple aerial vehicle that consists of four individual rotors attached to a rigid cross airframe. Its control is achieved by a differential control of each rotor’s generated thrust. Each pair of opposite rotors turn in the same direction: rotors 1 and 3 clockwise, and rotors 2 an 4 counterclockwise, as shown in figure 3.

![Quadrotor body frame for the plus configuration](image)

Figure 3: Quadrotor body frame for the plus configuration

The figure presents a possible reference frame, where both the $x$-axis and the $y$-axis point in the direction of rotors 1 and 2, respectively. This is referred to as the plus configuration, due to the resemblance between the relative position of the airframe regarding the $x$-axis direction with a plus signal (+). A slightly different configuration is the one shown in figure 4, where the $x$-axis points in the same direction as the camera, and the $z$-axis points downwards. This is called the cross configuration ($x$).

![Quadrotor body frame for the cross configuration](image)

Figure 4: Quadrotor body frame for the cross configuration

Using a plus configuration, a simple model of the quadrotor based on physics principles is presented in [19]. This section presents a similar dynamic model of the quadrotor using the cross configuration and introduces the equations that model its behavior. Foremost, a significant aspect of the model is the rotors’ aerodynamics, which provides insight on how the rotor speeds induce the different movements that allow the navigation of the quadrotor.
Each rotor generates a given local upward thrust, applied to the center of the rotor, $T_i$, described by equation (2.6). It also generates a local torque $Q_i$, induced by rotor drag, modelled by equation (2.7). Both variables $c_T$ ($c_T \geq 0$) and $c_Q$ are constants that can be determined from static thrust tests. They encompass the effect of air density($\rho$), rotor radius($r_i$) and disk area ($A_{r_i}$) on the quadrotor forces and torques.

$$T_i = c_T \omega_i^2$$ (2.6)  
$$Q_i = c_Q \omega_i^2$$ (2.7)

As a result of each thrust and torque a total thrust and a net moment are applied to the airframe’s center of gravity. Note that the presented equations do not account for blade flapping or other perturbations but they are, however, good approximations. The total thrust, $T_\Sigma$, is given by the sum of all thrusts, as shown in (2.8).

$$T_\Sigma = c_T \sum_{i=1}^{N} \omega_i^2$$ (2.8)

The net moment corresponds to the pitch ($\phi$), roll ($\theta$) and yaw ($\psi$) movements. Both the pitch and roll movements are very intuitive and depend almost only on the local thrusts from (2.6). To induce a roll movement, this is, to tilt the quadrotor relative to the $x$-axis, the summed thrusts at each side have to be different. As shown in figure 5, if rotors 1 and 4 generate a larger thrust summed, then the roll angle will increase, if rotors 2 and 3 generate a larger thrust summed, then it will decrease. The roll torque can then be described by equation (2.9) where $d$ is the distance from each rotor center to the airframe center.

$$\tau_\phi = c_T \frac{\sqrt{2}}{2} d (\omega_1^2 - \omega_2^2 - \omega_3^2 + \omega_4^2)$$ (2.9)

![Figure 5: Inducing roll movement through rotor actuation](image)

The pitch movement is similar to the roll movement, but about the $y$-axis. Therefore, the pitch angle increases with higher thrust values in rotors 1 and 2 and decreases with higher thrust values in rotors 3 and 4, like figure 6 shows. The pitch torque is described in equation (2.10).

$$\tau_\theta = c_T \frac{\sqrt{2}}{2} d (\omega_1^2 + \omega_2^2 - \omega_3^2 - \omega_4^2)$$ (2.10)
Chapter 2. Background

Figure 6: Inducing pitch movement through rotor actuation

The yaw movement, on the other hand, is induced by the combination of each rotor drag. Because the rotors turn in different directions, when all the rotors are turning at the same speed, the torques generated in different directions compensate and the quadrotor is still. If, however, there is a difference in the generated torques, there will be a yaw rotation, as shown in figure 7.

\[
\tau_\psi = c_Q \left( -\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2 \right)
\]  

(2.11)

Equations (2.8) to (2.11) can be transformed to an equivalent matrix form. The result is presented in equation (2.12),

\[
\begin{bmatrix}
T_{\Sigma} \\
\tau_\varphi \\
\tau_\theta \\
\tau_\psi
\end{bmatrix}
=
\begin{bmatrix}
ct & ct & ct & ct \\
\frac{\sqrt{2}}{2}dct & \frac{\sqrt{2}}{2}dct & \frac{\sqrt{2}}{2}dct & \frac{\sqrt{2}}{2}dct \\
\frac{\sqrt{2}}{2}dct & \frac{\sqrt{2}}{2}dct & \frac{\sqrt{2}}{2}dct & \frac{\sqrt{2}}{2}dct \\
-cQ & cQ & -cQ & cQ
\end{bmatrix}
\begin{bmatrix}
\omega_1^2 \\
\omega_2^2 \\
\omega_3^2 \\
\omega_4^2
\end{bmatrix}
\]

(2.12)

Figure 7: Inducing yaw movement through rotor actuation

where $\Gamma$ is a defined constant matrix that relates speeds with torques. By inverting $\Gamma$ and setting $T_{\Sigma} = mg$ it is possible to find the adequate values of $\omega_i$ that achieve a hovering state.

Another important aspect of the quadrotor model is the set of motion equations of the airframe, which requires a number of reference frames to be defined. Let $\{A\}$ describe a right-hand fixed inertial frame. The position and linear velocity of the vehicle are described by vectors in this frame as shown in equations (2.13) and (2.14).
\( \zeta = (x, y, z) \in \{A\} \)  
\( v = (\dot{x}, \dot{y}, \dot{z}) = (u, v, w) \in \{A\} \)  

Let \( \{B\} \) be a right-hand body-fixed frame for the airframe. Then the orientation of the airframe can be described by a rotation matrix \( {^A R_B} \). An intermediate matrix \( \{E\} \) can be designed to represent this rotation, as shown in figure 8. \( {^A R_B} \) is written as shown in equation (2.15), considering the \( Z - X - Y \) Euler angles.

\[
{^I R_B} = \begin{bmatrix}
  c\phi c\theta - s\phi s\theta & -c\phi s\theta & c\phi c\theta + c\phi s\phi s\psi \\
  c\phi s\phi - c\phi s\theta & c\phi c\theta & s\phi s\theta - c\phi c\phi s\psi \\
  -c\phi s\theta & s\phi & c\phi c\theta
\end{bmatrix}
\]  
(2.15)

Figure 8: Quadrotor frames

The three angles involved, \( \phi, \theta \) and \( \psi \), are defined in the inertial reference frame \( \{A\} \), while their velocities are defined in the body-frame \( \{B\} \), as shown in equations (2.16) and (2.17).

\( (\phi, \theta, \psi) \in \{A\} \)  
\( \Omega = (\dot{\phi}, \dot{\theta}, \dot{\psi}) = (p, q, r) \in \{B\} \)  

The physics principles that relate the position and orientation with its corresponding velocities are shown in equations (2.18) to (2.21), where \( F \) and \( \tau \) represent the non-conservative forces and moments, defined in the body-frame \( \{B\} \).

\( \dot{\zeta} = v \)  
(2.18)

\( m\ddot{v} = m g z + RF \)  
(2.19)

\( \dot{R} = R \Omega X \)  
(2.20)

\( I\dot{\Omega} = -\Omega I \Omega + \tau \)  
(2.21)
Chapter 2. Background

2.3 Control techniques

This section describes the control techniques supporting chapter 3, providing the background for its use and tuning. It presents each method in a generic way, to give a broad overview of the theory that supports them.

2.3.1 PID

A PID controller is a classic method of control through a feedback loop, which continuously updates an error value \( e \) with the difference between a tracking reference and the corresponding measured value. The error is given as feedback and used to compute the next control input \( u \), through equation (2.22). PID stands for Proportional, Integral, Derivative, corresponding to the three gains that can be tuned in the control law, \( K_p \), \( K_i \), and \( K_d \).

\[
    u(t) = K_p e(t) + K_i \int_0^t e(\tau) \, d\tau + K_d e'(t) \tag{2.22}
\]

Increasing the proportional action decreases the steady-state error, but high gains can trigger a non-desired oscillatory behaviour. A small gain can lead the target to the desired value, but it is normally slow. Single proportional control is not adequate when the system has too much delay or when it needs a fast response, like the attitude control of the quadrotor.

The integral action guarantees that the process output agrees with the reference in steady state. By itself it normally doesn’t result in a stable system, so it is almost always used with proportional action. An important effect is integrator windup, which happens if a system operating over a wide range of conditions reaches the actuator limits, causing the system to run in open loop as the actuator is saturated. It may take a long time before the output comes inside the saturation range, causing large transients.

The derivative action causes the oscillations in the system to be more damped as it increases. However, it is highly dependent on the sampling rate, which should be high to reproduce the changes as closely as possible. It will also scale noise, thus the term should not be too high, to avoid instability. In general, these gains can be tuned to obtain a good control, but a trade-off will exist between speed and stability, since a fast system can become too nervous, and a slow system might not fulfill the goal in useful time.

2.3.2 Space state control

The classical control theory where the PID is inserted is better applied to single input single output (SISO) systems. However, the quadrotor is a multiple input multiple output (MIMO) system. For controlling MIMO systems, a common approach is state space control.

The concept of state in a dynamic system refers to a set of variables that fully describe the system. It should be a minimum set, to avoid unnecessary redundancy. Section 3.3 defines the used state variables for the quadrotor.

The mathematical formulation of the model can be expressed by the state equation and the output equation, both shown in system (2.23). These equations correspond to a continuous time system, without accounting for any disturbance in the loop. The equivalent discretized equations used are shown in system (2.24). Sections 2.3.3 and 2.3.5 detail standard control methods for MIMO systems.
Chapter 2. Background

\[
\begin{aligned}
\dot{x} &= Ax + Bu \\
y &= Cx
\end{aligned}
\]  \hspace{1cm} (2.23)

\[
\begin{aligned}
x_{k+1} &= \Phi x_k + \Gamma u_k \\
y_k &= Cx_k
\end{aligned}
\]  \hspace{1cm} (2.24)

2.3.3 LQR

A common control law in state space control is a state feedback law that assumes all states are available at each time step. It is normally written as (2.25). There are several approaches to obtain the control gains such that the closed-loop is stable. One such approach is to use optimal control.

To obtain the solution for the matrix \( K \) in optimal control a cost function is first defined. In linear quadratic regulator (LQR), this cost function is quadratic, and defined as shown in equation (2.26). The optimal control function can be adjusted by tuning matrices \( Q \) and \( R \). It follows:

\[
u_k = -Kx_k
\]  \hspace{1cm} (2.25)

\[
J = \int_0^T \left[ x(t)^T Q x(t) + u(t)^T R u(T) \right] dt
\]  \hspace{1cm} (2.26)

- A large valued \( R \) relative to \( Q \) will increase the penalty weight of the control input \( u \). In the limit, \( R \gg C^T Q C \) leads to an approximate cost function where the function is dominated by the control effort \( u \). Increasing \( R \) thus leads to a fast convergence but with large values. This is a high-gain control, also called an expensive control.
- A small value of \( R \), on the other hand, will increase the penalty weight on the state \( x \). In the limit, for \( C^T Q C \gg R \), the function is dominated by the output errors. Reducing \( R \) thus leads to a slow but steady convergence.
2.3.4 LQR-tuned PID

The PID controller is robust to steady-state error, versatile and easy to implement. One of its benefits is to use integral action, which is not incorporated in the the LQR controller per se. The LQR controller, based on optimal control, always assures closed-loop stability. It is possible to combine both and define a so-called PID equivalent of the optimal regulator for single-input single-output systems, as explained in [20] and [3], which is discussed next.

The method simply computes gains $K_{PLQR}^{PID}$ and $K_{dLQR}^{PID}$ of a controller targeting a set of outputs through LQR computation of the corresponding state space system and transforms them into the equivalent PID gains: $K_{PID}, K_{dPID}$ and $K_{iPID}$. Equations (2.27) to (2.31) show the necessary computations to obtain the PID equivalent gains from the LQR gains. To do this, one needs to define two auxiliar matrices, $\bar{K}_d, \bar{C}$ and $\bar{K}_i$, where the matrices $A, B$ and $C$ refer to the a system in continuous time.

\[
\begin{bmatrix}
K_{PID}^{PID} & K_{dPID}^{PID}
\end{bmatrix} = \bar{K}_P \bar{C}^{-1}
\] (2.27)

\[
\bar{C} = \begin{bmatrix}
C \\
C A - C B \bar{K}_P
\end{bmatrix}
\] (2.28)

\[
\bar{K}_P = (I_m + K_{dLQR}^{LQR} C B)^{-1}(K_{PLQR}^{LQR} C + K_{dLQR}^{LQR} C A)
\] (2.29)

\[
K_i = (I_m + K_{dLQR}^{LQR} C B)\bar{K}_i
\] (2.30)

\[
\bar{K}_i = (I_m + K_{dLQR}^{LQR} C B)^{-1}K_{iLQR}^{LQR}
\] (2.31)

Considering a state space system with $n$ spaces, $l$ outputs and $m$ inputs, the matrices sizes are shown in equations (2.32) to (2.39).

\[
K_{PID}^{PID}, K_{dPID}^{PID}, K_{iPID}^{PID} \in \mathbb{R}^{m \times l}
\] (2.32)

\[
K_{PLQR}^{LQR}, K_{dLQR}^{LQR}, K_{iLQR}^{LQR} \in \mathbb{R}^{m \times l}
\] (2.33)

\[
A \in \mathbb{R}^{n \times n}
\] (2.34)

\[
B \in \mathbb{R}^{n \times m}
\] (2.35)

\[
C \in \mathbb{R}^{l \times n}
\] (2.36)

\[
\bar{C} \in \mathbb{R}^{2l \times n}
\] (2.37)

\[
\bar{K}_P \in \mathbb{R}^{m \times 2l}
\] (2.38)

\[
\bar{K}_i \in \mathbb{R}^{m \times l}
\] (2.39)

This technique is used to implement the quadrotor controllers, with very simple calculations, in sections 3. However, while the quadrotor is a MIMO systems, the control stategy will consider several SISO separate and decoupled loops.
2.3.5 LQE

The standard state space model does not consider sources of noise, but both the model and the measurements insert noise in the system. Equation (2.40) extends the model, with $w$ as the process noise or uncertainty in the system model, like a wind disturbance for example, and $v$ as sensing noise or uncertainty in the measurements.

\[
\begin{aligned}
\dot{x} &= Ax + Bu + w \\
y &= Cx + v
\end{aligned}
\]  

(2.40)

Both $w$ and $w$ are assumed to be uncorrelated Gaussian white random noises, for simplification. Thus, the covariance of each, $R_{ww}$ and $R_{vv}$ respectively, are related to the corresponding uncertainty, and a larger covariance means a higher uncertainty.

Defining an estimate $\hat{x}$, the estimator equation can be written as shown in (2.41), where to the state equation a component $\tilde{y}$ is added accounting for the error between the actual output $y$ and the output computed from estimation $\hat{y} = C\hat{x}$.

\[
\begin{aligned}
\dot{\hat{x}} &= A\hat{x} + Bu + L\tilde{y} \\
y &= C\hat{x}
\end{aligned}
\]  

(2.41)

With an estimation error $\tilde{x} = x - \hat{x}$ then the dynamics can be written as shown in equation (2.42). The optimal estimator should minimize the estimation error over time in some sense. Thus, a cost function is defined as in LQR, which will be dependent on the covariances of both noises.

\[
\dot{\tilde{x}} = (A - LC)\tilde{x} + w - Lv
\]  

(2.42)

Similar to LQR the relative values of the two matrices is what matters. Usually $R_{ww}$ is also written as $W$ and $R_{vv}$ as $V$. An intuitive interpretation is the following:

- A big valued $V$ relative to $W$ will increase the penalty weight of the sensor measurements, which means that the model of the system is considered good with poor sensors.
- A small value of $V$, on the other hand, will increase the penalty weight on the state $x$, meaning that the dynamics are poorly modeled but are well-observed.
2.4 Camera model

A camera without a lens and with a single aperture is called a pinhole camera. For such cameras, the light rays project an inverted image on the opposite side of the camera. The pinhole camera model represents the transformation from 3D world points to the two-dimensional (2D) image points. In this model a point in space $\mathbf{X} = [X \ Y \ Z]^T$ in space is projected into a point in the image $\mathbf{x} = [x \ y \ z]^T$ by perspective projection by equations 2.43 and 2.44. Considering a perspective projection with a frontal plane the equations are similar, but without the negative signal.

$$x = -f \frac{X}{Z} \quad (2.43)$$

$$y = -f \frac{Y}{Z} \quad (2.44)$$

The resulting perspective projection 2D coordinates must still be transformed into the pixel sensor spacing and the relative position of the sensor plane must be transformed to the origin, to obtain the final image 2D points. This is obtained through the camera intrinsic parameters, which maps 3D space points to 2D image pixel coordinates. One way of representing this matrix is shown in equation 2.45, with no skew considered between axis.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2.45)$$

If the 3D points have known coordinates in a world reference frame, the pose transformation is accounted for with a rotation matrix $R$ and a translation vector $T$ combined in a matrix called extrinsic, $[R \ T]$, which transforms the 3D points to the camera reference frame. The camera matrix $P$ that brings the 3D points in the known reference frame to the image coordinates in shown in 2.46.

$$P = K[R \ T] \quad (2.46)$$

When considering the effect of the camera length additional parameters have to be known to compute the right projection, namely the distortion coefficients. The main distortion effect is radial distortion, which manifests as a visible curvature in the projection of straight lines. This distortion can be modeled as a simple low order polynomial model, shown in equations 2.47 and 2.48, that transforms distorted image coordinates into undistorted ones. These formulas are the ones for which the parameters calibrated in section 4.2 apply.

$$x = x(1 + k_1(x^2 + y^2) + k_2(x^2 + y^2)^3) \quad (2.47)$$

$$y = y(1 + k_1(x^2 + y^2) + k_2(x^2 + y^2)^3) \quad (2.48)$$

The other type of distortion is tangential distortion, which exists when the lens are not exactly parallel to the image plane. These are less significant and are considered null throughout the work. The remaining, $f_x, f_y, c_x, c_y, k_1, k_2$ have to be found to model the camera correctly and thus estimate a correct pose when applying 2.46. A possible way of obtaining these parameters is through geometric intrinsic calibration by using a planar pattern.
In this case, the pattern’s pose has to be recovered in conjunction with the intrinsic parameters. In this technique, each input image is used to compute a separate homography mapping the pattern points to image coordinates, as shown in equation 2.49, where \( r_0 \) and \( r_1 \) are the first two columns of extrinsic rotation matrix \( R \) and \( t \) is the extrinsic translation vector \( T \). From several different images, the equation allows the estimation of \( K \).

A planar pattern commonly used is a chess grid of \( n \times n \) equal squares. By modeling the radial distortion and obtaining the undistorted image, as shown in figure 9, it is possible to retrieve through geometric calibration the camera intrinsic parameters. This is the method used in section 4.2.
2.5 Visual marker identification

The focus of the image algorithms used in this thesis is to apply results of pose estimation to quadrotor localization for position control. Therefore, the focus is not on object recognition, thus motivating the use of visual markers to simulate objects in the world.

The chosen visual markers for this project are ArUco markers [14], which are created with the specific purpose of camera pose estimation. The ArUco markers are square shaped and can be divided in a grid where each inner cell represents a bit of information. A black cell is a bit of value zero and a white cell a bit of value one.

The outermost cells compose the margin, which has to have at least one bit width for marker visual recognition. The markers can also have different sizes, this is, different number of inner cells, as shown in figure 10, thus encoding more or less information. A dictionary is a set of markers with equal number of inner cells (or bits).

Each dictionary also has another parameter, the inter-marker distance. The inter-marker distance is the minimum distance among its markers, which is related to the error detection and correction capabilities of the dictionary. In general, lower dictionary sizes and higher marker sizes increase the inter-marker distance and vice-versa. It is thus recommended to use a dictionary that matches the application, to increase error robustness.

In openCV the detection algorithm is already implemented in a contribution module [1], with possibility of tuning the parameters. The algorithm follows the described steps:

- The image is first segmented by applying a local adaptive threshold method robust to different lighting conditions. The interval where the threshold window sizes are selected can be tuned in openCV. However, values which are too low can ‘break’ the marker border if the marker size is too large, and too high values can produce the same effect for markers which are too small. This process is shown in figure 11.
- Afterwards, a contour extraction is performed on the thresholded image and a polygonal approximation is performed. The ones that do not approximate 4-vertex polygons are discarded.
- Each candidate is analyzed by removing the perspective and using another threshold method. The binarized image is divided into a regular grid and each element is assigned the value of zero or one, depending on the majority of its pixels.
- The first rejection test consists in detecting the presence of the black border. If the border is not detected, then the candidate is refused.
The candidates then have its inner region analyzed for code extraction. A code is obtained for each possible rotation of the marker, thus leading to four codes per marker, and each code is searched in the dictionary, which leads to the final markers.

After this process, it is possible to extract the pose of each marker relative to the camera by matching the 2D and 3D points and applying (2.46). This requires the calibration parameters of the camera described in section 2.4, namely the camera matrix and distortion coefficients. Each marker has in the image four 2D points, and the marker length is known, thus the world points of a marker in its reference frame are also known.

Inevitably, this process is subject to the ambiguity problem, this is, two valid solutions can explain the observed projection of the marker. It generally happens when the marker is small in relation to the image, or in very inclined positions. To mitigate the problem, it is possible to give an estimate of $R$ and $T$ and look for the best pose near the last one. After computing the pose, it is possible to check with the openCV module functions if the pose is correct, by plotting a reference frame in the image, as shown in figure 12.
2.6 Vision-based position control

The authors in [8] present a general formulation of the visual servo control problem, defined as using computer vision data in the servo loop, with focus on the eye-in-hand paradigm, where robot motion implies camera motion. The goal in a visual servo control problem is to minimize an error defined as the difference between a vector $s(m(t), a)$ of $k$ visual features and $s^*$, the desired values for the same features, as shown in (2.50).

$$e(t) = s(m(t), a) - s^*$$  \hspace{1cm} (2.50)

$m(t)$ are the image features measurements and $a$ the parameters that represent potential additional knowledge. The case considered assumes a constant $s^*$ and changes in $s(m(t), a)$ as a direct result of camera motion. A basic approach is a velocity controller where $v_c$ is the input to the robot, related to $\dot{s}$ by (2.51), where $L_s \in \mathbb{R}^{k \times 6}$ is called the interaction matrix. The vector $v_c = (\upsilon_c, \omega_c)$ uses $\upsilon_c$ and $\omega_c$ as the linear and angular velocities.

$$\dot{s} = L_s v_c$$  \hspace{1cm} (2.51)

Taking the derivative of the error leads to (2.52), where $L_e = L_s$. If we want to ensure exponential decay the control law is defined as shown in (2.53), where $L_e^+$ is the Moore-Penrose pseudo-inverse of $L_e$.

$$\dot{e} = L_e v_c$$  \hspace{1cm} (2.52)

$$v_c = -\lambda L_e^+ e$$  \hspace{1cm} (2.53)

2.6.1 Image Based Visual Servo scheme

In Image Based Visual Servo (IBVS), $s$ is defined as a set of features that are immediately available in image data. $m$ are usually pixel coordinates and $a$ camera intrinsic parameters. Using the projection equations for a 3-D point $X = (X, Y, Z)$ in the camera frame (2.54), we can correspond $m = (u, v)$, where $u$ and $v$ are the $x$ and $y$ pixel coordinates in the image, $a = (c_u, c_v, f, \alpha)$ and $s = x = (x, y)$.

$$\begin{cases}
x = \frac{X}{Z} = \frac{u - c_u}{f} \\
y = \frac{Y}{Z} = \frac{v - c_v}{f}
\end{cases}$$  \hspace{1cm} (2.54)

Taking the time derivative of (2.54), and relating the velocity of the 3-D point to the camera frame velocity using (2.55), we eventually get (2.56) and (2.57), where $Z$ is the depth of the point relative to the camera frame.

$$\dot{X} = -\upsilon_c - \omega_c \times X$$  \hspace{1cm} (2.55)
Chapter 2. Background

\[
\dot{x} = L_x \, v \tag{2.56}
\]

\[
L_x = \begin{bmatrix}
-\frac{1}{Z} & 0 & \frac{x}{Z} & x & y & -(1 + x^2) & y \\
0 & -\frac{1}{Z} & \frac{y}{Z} & 1 + y^2 & -x & y & -x
\end{bmatrix} \tag{2.57}
\]

Because \(Z\) might not be known and the camera intrinsic parameters also, which are necessary to calculate \(x\) and \(y\), an estimate \(\hat{L}_x\) is required and relies on an estimate of \(Z\). We also know that to control six DOF we need at least three points, and the interaction matrices can be simply stacked, and it is recommendable to use more than this number. Three main strategies are presented to estimate \(L_x\):

- A strategy where we assume that \(Z\) of each point is available, although in practice it is estimated at each iteration;
- A strategy where we use the interaction matrix for the desired position with \(e = 0\), which makes \(\hat{L}_x\) constant and only the desired depth of each point is needed;
- A strategy that combined both of the previous (2.58).

\[
\hat{L}_x = \frac{1}{2} \left( L^t_x + L^*_x \right) \tag{2.58}
\]

Both the first two strategies show convergence but not desirable properties in the velocity and image behavior. The last strategy (2.58), however, provides good performance in practice: there are not large oscillations in the camera velocity components and both the image and the 3-D have smooth trajectories.

### 2.6.2 Position Based Visual Servo scheme

In Position Based Visual Servo (PBVS), a problem also called 3-D localization, \(s\) is a set of 3-D parameters that are related to the image measurements defined using the camera pose in respect to some reference frame. The intrinsic parameters and 3D models of the observer models are considered in \(a\), since they are \textit{a priori} knowledge.

Considering \(F_{c^*}\) the desired camera frame, \(F_c\) the camera frame and \(F_o\) an object frame, we can define \(s\) to be \((t, \theta_u)\), in which \(t\) is a translation vector and \(\theta_u\) the angle parametrization for the rotation. An example of such reference frames is shown in figure 13, where the object could be an ArUc, for example.

If \(t\) is defined relative to \(F_o\), then we have \(s = (^c t_o, \theta_u)\) and \(s^* = (^c^* t_o, 0)\), that correspond to the error in (2.59), where \(^c^* t_o\) and \(^c t_o\) are the coordinates of the object frame on the camera frame and on the desired camera frame, respectively.

\[
e = (^c t_o - ^c^* t_o, \theta_u) \tag{2.59}
\]
The interaction matrix is given by (2.60), with $x \sin(x) = \sin(x)$ and $\sin(0) = 1$. The result is a decoupled control law as shown in (2.61), since $L_{\theta u}^{-1} \theta u = \theta u$, resulting in a rotational motion that follows a geodesic with exponential decreasing speed with a pure straight line as image trajectory.

$$L_e = \begin{bmatrix} -I_3 & [^c t_o] \times \\ 0 & I_3 - \frac{\theta}{2} [u] \times + (1 - \frac{\sin(\theta)}{\sin(\theta/2)}) [u]^2 \end{bmatrix}$$

$$\begin{cases} v_c = -\lambda ([^c t_o] - ^c t_o) + [^c t_o] \times \theta u \\ \omega_c = -\lambda \theta u \end{cases}$$

A different approach is to consider $s$ the deviation of the 3D parameters to its goal pose, given by $s = (^c t_o, \theta u)$. This simplifies $s^*$, which is a zero-valued vector, and we get $e = s$. The interaction matrix is thus very simple, and given by (2.62). From this we get a simple decoupled control law, in (2.63), that results in a camera trajectory following a pure straight line.

$$L_e = \begin{bmatrix} R & 0 \\ 0 & L_{\theta u} \end{bmatrix}$$

$$\begin{cases} v_c = -\lambda R^T c^* t_o \\ \omega_c = -\lambda \theta u \end{cases}$$

The implementation of the image based control, shown in the next chapters, is similar to the PBVS method, although not completely equal. The basis is the same, with similar reference frames, and a similar error. However, it does not compute the interaction matrices nor does it use the same control law. Nonetheless, both the two PBVS approaches and the IBVS methods are state of the art and are a good comparison base for the applied method.
Chapter 2. Background

2.7 Localization and mapping

The general problem of representing a given environment globally through a set of partial views obtained while walking the space is commonly referred in literature as simultaneous localization and mapping (SLAM). In this setting the typical representation used for the map is a point cloud, a set of positions and corresponding RGB or intensity values stored for each known point of the map. Normally a voxelization is applied, this is, a cubic grid is considered where each cell or voxel only has a point associated. An example of a point cloud is shown in figure 14.

![Figure 14: Point cloud of a room viewed with PLC viewer](image)

2.7.1 Global object map

From this problem a derived approach where the world is represented through objects instead of a point cloud, which is known as SLAM++. This has obvious advantages regarding memory issues, since the information is compressed into a few objects and its relative or global poses.

In this thesis the objects are substituted by visual markers since the focus of the thesis is not object recognition and the algorithm used is a closed form way of estimating global positions of objects given pairwise estimations of relative poses, as presented in [27]. The algorithm extracts relative positions among objects from different views, and tries to estimate the global poses by minimizing a cost function as defined in (2.64) and (2.65).

\[
\{^1R_1, ^2R_1, \ldots, ^NR_1 \} = \arg\min_{\text{obs}} \left( \sum \|^jR_1 - ^jR_i \|^2 \right)
\]

\[
s.t. \begin{cases} 
\det(^1R_1) = 1 \\
^1R_1^T \cdot ^1R_1 = 1 
\end{cases}
\] (2.64)

\[
\{^1T_1, \ldots, ^NT_1 \} = \arg\min_{\text{obs}} \left( \sum \|^{j}T_1 - ^jR_i \cdot ^{j}T_1 - ^jT_i \|^2 \right)
\] (2.65)

The method used to solve the problem uses a sub-optimal algorithm with numeric global optimization in which the rotation matrices are transformed in the Rodrigues form, thus removing the restrictions in (2.64). This is the algorithm used in section 4.4.
2.8 Exploration

The map throughout this project is described by objects and its relative positions among themselves and to a global reference frame. By moving along the space with the quadrotor, the camera will be able to capture local views and localize itself in the map. This gives a basis to tackle the main challenge of this thesis and study the problem of exploration.

The present section presents an overview of exploration and a related problem, CPP, followed by an optimization overview in section 2.9, since the exploration problem encompasses not only the problem of exploring a given space, but also the problem of doing it in an optimal way.

2.8.1 Coverage path planning

The CPP problem has been studied by many in the research community and it is defined as the task of determining a path over all points of an area or volume of interest while avoiding objects. Based on a taxonomy proposed by Choset in [9], a survey on the CPP problem presented in [13] described several state of the art approaches. This section describes some of these approaches, with focus on 3D coverage;

Using the referred taxonomy, the approaches are classified on the decomposition, by whether or not they probably guarantee complete coverage of the free space, as heuristic or complete, and by whether or not the environment is known a priori, being off-line if it is, and on-line if they use sensor measurements in real time to sweep the target space. The main requirements defined for a coverage scenario are, according to [7]:

- The robot should pass through all points in the area;
- The robot must fill the region without overlapping paths;
- Continuous and sequential operation without any repetition of paths is required;
- Obstacle avoidance;
- Simple motion trajectories;
- An optimal path is desired.

The CPP problem is related to the covering salesman problem (CSP), a variant of the traveling salesman problem (TSP), both presented in section 2.9. As such, it can only be solved by approximation algorithms, since optimal algorithms do not guarantee timely solutions. In certain scenarios, a valid approach to solve the problem, although not optimal, is to randomize.

One of the main parts of solving a CPP problem is cell decomposition. Cells are non-overlapping regions of the map, and the union of all cells in a map fills the free space, this is, the space free from obstacles. Two cells are said to be adjacent if they share a common boundary. Typically, a planner based on cellular decomposition generates a coverage path in two steps:

1. It decomposes the path into cells and stores the decomposition graph;
2. It computes an exhaustive walk through the graph that determines the order in which each cell are visited to achieve complete coverage using some chosen algorithm.
2.8.1 Grid-based methods

One cell decomposition method is the grid-based approach, which uses a representation of the environment decomposed into a collection of uniform grid cells, each cell with an associated binary or probabilistic value stating whether an obstacle is present or not. These are approximate cellular decompositions, since they only approximate the shape of the target region and its obstacles. Usually, the cells are squares and robot-sized.

Most of these maps are "resolution-complete", this is, they are complete depending on the grid resolution. The maps also suffer from exponential growth of memory usage because the resolution remains constant regardless of the environment complexity. It also requires accurate localization to maintain map coherency. Because of this, these methods are suited for indoor mobile robot operations, as the size of the area is typically small and mobile robots are normally not capable of very fine adjustments, and there is no need for ultra-high resolution in coverage path planning.

2.8.2 Three-dimensional coverage

Most coverage path planning methods assume that the environment can be modeled as a simple planar surface. But there are 3D surfaces in nature that have to be covered by 3D coverage path planning, such as the case of autonomous underwater vehicle covering the seabed. In 3D coverage, covering 2D surfaces embedded in the space such as boundaries of automotive parts or buildings are normally the main focus, contrasting with the standard CPP problem which all free space must be covered.

2.8.2.1 Random sampling-based coverage of complex structures

In confined 3D areas where a robot cannot go through the spaces between component structures, global path planning strategies can be used with sampling-based techniques to find feasible, collision-free paths and obtain full coverage of a 2D target structure [12]. An adaptation of this algorithm is used to tackle the problem defined in 5.1.

The agent is considered to be fully actuated and have six DOF. First a graph of feasible paths is created by random sampling nodes until it allows full coverage. Then a minimum cost walk along the graph which fully covers the structure is searched. The first step is equivalent to solving the art gallery problem and the second a variant of the TSP. By favoring the random sampling it reduces computational burden.

Moreover, a method for smoothing and shortening initial paths is provided. Further advances presented an on-line version that incrementally explores the robot’s configuration space while constructing an inspection path until all points of the target are covered. This method is probabilistically optimal with respect to a given cost function.

2.9 Optimization and complexity

An optimization problem is a problem for which the best feasible solution is desirable, and the study of optimization algorithms is relevant for many practical applications. Some common groups of problems are task scheduling, routing and covering and packing problems. To be able to understand, evaluate and apply algorithms to any optimization problem, it is necessary to first define some concepts and notation.
Thus, in this section, some basic concepts are first presented, followed by a brief overview on some commonly known optimization problems of interest. Then some notes on dynamic programming (DP) are introduced, along with typical algorithms in the literature. The concepts and notation here presented are based on Ausiello’s, Bertsekas’ and LaValle’s literature [4] [5] [16]. The concepts and problems here presented are the basis for comparison and classification in section 5.1.

2.9.1 Notation and definitions

**Optimization problem** An optimization problem is a problem for which the goal is to find the "best" solution among all solutions. A problem \( P \) is characterized by the following:

- its set of instances \( I_P \);
- \( SOL_P \), a function that associates to any input instance \( x \in I_P \) the set of feasible solutions of \( x \);
- a measure function, \( m_P \), defined for pairs \( (x, y) \) such that \( x \in I_P \) and \( y \in SOL_P(x) \) that provides a positive integer which is the value of the feasible solution \( y \);
- \( goal_P \in \{MIN, MAX\} \) that specifies whether \( P \) is a maximization or a minimization problem.

**Decision problem** A computational problem \( P \) with an yes/no answer: formally, the set of all instances \( (I_P) \) can be partitioned into a set of positive \( (Y_P) \) and a set of negative \( (N_P) \) instances and the problem asks, for any instance, to verify whether it belongs to the positive set. The whole setting of complexity theory is built up in terms of decision problems.

**Deterministic algorithm** An algorithm where for the same input and setting, the result is the same (also known as conventional algorithm).

**Non-deterministic algorithm** An algorithm where, at some point, a guess is performed, this is, a choice is done randomly.

**Reducibility** A problem \( P_1 \) is reducible to another \( P_2 \) if there is a method for solving \( P_1 \) using an algorithm for \( P_2 \). This means that \( P_2 \) is at least as difficult as \( P_1 \), if the reduction only involves simple calculations.

2.9.2 Complexity analysis of optimization problems

The setting of complexity theory for optimization is built up in terms of decision problems, as stated in section 2.9.1. These can be classified in the following categories, depending on the time/space complexity of solving the problem.

**Class P** Class of problems solvable by deterministic algorithms in time proportional to a polynomial of the input size

**Class PSPACE** Class of problems solvable by deterministic algorithms in space proportional to a polynomial of the input size

**Class EXPTIME** class of problems solvable in time proportional to an exponential of the input size

**Class NP** Class of problems that a non-deterministic algorithm can solve in time proportional to a polynomial of the input size
Chapter 2. Background

Since no algorithm can access more memory locations than the number of computing steps performed, \( P \subseteq PSPACE \). Furthermore, \( PSPACE \subseteq EXPTIME \) under suitable assumptions on the computation model. It is also known that \( P \subseteq EXPTIME \), and more specifically, \( P \subset EXPTIME \). Regarding \( NP \), it is clear that \( P \subseteq NP \), since conventional algorithms can be considered a subset of non-deterministic algorithms, in which no “guessing” is performed. Other useful categorization, that gives an insight on the difficulty of the problems and on the complexity of the algorithms that solve them, is now stated.

**NP-hard** A problem is NP-hard if it is at least as difficult to solve (in terms of time complexity and apart from a polynomial-time reduction) as any problem in NP.

**NP-complete** A problem is NP-complete if it belongs to NP class and every NP class problem is reducible to the NP-complete problem in polynomial time.

**PSPACE-hard** A problem is PSPACE-hard if it is at least as difficult to solve (in terms of time complexity and apart from a polynomial-time reduction) as any problem in PSPACE; PSPACE-hard implies NP-hard.

**PSPACE-complete** A problem is PSPACE-complete if it belongs to PSPACE class and every PSPACE class problem is reducible to the PSPACE-complete problem in polynomial time.

A great number of relevant optimization problems is computationally intractable, in the sense that the only way to solve them is by using algorithms that run in exponential time. In formal terms, this fits in the NP-hard category and trying to compute the exact solution might require months or years of machine time. It is important to note that, although until now there is no solid proof that NP-hard problems cannot be solved using algorithms that run in polynomial time, there is strong evidence that suggests such algorithms may not exist. Hence, most algorithms used are approximation algorithms.

In the following section we present some paradigmatic problems that have been and continue to be studied thoroughly in the literature. Many times it is only necessary to prove that a given problem is as difficult as one of these emblematic settings to categorize it, and understanding its complexity. They are also extremely important because they are representative of many others and finding a way to solve them leads to the understanding of how to solve hundreds of different but related others.

### 2.9.2.1 Traveling salesman problem

One of the most emblematic problems, studied for over 60 years, is the TSP \([2]\). It was first referred as such in literature in \([26]\) and it can be stated as follows:

"Given a set of cities along with the cost of travel between each pair of them, the traveling salesman problem, or TSP for short, is to find the cheapest way of visiting all the cities and returning to the starting point."

In the standard version, the travel costs are symmetric, this is, traveling from city \(X\) to city \(Y\) costs the same as traveling from \(Y\) to \(X\). This version of TSP is called symmetric travelling salesman problem (STSP), otherwise the problem is called asymmetric travelling salesman problem (ATSP).

A simple variant that doesn’t require a return to the starting point is simply the problem of finding a Hamiltonian path, i.e. a path in a graph (undirected or directed) that visits each vertex exactly once, which is already by itself a hard problem.
2.9.2.2 Covering salesman problem

The CSP presented in [11] is a variant of the TSP where rather than visiting every city directly, it is sufficient to stop at some other, within a specified maximum distance, in its neighborhood. A generalized approach was presented in [15], referred to as generalized covering salesman problem (GCSP), where each node needs to be covered at least \( k \) times, and a cost can be associated with visiting a given node. The problem is still that of finding the minimum tour that "covers" all nodes, such that each node is covered the necessary times. It can be divided in three cases:

**Binary Generalized Covering Salesman Problem** A version where the tour can only visit each node once, and after visiting a node and must satisfy the coverage need of that node by visiting others.

**Integer Generalized Covering Salesman Problem without Overnights** A node can be visited more than once, but an overnight stay is not allowed, this is, it can’t visit the same node without visiting at least some other node.

**Integer Generalized Covering Salesman Problem with Overnights** A similar version to the previous one, but where overnights are allowed.

2.9.2.3 Art gallery problem

The art gallery problem is the problem of finding the minimum number of guards required to supervise any art gallery with \( n \) walls. It was introduced by Victor Klee in 1973 and Chvátal presented a theorem [10], that proved the problem is bounded: "Every \( n \)-triangulation can be partitioned into \( m \) fans where \( m \leq n/3 \)."

However, the goal of the art gallery problem is to find the minimum number of guards, and not only a bound, and this problem was shown to be NP-hard in [17]. More specifically, both the minimum vertex guard problem, which is the art gallery problem where the guards are stationed at the vertices, and the minimum edge guard problem, were shown to be NP-hard.
2.9.3 Graph search algorithms

The section provides an overview on the basic search algorithms, depth first search (DFS) and breadth first search (BFS), and it then presents optimal algorithms that take into account the edge costs. The optimal algorithms presented work as a starting point for solving the coverage problem presented in chapter 5.

The DFS is one of the most simple algorithms in the literature. It explores in depth, this is, when it chooses a given neighbor, it tries to go the furthest possible before exploring any other neighbor. In this sense, while exploring nodes, it behaves like a last-in-first-out (LIFO) or a stack. As a result, it is not an optimal algorithm in any way, since it will find any path, and not the best path. Figure 15 represents a simple example of the DFS for path search in a grid. The shown example considers 4 possible movements, left, right, up and down in a grid with "obstacles". Moreover, the resulting path depends on the order the neighbors are pushed into the stack.

Unlike its counterpart, BFS finds the shortest path, this is, the path that yields a lesser number of nodes. It explores in breadth, going through every neighbor before exploring further possible before exploring any other neighbor. However, although the resulting path is the shortest, in many examples where the edges have costs, this path is not necessarily the optimal path. Its behavior while exploring nodes is that of a first-in-first-out (FIFO) or a queue. Figure 16 represents an example for the BFS algorithm.

To achieve optimality more complex algorithms have to take into account the cost of the edges, which can take different
values. Another concept introduced in these algorithms is the **heuristic**. An heuristic is an estimate, and particularly, an underestimate of the necessary cost to achieve the goal.

The Dijkstra’s algorithm takes into account the total cost of the path, this is, it always explore the node corresponding to the shortest path. It works as an ordered queue and it guarantees to find an optimal path. Figure 17 shows an example where Dijkstra’s and BFS would produce different outputs, showing Dijkstra’s optimality. However, when all edges have equal cost it degenerates to a BFS.

![Figure 17: Dijkstra’s algorithm in a graph where the optimal result is not the BFS path](image)

The Best first search algorithm, described in algorithm 2.1, is an algorithm that takes into account only the heuristic, expanding always the most promising node.

```
Algorithm 2.1: Best-first search

begin

OPEN = ∅
CLOSE = ∅
Add(S, OPEN) /* S starting node */

while OPEN ≠ ∅ do
    V ← node with minimum heuristic
    Remove(V, OPEN)
    Add(V, CLOSE)
    if IsGoal(V) then
        return reconstructPath(V)
    end
    for each neighbor Q of V do
        if Q ∈ CLOSE then
            continue
        end
        if Q ∉ OPEN then
            Add(Q, OPEN)
        end
        parent(Q) = V
    end
end

```

The Best first search algorithm, described in algorithm 2.1, is an algorithm that takes into account only the heuristic, expanding always the most promising node.

![Algorithm 2.1: Best-first search](image)

The procedure also works as an ordered queue, although ordered with a different criterion, and it does not guarantee an optimal path. Figure 18 exemplifies this behavior.
Chapter 2. Background

Figure 18: Best-first search result in for a graph where the heuristic alone produces a non-optimal path

The A* algorithm takes into account both the cost and the heuristic. The closer the heuristic is to the real cost until the goal, the faster the algorithm progresses. This means that if the heuristic is exactly equal to the cost until the goal, then A* only expands the shortest path: with perfect information the algorithm behaves optimally.

Algorithm 2.2: A* algorithm

1 begin
2 OPEN = ∅
3 CLOSE = ∅
4 Add(S, OPEN) /* S starting node */
5 while OPEN ≠ ∅ do
6 \( V \leftarrow \text{node with minimum } f_{\text{score}} \)
7 Remove(V, OPEN)
8 Add(V, CLOSE)
9 if IsGoal(V) then
10 \( \text{return reconstructPath(V)} \)
11 end
12 for each neighbor Q of V do
13 if Q ∈ CLOSE then
14 continue
15 end
16 tentative_gscore = gscore(V) + cost(V,Q)
17 if Q ∈ OPEN then
18 \( \text{Add}(Q, OPEN) \)
19 end
20 if tentative_gscore ≥ gscore(Q) then
21 continue
22 end
23 gscore(Q) = tentative_gscore
24 fscore(Q) = gscore(Q) + heuristic(Q)
25 parent(Q) = V
26 end
27 end
28 end

Algorithm 2.2 describes the procedure, and its clear that if the heuristic is null (\( h(n) = 0 \)) then the algorithm degenerates to Dijkstra’s algorithm. On the other hand, if the cost is equal for all edges or ignored (\( g(n) = 0 \)), then it degenerates to best-first search.

Figure 19 presents the results of the optimal A* method for the grid example, since the heuristic is easily computed to represent the best underestimate. This is not the case in many examples, and the A* can be a slow algorithm for large
graphs where the heuristic is not close to the real cost to the goal.

Figure 19: A* search result for a grid, using the Manhattan distance \((x + y)\) as heuristic; in each step there is the tie with the node exactly above the starting node, and it is assumed that the tie was solved using some criteria.
Chapter 2. Background
3

Quadrotor control

The first goal of the thesis is to design control and computer vision algorithms for indoor localization and navigation for a low cost UAV. To achieve this, one must first address the quadcopter stabilization problem, i.e., bring the quadcopter to the equilibrium state (at hover) from any initial state. This is also referred to as the inner-loop control.

For the control code deployment the chosen platform is Simulink® library ARDrone Target by Daren Lee [18]. This application program interface (API) provides means of extracting sensor information and operate the quadrotor actuators.

3.1 Sensor calibration

3.1.1 Accelerometer

The Simulink® AR Drone API retrieves measurements from the MEMS accelerometer and makes this data available at the output of the Navigation Subsystem. In spite of the pre-processing that happens in this block, this is still raw data from the sensor, which needs to be processed for angle estimation.

As referred in section 2.1 there is a persistent deviation of the sensor measurements from the true values even when the device is still, that is, a bias in each axis. Quantization is also an issue since the device has a digital output. Therefore, the values have to be scaled to match the real acceleration. Both operations are computed in the AccelerometerCalibration block, a subsystem presented in figure 20.

![Figure 20: Accelerometer calibration subsystem scheme](image)

For each of these the raw values obtained, an average is computed over four seconds and stored in the matrix defined in (3.1a) with the maximum and minimum values, where each row corresponds to an axis. The bias are the averaged values when the axis is perpendicular to the gravity vector, thus calculated by equation (3.1b).
Chapter 3. Quadrotor control

Acc_cali = \begin{bmatrix}
  acc_{x}^{\text{max}} & acc_{x}^{\text{bias}} & acc_{x}^{\text{min}} \\
  acc_{y}^{\text{max}} & acc_{y}^{\text{bias}} & acc_{y}^{\text{min}} \\
  acc_{z}^{\text{max}} & acc_{z}^{\text{bias}} & acc_{z}^{\text{min}} 
\end{bmatrix}

(3.1a)

acc^{\text{bias}} = Acc_{\text{cali}} \binom{0}{1}{0}^T

(3.1b)

The scaling factors are calculated with the average of the absolute values of the maximum and minimum measurements, which will represent the scale limit, and are then scaled to the real acceleration scale, as shown in (3.2).

acc^{\text{gain}} = \begin{bmatrix}
  9.81 & 0 & 0 \\
  -9.81 & 0 & 0 \\
  0 & -1 & 0
\end{bmatrix} \times \begin{bmatrix}
  acc_{x}^{\text{scale}} \\
  acc_{y}^{\text{scale}} \\
  acc_{z}^{\text{scale}}
\end{bmatrix}

with \begin{bmatrix}
  acc_{x}^{\text{scale}} \\
  acc_{y}^{\text{scale}} \\
  acc_{z}^{\text{scale}}
\end{bmatrix} = \frac{1}{2} * Acc_{\text{cali}} \begin{bmatrix}
  1 \\
  0 \\
  -1
\end{bmatrix}

(3.2)

All values are obtained through calibration, before each set of flights, where each axis is calibrated first with the axis orthogonal to the gravity acceleration and then parallel, once pointing in the same direction and other reversed. To simplify the measurements, the positions can be colored to the six presented in figures 21a to 21f.

![Images of calibration positions](image)

Figure 21: Calibration minimum set of positions

3.1.2 Gyroscope

The gyroscope has a similar calibration process, with the Simulink AR Drone API returning the raw data from the MEMS gyroscope at the output of Navigation Subsystem. The raw data only needs to account for the bias (section 2.1), and is only scaled to switch between degrees per second and radians per second. The bias is also different for each axis.

Each axis sensor calibration can be made from any position, since it only depends on the stillness of the quadrotor. However, the sensor will be better calibrated with the quadrotor positioned as shown in figure 21f, since it has most stability, therefore ensuring the sensor is still. This process is done inside the calibration state, returning a bias vector as (3.3).

\[
\begin{bmatrix}
  \text{gyro}_{x}^{\text{bias}} \\
  \text{gyro}_{y}^{\text{bias}} \\
  \text{gyro}_{z}^{\text{bias}}
\end{bmatrix}
\]

(3.3)
3.2 PWM calibration

The quadrotor’s model relates rotor speed with forces, as seen in section 2.2. However, the quadrotor motors receive as input a PWM value, explained in 2.1. The relation between the desired forces and corresponding PWM, in equation (2.1), must then be determined.

To obtain these constants, each rotor is tested separately and for each PWM value, the corresponding generated force must be registered. Since the forces of each rotor are approximately parallel to the gravity acceleration, with an opposite direction, this can be done by a simple process:

- The quadrotor is placed on a scale, with its center of mass aligned to the center of the scale
- Each PWM value is given at a time and the weight is registered

By taking the weight correspondent to a PWM value, and taking that the applied downwards force is the sum of both the gravity acceleration and the generated force, depicted in (3.4) and (3.5), it is possible to extract the force generated by PWM.

\[
F_{\text{total}} = F_g + F_{PWM} \tag{3.4}
\]

\[
F_{\text{total}} = m g - F_{PWM} \tag{3.5}
\]

The gravitational mass of the object is the one shown in the scale when no other force is applied on it (3.6). However, if some force is applied, the value shown by the scale will differ, and it will account for the total force applied, as shown in (3.7).

\[
F_g = m g \tag{3.6}
\]

\[
m_{sc} = \frac{F_{\text{total}}}{g} \tag{3.7}
\]

\[
F_{PWM} = g (m_g - m_{sc}) \tag{3.8}
\]

Using the previous equations, and considering the distance from the center to the rotors to be small, therefore considering it as applied vertically on the center of mass, the values are obtained from equation (3.8). These values are extracted for all possible range of PWM and linear interpolated to get a polynomial curve.

The results obtained are presented in section 6.2, and cover mainly two cases: the quadrotor using the indoor hull; and the quadrotor using the indoor hull while carrying a raspberryPi3, which should shift the weights, affecting the curves relative to each other. The obtained constants are shown in Table 1, with approximate values.

| | case 1 | | case 2 | | | |
|---|---|---|---|---|---|
| | a | b | c | a | b | c |
| Rotor 1 | 0.12383 | 0.0062604 | 0.00015380 | 0.11683 | 0.0073997 | 0.00013051 |
| Rotor 2 | 0.12538 | 0.0068520 | 0.00014427 | 0.15964 | 0.0033750 | 0.00016811 |
| Rotor 3 | 0.11745 | 0.0061852 | 0.00011921 | 0.12629 | 0.0049323 | 0.00017001 |
| Rotor 4 | 0.11576 | 0.0059658 | 0.00011246 | 0.15399 | 0.0025506 | 0.00015782 |
3.3 State definition

Section 2.3.2 presented the state as a minimum set of variables that describes the system behavior and can represent any change throughout time. For the quadrotor, the state can be represented by (3.9), gathering all variables that involve position and orientation of the quadrotor. From (2.18) it is possible to directly take the state equations for \((x, y, z)\), since both triples are represented in the inertial frame \(\{A\}\).

\[
x = \begin{bmatrix} \phi & \theta & \psi & p & q & r & x & y & z & u & v & w \end{bmatrix}
\]

(3.9)

From (2.19) it is possible to write equations for \((u, v, w)\), shown in (3.11), considering the vector \(F\) accounts for the thrust applied in the \(z\) component, as shown in (3.10), represented in the body frame \(\{B\}\), which multiplied by the rotation matrix puts it in the inertial frame.

\[
F = \begin{bmatrix} 0 & 0 & T \end{bmatrix}^T
\]

(3.10)

\[
\begin{cases}
\dot{u} = T \frac{c_\phi s_\phi - c_\theta s_\phi s_\psi}{m} \\
\dot{v} = T \frac{s_\phi c_\phi - c_\psi s_\phi}{m} \\
\dot{w} = g - T \frac{s_\phi}{m}
\end{cases}
\]

(3.11)

For the orientation, since vector \((\phi, \theta, \psi)\) is expressed in \(\{A\}\) and \(\Omega\) in \(\{B\}\) it is necessary to transform \(\Omega\) to \(\{A\}\). The idea with angular rates is to consider small changes on each angle, and compute its effects on the rotation vector. The first Euler angle goes through two rotations, the second through one and the final angle does not need any additional rotations. Following the same convention as (2.15), the order \(Z - X - Y\) is used, and equation (3.12) is obtained.

\[
\Omega = R(\theta) R(\phi) \begin{bmatrix} 0 \\ 1 \\ \frac{d\theta}{dt} \end{bmatrix} + R(\theta) \begin{bmatrix} \frac{d\phi}{dt} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
\]

(3.12)

The equation gives a relation between inertial angular rates and body frame angular rates. By inverting the resulting matrix, the opposite relation is computed, presented in (3.13). It is thus possible to write the state equations of the angular rates, as shown in (3.14).

\[
\begin{bmatrix} \frac{d\phi}{dt} \\ \frac{d\theta}{dt} \\ \frac{d\psi}{dt} \end{bmatrix} = \begin{bmatrix} c_\theta & 0 & s_\theta \\ s_\theta t_\phi & 1 & -c_\theta t_\phi \\ -s_\theta & 0 & c_\theta \end{bmatrix} \Omega
\]

(3.13)

\[
\begin{cases}
\dot{\phi} = p + q s_\phi t_\theta + r c_\phi s_\theta \\
\dot{\theta} = q c_\phi - r s_\phi \\
\dot{\psi} = q s_\phi + r \frac{c_\psi}{c_\theta}
\end{cases}
\]

(3.14)
Lastly, from (2.21) the equations for \( \dot{\Omega} \) are written, with \( I \) a diagonal matrix of the inertial moments and \( \tau \) the vector with the three torque components \((\tau_\phi, \tau_\theta, \tau_\psi)\). The resulting state equations for \((p, q, r)\) are shown in (3.15).

\[
\begin{align*}
\dot{p} &= -q \, r \left( \frac{I_{yy}}{I_{xx}} - \frac{I_{yx}}{I_{xx}} \right) + \frac{\tau_\theta}{I_{xx}} \\
\dot{q} &= -p \, r \left( \frac{I_{yx}}{I_{xx}} - \frac{I_{xx}}{I_{xx}} \right) + \frac{\tau_\phi}{I_{xx}} \\
\dot{r} &= -p \, q \left( \frac{I_{yy}}{I_{xx}} - \frac{I_{yx}}{I_{xx}} \right) + \frac{\tau_\psi}{I_{xx}} \\
\end{align*}
\]

All the state equations are now written, but need to be linearized to be used with the control methods presented. The first possible simplification is the small angles approximations for \( \phi \) and \( \theta \), presented in (3.16), since for the quadrotor to be stable, both these angles will always be small.

\[
\begin{align*}
\sin(\theta) &\approx \theta \\
\cos(\theta) &\approx 1 - \frac{\theta^2}{2} \approx 1 \\
\end{align*}
\]

Another approximation is considering that small angles multiplied by each other will be approximately zero, that is, \( \delta \phi \, \delta \theta = 0 \), \( \delta \phi \, \delta \psi = 0 \), \( \delta \psi \, \delta \theta = 0 \). Finally, a last approximation is to consider that the angular rate about the \( z \) axis, the yaw rate \( r \), is small enough to be neglected, and that the inertial moments in \( x \) and \( y \) are almost equal, as shown in (3.17).

\[
\begin{align*}
\{ r << 1 \\
I_{xx} \approx I_{yy} \\
\end{align*}
\]

Putting together all the equations and enforcing the described approximations the set of state equations is transformed as shown in (3.18). From these, it is easy to see that many of these are decoupled from each other, this is, the state variables are independent, and the system can be partitioned in smaller subsystems.

\[
\begin{align*}
\dot{x} &= u \\
\dot{y} &= v \\
\dot{z} &= w \\
\dot{u} &= T \frac{c_\phi^2 \theta c_\theta}{m} \\
\dot{v} &= T \frac{\theta c_\phi \phi c_\theta}{m} \\
\dot{w} &= -T \frac{\theta c_\phi s_\theta}{m} \\
\dot{\phi} &= p + q \, s_\phi \theta + r \, c_\theta \theta \\
\dot{\theta} &= q \, c_\phi - r \, s_\phi \\
\dot{\psi} &= q \, s_\phi - r \, c_\phi \\
\end{align*}
\]

\[
\begin{align*}
\dot{x} &= u \\
\dot{y} &= v \\
\dot{z} &= w \\
\dot{\phi} &= p \\
\dot{\theta} &= q \\
\dot{\psi} &= r + q \phi \\
\end{align*}
\]

For quadrotor stabilization the focus is attitude and altitude control. Taking \( z \) and \( w \) equations, the obtained subsystem is defined by (3.20a), (3.20b) and (3.21), where the state and input vectors are given in (3.19a) and (3.19b). The output matrix is such that only \( z \) is a measurable component of the state of this subsystem, because the sensors do not allow to obtain the velocity along the \( z \) axis. This altitude will be measured with an observer.
Chapter 3. Quadrotor control

\[
x_z = \begin{bmatrix} z \\ w \end{bmatrix} \quad \text{(3.19a)} \quad \quad u_z = [T] \quad \text{(3.19b)}
\]

\[
A_z = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad \text{(3.20a)} \quad \quad B_z = \begin{bmatrix} 0 & \frac{1}{m} \end{bmatrix} \quad \text{(3.20b)}
\]

\[
C_z = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \text{(3.21)}
\]

The attitude control is decoupled in controlling the subsystems corresponding to \((\phi, \dot{\phi})\) and \((\theta, \dot{\theta})\), with state and output equations presented from (3.22a) to (3.27). The inertial moments used are \(I_\phi = 0.0022375681\) and \(I_\theta = 0.002985236\). Note that the output equations are such that the full state of each system is measurable, since it is available via the sensors.

\[
x_\phi = \begin{bmatrix} \phi \\ \phi_p \end{bmatrix} \quad \text{(3.22a)} \quad \quad u_\phi = [\tau_\phi] \quad \text{(3.22b)}
\]

\[
A_\phi = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad \text{(3.23a)} \quad \quad B_\phi = \begin{bmatrix} 0 & \frac{1}{I_\phi} \end{bmatrix} \quad \text{(3.23b)}
\]

\[
C_\phi = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \text{(3.24)}
\]

\[
x_\theta = \begin{bmatrix} \theta \\ \theta_q \end{bmatrix} \quad \text{(3.25a)} \quad \quad u_\theta = [\tau_\theta] \quad \text{(3.25b)}
\]

\[
A_\theta = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad \text{(3.26a)} \quad \quad B_\theta = \begin{bmatrix} 0 & \frac{1}{I_\theta} \end{bmatrix} \quad \text{(3.26b)}
\]

\[
C_\theta = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \text{(3.27)}
\]

The \((\psi, \dot{\psi})\) equations show a relation with \(q\) and \(\phi\), which don’t allow decoupling. However, \(\psi\) is available through \(r\) integration, as section 3.5 shows, then \(\psi \in \{B\}\) and the equations are presented in (3.28a) to (3.30), with \(I_\psi = 0.00480374\).

\[
x_\psi = \begin{bmatrix} \psi \\ \psi_r \end{bmatrix} \quad \text{(3.28a)} \quad \quad u_\psi = [\tau_\psi] \quad \text{(3.28b)}
\]

\[
A_\psi = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad \text{(3.29a)} \quad \quad B_\psi = \begin{bmatrix} 0 & \frac{1}{I_\psi} \end{bmatrix} \quad \text{(3.29b)}
\]

\[
C_\psi = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \text{(3.30)}
\]
3.4 State estimation

This section describes how to estimate the state values, using Kalman filtering. However, first it is necessary to calibrate the sensors (accelerometer and gyroscope), by removing the bias, and extract the angles from the sensor measurements.

3.4.1 Angles from accelerometer

As referred in section 2.1, the accelerometer gives values which can be transformed into $\phi$ and $\theta$. This is achieved by using the values after calibration in equations (2.4) and (2.5). These can be simplified, however, by using the same approximation presented in 3.3 for the first angle, $\phi$, approximating $\sqrt{x^2 + z^2}$ to $z$, since $x \ll z$ when close to hovering.

\[
\phi_{\text{acc}} = \arctan \left( \frac{y}{z} \right) \quad (3.31)
\]

\[
\theta_{\text{acc}} = \arctan \left( \frac{x}{\sqrt{y^2 + z^2}} \right) \quad (3.32)
\]

Equations (3.31) and (3.32) are obtained and rearranged to (3.33), which after applying trigonometric relations, results in the equivalent for $\theta$ presented in (3.34). Since the accelerometer frame does not match the body frame, an adjustment is made and $\theta \approx -\theta_{\text{acc}}$. The equations are implemented with simple MATLAB blocks, as shown in figure 22.

\[
\theta_{\text{acc}} = \arctan \left( \frac{x}{\sqrt{1 + \tan^2(\phi)}} \right) \quad (3.33)
\]

\[
\theta_{\text{acc}} = \arctan \left( \frac{x \cos(\phi)}{z} \right) \quad (3.34)
\]

![Figure 22: Roll and pitch computation from accelerometer in Simulink®](image)

3.4.1.1 Attitude estimation

The processes described in section 2.3.5 are implemented, with a Kalman filter for both $\phi$ and $\theta$. A slight change in the systems is that matrix $B$ is different from the one used in control. Instead, it is set to zero, canceling the inputs. This is an approximation that works well. The matrices defining the system for $\phi$ are shown in equations (3.35a) to (3.35c).

\[
A_\phi = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad (3.35a)
\]

\[
B_\phi = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (3.35b)
\]

\[
C_\phi = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (3.35c)
\]
For $\theta$, the matrices are presented in equations (3.35a) to (3.35c). The resulting Kalman systems are shown in table 2.

\[
A_\theta = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad B_\theta = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad C_\theta = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\] (3.36a) (3.36b) (3.36c)

Table 2: Kalman filter values for attitude control

<table>
<thead>
<tr>
<th>LQE tuning</th>
<th>Resulting system</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_{\phi}$</td>
<td>$R_{\phi}$ 0.01</td>
</tr>
<tr>
<td>$Q_{\theta}$</td>
<td>$R_{\theta}$ 0.01</td>
</tr>
</tbody>
</table>

The value of angular velocity $r$ is also represented in this process, passing only through calibration, as explained further on. An important factor to take into account, though, is the units in which the measurements are at each point, to guarantee coherence in the model and avoid unnecessary error.

### 3.4.1.2 Altitude estimation

For the altitude estimation, the system is presented in equations (3.37a) to (3.37c) and the resulting matrices are shown in table 3.

\[
A_z = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad B_z = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \quad C_z = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\] (3.37a) (3.37b) (3.37c)

Table 3: Kalman filter values for altitude control

<table>
<thead>
<tr>
<th>LQE tuning</th>
<th>Resulting system</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q$ $\begin{bmatrix} 10 &amp; 0 \ 0 &amp; 10 \end{bmatrix}$</td>
<td>$R$ 0.01 $A_{\theta}$ $\begin{bmatrix} -0.0015 &amp; 0.0025 \ -0.9978 &amp; 1.0000 \end{bmatrix}$</td>
</tr>
</tbody>
</table>
3.5 Control scheme

The implementation of a stabilizing controller for the quadcopter requires an inner loop controller, which is preceded by the state estimator and followed by the conversion to PWM values. The following sections present these systems and the gains used for quadrotor stabilization.

3.5.1 Attitude control

The `lqr` Matlab command is then applied to the systems in (3.23a) and (3.23b) as shown in (3.38), resulting in two LQR gains, each corresponding to one state, respectively, $\phi$ and $\dot{\phi}$. The obtained gain values from the LQR computations are presented in table 4.

$$K_{lqr} = lqr(A, B, Q, R); \quad (3.38)$$

These gains are not applied in a typical control law, $u = -Kx$, but are rather used to tune a $\phi$ PD controller, as explained in 2.3.4. Hence, applying (3.23a), (3.23b) and (3.24) in equation (2.27), (3.39) is obtained. The equations simplifies to (3.40) and the results return the gains in (3.42).

$$\bar{K}_{p} = (1 + K_{dLQR} \begin{bmatrix} 1 & 0 \end{bmatrix})^{-1}(K_{pLQR} \begin{bmatrix} 1 & 0 \end{bmatrix} + K_{dLQR} \begin{bmatrix} 0 & 1 \end{bmatrix}) \quad (3.39)$$

$$\bar{K}_{p} = \begin{bmatrix} K_{pLQR} & K_{dLQR} \end{bmatrix} \quad (3.40)$$

$$\bar{C} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ \frac{1}{\tau} & 0 \end{bmatrix} \begin{bmatrix} K_{pLQR} & K_{dLQR} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (3.41)$$

$$\begin{bmatrix} K_{pPID} & K_{dPID} \end{bmatrix} = \begin{bmatrix} K_{pLQR} & K_{dLQR} \end{bmatrix} \quad (3.42)$$

The control block receives both the variables to control and its references, respectively, $\phi$, $\phi_{ref}$, $p$ and $p_{ref}$, as shown in figure 23, as well as a torque reference for $\tau_\psi$. This reference is used to force a small tilt to the quadrotor, if necessary, and it is computed in the outer loop controller, explained in section 3.6. Gains 6 and 7 in the figure’s control scheme correspond to the LQR gains, $K_d$ and $K_p$, respectively. The additional gains shown, gain 18 and 23, are adjustment gains, to do small and fast adjustments after obtaining good LQR values. These adjustments are also shown in table 4.

The control scheme for $\theta$ is very similar to the one used for $\phi$, with the same LQR-tuned PD strategy. Since the matrices are similar, once again the gains have a direct correspondence among themselves, this is, $K_{pPID} = K_{pLQR}$ and $K_{dPID} = K_{dLQR}$. The control scheme is analogous to the one in figure 23. Once again, there is a fifth input related to the torque reference for $\theta$, discussed further on, and adjustment gains for each component, gain 16 and 17. The LQR and adjustment gains are shown in table 4.
Chapter 3. Quadrotor control

Figure 23: Phi LQR-tuned PD controller

The control scheme for $\psi$ adopts the same strategy, but $\psi$ is obtained through integration of $R$. Thus, the controller is actually a PI over this angle’s velocity. In the control loop, a switch turns the integration of $\psi$ on or off, giving the possibility to use a simple feedback law of the velocity. Through testing the PI shows its usefulness, although the adjustments are bigger than for the previous controllers. Table 4 puts together all $V$ and $W$ matrices used for LQR gain computation, as well as the LQR gains and the adjustment gains used.

Table 4: Control parameters for attitude control

<table>
<thead>
<tr>
<th>Parameter</th>
<th>$Q$</th>
<th>$R$</th>
<th>LQR</th>
<th>Adjustment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi$</td>
<td>$\begin{bmatrix} 2 &amp; 0 \ 0 &amp; 0.002 \end{bmatrix}$</td>
<td>10</td>
<td>$K_p$ 0.4472 $K_p$ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$K_d$ 0.0469 $K_d$ 1.5</td>
</tr>
<tr>
<td>$\theta$</td>
<td>$\begin{bmatrix} 3 &amp; 0 \ 0 &amp; 0.002 \end{bmatrix}$</td>
<td>10</td>
<td>$K_p$ 0.5477 $K_p$ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$K_d$ 0.0589 $K_d$ 1.5</td>
</tr>
<tr>
<td>$\psi$</td>
<td>$\begin{bmatrix} 1 &amp; 0 \ 0 &amp; 1 \end{bmatrix}$</td>
<td>1000</td>
<td>$K_p$ 0.0361 $K_p$ 0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$K_d$ 0.0316 $K_d$ 2.5</td>
</tr>
</tbody>
</table>

3.5.2 Altitude control

The method for altitude control is similar to the attitude control. However the thrust reference will always have a value equal to the nominal thrust. The LQR gains have the same correspondence as the attitude control, because of the similarities between the structure of $A$, $B$ and $C$. The scheme also has the adjustment gains as the attitude control.

The used tuning matrices and obtained gains for the altitude control are presented in table 5. Another difference in this scheme is that an adjustment is made to the final value, with the gain $K_{thrust}$. Section 6.4.1 presents the flight results for a hovering situation with the methods and gains presented for both attitude and altitude control.

Table 5: Control parameters for altitude control

<table>
<thead>
<tr>
<th>Parameter</th>
<th>$Q$</th>
<th>$R$</th>
<th>LQR</th>
<th>Adjustment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\begin{bmatrix} 1 &amp; 0 \ 0 &amp; 1 \end{bmatrix}$</td>
<td>0.1</td>
<td>$K_p$ 0.4472 $K_p$ 0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$K_d$ 1 $K_d$ 1.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$K_{thrust}$ 0.7</td>
</tr>
</tbody>
</table>
3.6 Outer loop

The outer loop implements position control, taking \( x, y \) position errors and transforming them into torque references for the inner loop, integrating with the image processing algorithms presented in chapter 4, as shown in figure 24.

\[ x_B \approx c_\psi x_I + s_\psi y_I \]  
\[ y_B \approx c_\psi y_I - s_\psi x_I \]  

To obtain the torques in step 3 of figure 24 a Kalman filter is applied to the positions and a PID controller is applied to each coordinate, obtaining from the \( x \) error a torque \( \tau_\theta \) and from the \( y \) error a torque \( \tau_\phi \).
3.7 Take off and landing

The control strategies used for the drone stabilization are mainly intended for use in a hovering or semi-hovering state. In this situation, the goal is to keep the references, and the error to correct is very small. However, to guide the quadrotor, there are other situations where it will be necessary to find ways of specifying these references.

The cases of take off and landing are an example of guidance, but they do not change from flight to flight, and can be implemented as standalone procedures. For this purpose, the chosen implementation is simple, relying on a state flow with small references at each time. The variable to which the reference is given is the height, $z$.

![Figure 27: Take off procedure for the Parrot AR Drone 2.0](image)

The states are programmed to change at each specified time, as shown in figure 27, the take off procedure. The height is changed in limited in steps, so that the error is not as large as to cause instability in the quadrotor. The landing procedure follows the reverse process, but in a smaller steps, since it showed to be more unstable.

Both are affected by ground effect, this is, when the quadrotor moves close to the ground a cushion of air below it produces an upward force that affects its aerodynamics. The landing also has another caveat, the dead-zone of the ultrasound sensor. Thus, in the last steps, instead of specifying a small height the rotor power is reduced, as shown in figure 28.

![Figure 28: Outer loop outputing angle references](image)

To feed the variable controlled by the procedures presented, a guidance block is implemented to control all references. At first, all references are null for quadrotor stabilization, except for $z$. However the same block will be used to feed the references to the outer loop referred before. Section 6.4.2 presents the control results for the procedures presented.
4.1 Image acquisition

For image processing applications the available hardware and its impact on the image quality have to be taken into account. In the case of Parrot’s AR Drone 2.0, the available hardware consists of two cameras: front and bottom. The base program (program.elf) makes both cameras’ streaming available at port 5555, alternately.

However, by killing program.elf, the image is lost. It is then necessary to retrieve the images in other way. Since the image will be used for real time localization and control the important features that measure quality are the available framerate, the delay until extracting useful results and the image resolution.

This section intends to explain the process that lead to the final implemented strategy, as well as to clarify the work done in the scope of the Simulink® ARDroneTarget API submitted to Mathworks, one of the contributions of this work. It also compares different methods and presents the trade-offs in image based control methods. The described methods are:

- use of the Video4Linux2 API for C code inside Simulink® blocks through the Legacy tool, in section 4.1.1;
- installation of the Gstreamer program for ARM processors, in section 4.1.2;
- assembly of a Raspberry Pi 3 with a Camera Board, for acquisition and processing, in section 4.1.3.

4.1.1 Simulink® ARDroneTarget Video Library

The base approach uses the available hardware and the already existing API - Simulink® ARDroneTarget. It has the advantage of keeping all in the same scope, and avoiding the use of additional hardware. Inside the code of the API there is already an unused set of blocks that consist of a raw implementation of the video4linux2 streaming flow [28].

However, it does not yield a good rate or a small delay and because the algorithms are used from openCV, it is difficult to implement them as blocks, since slower computations block the model. This way, the first approach to acquire image data is to use the existing blocks and extract the image from Simulink®, transmitting it to an external image processing module.

Figure 29: Simulink® scheme to output and send image over TCP
The communication blocks are then added, a TCP block for sending and one for receiving, shown in figures 29 and 30, respectively. Although it is a real time application, and UDP should be more adequate, TCP was used with a small time-out, allowing that in the receiving end the data is not mixed between frames, without blocking waiting for packets.

![Simulink scheme to receive over TCP and save the image](image)

Figure 30: Simulink® scheme to receive over TCP and save the image

The results depends on the sampling times of each block. Since according to the specifications the forward camera can work at 30 FPS, the values tried out for each block are: a smaller but close value, \( \frac{1}{20} \), a value ten times larger, \( \frac{1}{2} \), and a value ten times smaller, \( \frac{1}{200} \). To evaluate image extraction the process, presented in figure 31, is the following:

- Start a timer in the host machine and display it in the monitor together with the Video Viewer
- Build, connect and run the model, which saves the video to the file
- For a given frame, subtract the value on the Video Viewer from the value on the timer to get the delay
- For a pair of sequential frames, subtract the value of the timers to get the time between frames (\( T_f \))
- Repeat the previous two steps for four or five frames and average the values of the features

![Computing image acquisition performance results](image)

Figure 31: Computing image acquisition performance results

After analysis of the application, to avoid the overhead of passing the image as a signal inside Simulink®, a second implementation is made to combine the previous blocks to a single block. It also makes available the tuning of new parameters: the number of buffers used and the possibility of running an initial cycle to avoid frame hold-up in the buffers.

The initial cycle simply reads as much frames as possible, until no frame is left enqueued in the buffer without transferring them. This way the initially hold-up frames are discarded. The number of buffers is limited to the range \([1, 16]\), outside of which the blocks fail to perform the code.

The final implementation focuses on utilizing the maximum computation capacity of the host station. For this, it makes use of threads, and keeps the same parameters as the previous one. All results are presented in appendix B, and table 6 shows the parameters and results that represent the best achieved values, for which the image is not defective.

The presented results are obtained for the forward camera, with a resolution of \(1280 \times 720\), but the bottom camera has equivalent results for its resolution. Although each case showed improvement, none of the results are satisfactory, as discussed in appendix B.
Chapter 4. Image processing

Table 6: Best image acquisition parameters and results for Simulink ® AR Drone Target

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block sampling times (s)</td>
<td></td>
</tr>
<tr>
<td>Grab Send Receive</td>
<td>Delay $T_F$ FPS (ms)</td>
</tr>
<tr>
<td>Grab Send Receive</td>
<td></td>
</tr>
<tr>
<td>1 0.05 0.05 0.25 - - -</td>
<td>5828 948 1.05</td>
</tr>
<tr>
<td>2 - 0.05 0.05 8 1 -</td>
<td>1673 833 1.20</td>
</tr>
<tr>
<td>3 - 0.05 0.05 4 1 -</td>
<td>1212 600 1.67</td>
</tr>
</tbody>
</table>

4.1.2 Gstreamer pipeline

The need for better delay and framerate and for more flexibility in downscaling and downsampling motivates the second approach: to use the gstreamer program to transmit the image directly from the drone to the host station. This application is a linux program that has a version for ARM processors, and is easily installed on the quadrotor. It has the advantage of having modules that manage the communication with the camera and the external communication through RTP protocol.

To use the streaming abilities, the program must also be installed on the host. Both the host and the target have a running pipeline that can be composed by several modules. The target pipeline accesses the camera which is a video4linux2 source, where the queue-size, which is the number of buffers, is tunable. After this it can be downscaled and downscaled, it is encoded and sent through RTP using UDP, as shown in figure 32.

![Gstreamer pipeline elements](image.png)

Figure 32: Gstreamer pipeline elements
Chapter 4. Image processing

The host pipeline simply receives and decodes the frames and sends them to a sink, which can be a file, or a video display prepared for it. Another parameter of interest is the maximum transmission unit (MTU), this is, the maximum number of bytes transmissible at each time. The results for the image acquisition are much better than the previous, and are presented in table 7.

Table 7: Best image acquisition parameters and results for gstreamer

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue size</td>
<td>Frame resolution</td>
</tr>
<tr>
<td>4</td>
<td>640x480</td>
</tr>
</tbody>
</table>

4.1.3 Raspberry Pi Camera

The previous approaches do not show satisfactory results, thus the final approach uses additional hardware, a Raspberry Pi 3 and its RGB camera, to retrieve the image. Since the hardware has a processor with enough computation power, the openCV libraries are installed in the processor and the image processing module is implemented inside the Raspberry Pi.

![Raspberry Pi assembly scheme](image)

(a) Raspberry Pi assembly scheme

![Raspberry Pi assembled on drone](image)

(b) Raspberry Pi assembled on drone

Figure 33: Rasperry Pi assembly on top of the indoor hull of Parrot AR Drone 2.0

The first step is the assembly shown in figure 33. A polystyrene sheet is attached to the hull with four openings matching the Raspberry Pi mounting holes. The device can be fixed by tightening four bolts, keeping its weight distribution fixed. The camera is mounted with a flat cable of 30cm and to avoid shifts, its position is marked in the drone. The whole set weights about 45,23 g, which leads to the PWM calibration values presented in 3.2 and the curves presented in 6.2.

With the new implementation the delay is almost insignificant and the framerate is above 20 FPS for a resolution under 1280x720. The limitation in this implementation is actually on the image processing abilities. These highly reduce the effective rate at which useful results can be passed over to the control scheme. This trade-off is described in section 4.5. The final values used, considered the most satisfactory, are presented in table 8.

Table 8: Best image acquisition parameters and results for Raspberry Pi camera

<table>
<thead>
<tr>
<th>Frame resolution</th>
<th>Processing time (ms)</th>
<th>Effective rate (FPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>320x240</td>
<td>≈ 100</td>
<td>≈ 10</td>
</tr>
</tbody>
</table>

48
4.2 Camera calibration

The focus of the image processing algorithms is localization, which requires the use of the image 2D points to find a relation to the world 3D points. The first step in then to perform camera calibration, this is, to find the intrinsic parameters that model the camera described in 2.4. To obtain these parameters, a 2D pattern is used, a squared chess board with square size of 25 mm with 10x7 squares. To compute them, MATLAB®’s application Camera Calibration is used.

The calibration process requires at least three images, however it is recommended to use between ten and twenty images of the pattern. Thus, sixteen images are used with the relative positions shown in figure 34. The pattern positions are the real used positions, since the camera is fixed and the pattern is being moved to several positions. The obtained parameters are shown in equations 4.1a and 4.1b, respectively, the camera intrinsics matrix and the vector of distortion coefficients.

![Pattern positions and Camera equivalent positions](image)

(a) Pattern positions  (b) Camera equivalent positions

Figure 34: Camera calibration camera and pattern poses

\[
K = \begin{bmatrix}
239.3228 & -0.2198 & 166.9979 \\
0 & 255.2452 & 125.5393 \\
0 & 0 & 1.0000
\end{bmatrix} \quad (4.1a)
\]

\[
\begin{bmatrix}
0.0402 & -0.0115 & 0 & 0
\end{bmatrix}^T \quad (4.1b)
\]

The application used also gives the mean reprojection error associated to each image, as shown in figure 35. As a general rule, reprojection errors of less than one pixel are acceptable. The calibration described here is a process that only has to be completed once, since the camera parameters do not change. However, for different resolutions different camera parameters will be obtained. The presented matrices correspond to the Raspberry Pi 3 camera for a resolution of 320x240.
Chapter 4. Image processing

4.3 ArUco visual markers

In this implementation the ArUco visual markers are used as objects in the map. This allows avoiding to deal with feature extraction and focus on the main goal of the project, i.e., using object maps for localization in the context of exploration. Section 2.5 explains how the marker identification is done.

The part of the program that deals with this marker identification and pose extraction is based on the ArUco contrib module of the openCV. Firstly a dictionary has to be chosen. In this work the standard AruCo dictionary was used, with 5x5 bits and a margin of 1 bit to the edge, corresponding to dictionary 16 in the code enumerates.

The image is then processed to check for potential ArUco markers, using the `detectMarkers()` function. This function needs the dictionary used, so it can match the codification of the markers. It returns the markers ids and image coordinates for each corners, that through function `drawDetectedMarkers()` can be checked by drawing on top of the image, as shown in figure 36. The reference frames drawn belong to each marker with the origin on the center and the following axes:

- Blue axis as the $x$ axis
- Green axis as the $y$ axis
- Red axis as the $z$ axis

![Figure 36: ArUco marker detection](image)

If the detection fails to find the markers it might help to tune the parameters of the algorithm. One of the most important is the minimum, step, and maximum values for the window used to search for the markers, which might affect the ability of finding smaller or bigger ArUcos. However, it is necessary to take into account that the most exhaustive the search, the slowest the algorithm will be. The default parameters usually work fine, and are used in the final program.

It is then possible to obtain the pose of each ArUco relative to the camera, by running `estimatePoseSingleMarkers()`. This function requires the calibration results and the size of the marker. Considering $\{V\}$ the ArUco reference frame and
{C} the camera reference frame, the algorithm returns a translation vector $^VT_C$ and a rotation vector $^VR_C$, which can be transformed into a rotation matrix by applying the Rodrigues() method.

Table 9: Best combined image acquisition and processing results for Raspberry Pi camera

<table>
<thead>
<tr>
<th>Frame resolution</th>
<th>320x240</th>
<th>640x480</th>
<th>1280x720</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing time (ms)</td>
<td>88</td>
<td>190</td>
<td>408</td>
</tr>
</tbody>
</table>

In terms of computation required, the detectMarkers() is one of the methods that compose the bottleneck referred in section 4.1.3. Since it has to scan the image searching for ArUcos, the higher the resolution, the heavier the method gets. Table 9 shows the obtained results for different resolutions using the ArUco camera.
4.4 Mapping

The focus of the image processing implementation is the localization, with the mapping being done offline through a closed form solution described in 2.7.1. It follows a simple explanation of the mapping program and how to use it.

The program requires several views of different ArUcos, so it can extract the pose of each relative to the camera and to each other. A video should be filmed of the map as to have many partial views to be analyzed. An example of an ArUco map and one partial view extracted from a video frame is shown in figure 37.

![ArUco map](image)

Figure 37: ArUco map

After analyzing each frame, the program statistically constructs relative transformations between each pair of ArUcos and selects a global frame, linked to one of the markers. It then uses these relations to compute a transformation from each ArUco to the global reference frame. These results are outputted by the program as vectors, in the following way:

- **ids** A vector with the ids of the ArUcos in the map, of length \( m \)
- **mapa_RT** A vector with a structure containing a pair \((R, T)\) for a given ArUco, also of length \( m \)
- **mapa_pontos** A matrix where each row is an ArUco center \([x_c, y_c, z_c]\), with dimensions \( m \times 3 \)
- **mk_dist** A cell matrix where each cell combines the several \((R, T)\) pairs of each view constructed as follows:

\[
\text{mk\_dist}(1, 2) = \begin{bmatrix}
R_{\text{view 1}} | T_{\text{view 1}} | R_{\text{view 2}} | T_{\text{view 2}} | \cdots | R_{\text{view n}} | T_{\text{view n}}
\end{bmatrix},
\]

with dimensions \( 3 \times (4 \times n) \), where \( n \) is the number of views.

From the outputs it is easy to represent the world map in a simple 3D MATLAB plot, and check if the map is computed correctly. For this, not only the centers are used, but also the corners are computed and plotted, and the ArUco ids are drawn, overlapping each center. The result for the previous presented map with 11 ArUcos is shown in figure 38.

![Aruco map represented in MATLAB](image)

Figure 38: Aruco map represented in MATLAB
4.5 Pose extraction

As described in the previous section the mapping is stored as a set of ids, rotations and translation matrices that correspond to the object poses on the global reference frame. To perform localization with this map, it is necessary to have any partial view with at least one of the ArUcos presented.

The first process that has to run on a frame is ArUco detection to return the found ArUcos in the frame and the respective corners. Then with the ids the map positions can be computed with the rotation and translation matrices. Using `solvePnP()` the transformation that describes the pose of the camera in the world can be found, while accounting for outliers, this is, wrongly identified ArUcos that do not belong to the map.

At this point the matrices $^{B}R_{V}$ and $^{B}T_{V}$ are available, that transform coordinates in the ArUco map reference frame to the camera reference frame. To get the quadrotor pose in the ArUco reference frame, it is necessary to invert the rotation matrix and compute the correspondent translation vector as shown in equations (4.2) and (4.3) and in figure 39.

$$^{V}R_{B} = ^{B}R_{V}^{-1}$$

(4.2)

$$^{V}T_{B} = -^{B}R_{V}^{-1}^{B}T_{V}$$

(4.3)

![Figure 39: Reference frame transformations to obtain the quadrotor position](image)

The ArUco reference frame can be used as global frame or any other can be selected with a constant transformation to the ArUco map. For this application, in which the main coordinates of interest are $x$ and $y$, the first position of the quadrotor is considered global, and an assumption is made that a partial view of the map is available from the beginning.

Another possibility in this application would be to start the quadrotor in a point where the ArUcos would be visible almost at start in a known position and transform that to a desired global frame. Both this possibilities are represented in figure 39. Equations (4.4) and (4.5) present the computation of the final matrices for the first case.

$$^{I}R_{B} = ^{I}R_{V}^{V}R_{B}$$

(4.4)

$$^{I}T_{B} = ^{I}R_{V}^{V}T_{B} + ^{I}T_{V}$$

(4.5)
Chapter 4. Image processing

4.6 Code flow and organization

To implement pose extraction the code is run through a C++ program in the raspberry. The configuration files and the resulting map are stored in the same location as the script and the script is triggered to run when the raspberry turns on. The code is threaded as to take advantage of the maximum computation possible and it is structured as follows:

**Main thread** Deals with all external communication
- Throws thread 1 for image acquisition and thread 2 for pose estimation
- Sends pose estimation at a constant rate to the UDP block shown in figure 25 in section 3.6

**Thread 1** Acquires image from the camera
- Camera set up - frame resolution, framerate and format negotiation
- Stores frame in shared memory

**Thread 2** Pose estimation
- Reads map and stores each object \(R, T\) and computed world points
- Copies frame in shared memory
- Runs Aruco detection
- Runs solvePnP with 2D points and world points, obtaining \(\mathbf{B}_R\) and \(\mathbf{B}_T\)
- Computations transformations to desired reference frame \(\mathbf{I}_R\) and \(\mathbf{I}_T\)

Figure 40 sums up the code flow of the program, pointing out the bottleneck in thread 2, where the detection and pose estimation algorithms run. These depend on the resolution and on the number of iterations of the solvePnP procedure.

![Figure 40: Code flow of the image algorithm implementation](image)

Several tests were performed for different resolutions, analyzing the effective rate and the quality of the measures. The quality of the measurements is evaluated after the Kalman filter, since this is the actual value that will go into the loop. The results with better balance between time and quality is presented in table 10, for a still test and a moving test.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame resolution</td>
<td>Number of iterations</td>
</tr>
<tr>
<td>Still</td>
<td>320x240</td>
</tr>
<tr>
<td>Moving</td>
<td>320x240</td>
</tr>
</tbody>
</table>
5 Exploration and coverage

5.1 Mapping using confidence levels

The second problem tackled in this thesis can be defined as follows: to study optimal or suboptimal algorithms to map a space with a desired level of confidence and minimal cost. This cost can be the distance traveled, time or energy spent in mapping the space. This problem has some resemblances to the prevailing coverage problem, with some nuances.

For a general formulation, the space from where an agent can map is considered N-dimensional and the space to be mapped M-dimensional. However, in the physical world, the agents often move in 3D or 2D spaces, with constrains, and the spaces to be mapped or investigated will have similar constrains. In section 5.1.1 the problem is structured and defined in a mathematical way, followed by a 2D narrowing in section 5.2 to some interesting examples.

5.1.1 Problem formalization

The context of this problem comprises a finite space, which can be discretized into smaller part. A level of confidence is then associated with each part, with the following meaning: a part with higher level of confidence is better known and a part with lower confidence is less well known. It can thus be seen as a reverse of a measure of uncertainty.

Although different scales can be used, an intuitive scale is a confidence level between zero and one, where zero means the part is fully unknown and one the part is fully known. To simplify the problem it will be assumed that at any time, the mapping agent has full knowledge of its position.

Equation (5.1) presents a vector of confidence for the whole space, assuming there are \( n \) parts, at time \( k \). A state representing the state of the agent can also be defined as shown in (5.2). It can include information about the pose of the agent, for example.

\[
c^{(k)} = [c_1^{(k)}, c_2^{(k)}, \ldots, c_n^{(k)}]
\]

\[
r^{x^{(k)}} = [r_{x_1^{(k)}}, r_{x_2^{(k)}}, \ldots, r_{x_n^{(k)}}]
\]

The confidence of each part in an instant \( k + 1 \) can then be described by a function involving the previous confidence vector and the previous state, as shown in (5.3).

\[
c_i^{(k+1)} = f_c (c_i^{(k)}, r^{x^{(k)}})
\]
Considering an agent has a finite set of action that can choose at time $k$ then the state at $k + 1$ will depend on the previous state and the given action it chooses, leading to the relation in equation (5.4), where $U^{(k)}$ is the set of possible actions at time $k$.

$$r^{(k+1)} = f_d \left( r^{(k)}, u^{(k)} \right), \quad u^{(k)} \in U^{(k)} \quad (5.4)$$

The problem can be stated as a DP optimization problem. From (5.1) and (5.2), a state $x$ can be defined as shown in (5.5), with an update function composed by the previous from (5.3) and (5.4), as shown in (5.6).

$$x^{(k)} = \left[ r^{(k)}, c^{(k)} \right] = \left[ c_1^{(k)}, c_2^{(k)}, ..., c_m^{(k)}, c_1^{(k)}, c_2^{(k)}, ..., c_n^{(k)} \right] \quad (5.5)$$

$$x^{(k+1)} = f(x^{(k)}, u^{(k)}) = \left[ f_d \left( r^{(k)}, u^{(k)} \right), f_c \left( c^{(k)}, r^{(k)} \right) \right] \quad (5.6)$$

So far, the definitions presented are elements of the environment and the agent, and need to be written to represent the situation properly. Since the goal is to increase the confidence in the map with minimum distance/time cost, a cost function can be defined to account for this variables, that computes for each action at time $k$ the cost of applying it, as stated in (5.7), where $\mu(x)$ is a mapping from a state to an action $u_k$.

$$\forall u^{(k)} = \mu(x^{(k)}), \quad a_{ij}^k = G(x^{(k)}, \mu(x^{(k)})) \quad (5.7)$$

The goal state can be described in two ways: a first approach might be to reach some level of confidence over the whole map; a different approach might be to maximize the confidence over a finite time-horizon. The first can be described by some condition similar to equation (5.8): for a given state $x$, the agent has reached its goal if the sum of all confidence levels is higher than some constant $C$ in a finite time horizon $L$. The second can be described by (5.9).

$$x^{(L)} : \sum_{i=1}^{n} c_i^{(L)} > C \quad (5.8)$$

$$x^{(L)} : \max_{(u^{(1)}, u^{(2)}, ..., u^{(L-1)})} \sum_{i=1}^{n} c_i^{(L)} \quad (5.9)$$

The coverage problem described in 2.8.1 can actually be seen as a special case of this problem, with some simplifications:

- The confidence takes only two different values or states and its meaning can be seen as whether a specific part of the map has been visited or not;
- The goal is defined so as all the map has to be visited.
5.2 Exploration and coverage scenarios

There are several scenarios that can be taken into account, and each will lead to differences in the functions and vectors that compose the problem. In this thesis, a relevant scenario is that of covering a given space from above by using a downward camera with different levels of confidence for each part. The goal will be to maximize the confidence in the space, thus exploring the environment. This scenario will be referred to as ground coverage.

5.2.1 Ground coverage

Supposing the camera can move in a plane parallel to the 2-D space being mapped, at a distance \(d\), without changing its orientation, then the state, which can be seen as the pose, is given by \(c = [x, y]\) and the composed state vector is given by (5.5) with actions \(u_k\) at any given time \(k\) belonging to the set \(U\) in (5.11).

\[
\begin{align*}
    c^{(k)}_1, c^{(k)}_2, \ldots, c^{(k)}_n, x, y \\
u^{(k)} \in U = \{\text{up, down, left, right, stay}\}
\end{align*}
\]

Function \(f\) can be obtained by tackling the confidence update and the pose update separately, as shown in (5.12) and (5.13). In (5.12), \(v\) is a function that given the agent state and a given cell, returns \(True\) if the cell is visible by the agent.

Note that not all actions are allowed at each step \(k\). When the agent is near the boundary, for example, it is not possible to move in some directions.

\[
\begin{align*}
c^{(k+1)}_i &= \begin{cases} 
    c^{(k)}_i + \lambda v(c^{(k)}_i, \mathbf{x}^{(k)}) & \text{if } u^{(k)} = \text{stay} \\
    c^{(k)}_i & \text{otherwise}
\end{cases} \\
\mathbf{e}^{(k+1)}_x &= \begin{cases} 
    0 & \text{if } u^{(k)} = \text{stay} \\
    \mathbf{e}^{(k)}_x + \begin{bmatrix} 1 & 0 \end{bmatrix} & \text{if } u^{(k)} = \text{right} \\
    \mathbf{e}^{(k)}_x + \begin{bmatrix} -1 & 0 \end{bmatrix} & \text{if } u^{(k)} = \text{left} \\
    \mathbf{e}^{(k)}_x + \begin{bmatrix} 0 & 1 \end{bmatrix} & \text{if } u^{(k)} = \text{up} \\
    \mathbf{e}^{(k)}_x + \begin{bmatrix} 0 & -1 \end{bmatrix} & \text{if } u^{(k)} = \text{down}
\end{cases}
\end{align*}
\]

Considering a simple cost function that takes into account the cost of each action in (5.14), the cost at time \(k\) is given by equation (5.15) and the total cost of a sequence of actions is given by equation (5.16). The goal is to reach a desired state, such as the one in equation (5.8), while minimizing the total cost, as defined in (5.17).

\[
\begin{align*}
l(\mathbf{x}^{(k)}, u^{(k)}) &= \begin{cases} 
    0 & \text{if } u^{(k)} = \text{stay} \\
    1 & \text{otherwise}
\end{cases} \\
G(\mathbf{x}^{(k)}, u^{(k)}) &= l(\mathbf{x}^{(k)}, u^{(k)}) + \sum_{i=1}^{n} \alpha_i g(c^{(k)}_i) \\
J_\pi(x_0) &= G^{(N)}(\mathbf{x}^{(N)}) + \sum_{k=0}^{N-1} G(\mathbf{x}^{(k)}, u^{(k)}) \\
J^*(x_0) &= \min_{\pi} J_\pi(x_0)
\end{align*}
\]
5.3 Offline coverage algorithms

One of the optimal algorithms most used in literature is A*. However, as discussed in sections 2.8.1 and 2.9, this problem is an NP-hard optimization problem which can be shown to be at least as hard as the CSP, and consequently, the TSP, requiring sub-optimal or approximation algorithms to compute solutions in useful time.

The first proposed algorithm is an approximation to the A* with a finite horizon. Like the A* it will branch out by comparing a score of the sum of cost until the node and a heuristic until the goal. However, it will stop at a given level and simply pick the node with best score, repeating the process until it finds the goal. In this sense, it can also be seen as somewhat close to best-first search.

**Algorithm 5.1:** Best-first in horizon

```
1 begin
2     OPEN = ∅
3     CLOSE = ∅
4     Add(S, OPEN) /* S starting node */
5     bestfscore = ∞
6     bestNode = S
7     while OPEN ≠ ∅ do
8         V ← node with minimum fscore
9         Remove(V, OPEN)
10        Add(V, CLOSE)
11        if IsGoal(V) then
12            return reconstructPath(V)
13        end
14        if IsLeaf(V) then
15            if fscore(V) ≤ bestfscore then
16                bestfscore = fscore(V)
17                bestNode = V
18            end
19        end
20        for each neighbor Q of V do
21            if Q ∈ CLOSE then
22                continue
23            end
24            tentative_gscore = gscore(V) + cost(V, Q)
25            if Q ∉ OPEN then
26                Add(Q, OPEN)
27            end
28            if tentative_gscore ≥ gscore(Q) then
29                continue
30            end
31            gscore(Q) = tentative_gscore
32            fscore(Q) = gscore(Q) + heuristic(Q)
33            parent(Q) = V
34        end
35     end
36 end
```

The algorithm will not have memory issues and is in most cases fast, but it has an important caveat. It can get stuck in a loop if a given area has already been completely covered and thus it will not find a direction. One possible solution is to increase the level of depth until it breaks the loop. The pseudo code of this algorithm is presented in procedure 5.1. The results will also depend on the chosen heuristic, such as the A*, and the horizon chosen in the begin reflects a trade-off between speed of the algorithm and the probability of the approximation being closer to the optimal solution.
Chapter 5. Exploration and coverage

The second proposed algorithm is based on the random sampling-based coverage [12] described in section 2.8.1. In a first approach the algorithm tries to find a set of states that allow complete coverage of the goal state, which is similar to solving the art gallery problem, also an NP-hard problem. The solution here is to randomly select nodes until all neighbours are covered, which will clearly be a sufficient set but also probably far from minimal.

To avoid to much redundancy, after constructing the set, an iteration is made over it to remove the unnecessary nodes. For each node, if its remotion does not destroy the sufficient set, it means it is redundant and it is taken out. For this algorithm, only the path affecting attributes of the state matter, this is, the confidence levels can be ignored when comparing states. The algorithm is described in procedure 5.2.

Algorithm 5.2: Random coverage set

```plaintext
1 begin
2    MIN = ∅
3    ALL = set of all path nodes
4    Add(S,MIN) /* S starting node */
5 while ALL ⊈ neighbours(MIN) do
6        V = random(ALL)
7        if V \∈ MIN then
8            Add(V,MIN)
9        end
10    end
11 for each node V ∈ MIN do
12        if ALL ⊆ MIN \ V then
13            Remove(V,MIN)
14        end
15    end
16 end
```

After computing the pseudo-minimal set that covers all nodes, an order should be defined so as to go through all nodes. This is the problem of the traveling salesman, which is solved with a simple algorithm, the nearest neighbour algorithm, that from the starting node simply picks the nearest available neighbour, as shown in procedure 5.3.

Algorithm 5.3: Nearest-neighbor

```plaintext
1 begin
2    OPEN = ∅
3    CLOSE = ∅
4    Add(S,OPEN) /* S starting node */
5 while OPEN ≠ ∅ do
6        V ← node with minimum gscore
7        Remove(V,OPEN)
8        Add(V,CLOSE)
9        if IsGoal(V) then
10            return reconstructPath(V)
11        end
12 for each neighbor Q of V do
13        if Q ∈ CLOSE then
14            continue
15        tentative_gscore = cost(V,Q)
16        gscore(Q) = tentative_gscore
17        parent(Q) = V
18        Add(Q,OPEN)
19    end
20 end
21```

59
Although there are certain cases where the algorithm can lead to a worst case scenario, in the average case it works well enough and it is very fast to compute. To retrieve the complete path it is simple enough to run any path finding optimal algorithm between the two neighbours, since it is a problem with very reduced complexity. The final step is to compute the overnights, which can be done by simply assuming that the confidence in a view must be higher than $C$ in each area.

### 5.3.1 Ground coverage heuristic

Both the A* and the best first in horizon need an heuristic and the better the heuristic represents the distance to the goal, the better the algorithm. For the ground coverage scenario the chosen heuristic is the following. Given a grid of size $w \times h$ and a view of range $r$, then the number of squares inside the view range is given by (5.18).

$$sq = (2r + 1)^2$$ (5.18)

Let $V_k$ be the set of all cells inside this view range at a time step $k$. The total average confidence in the grid is given by (5.19) and the minimum confidence inside the current view is defined in (5.20).

$$C = \sum_{i} c_i \frac{w \times h}{w \times h}$$ (5.19)

$$C_{min}^r = \min_{i \in V_k} c_i$$ (5.20)

The first consideration is that the minimum number of time steps required to stay in the current view to increase its confidence to the desired level is then underestimated as shown in (5.21), by considering the number of necessary overnights to bring the cell with minimum confidence to the goal confidence value, where $C_{\text{step}}$ is the defined confidence step.

$$o_v = \text{ceil} \left( \frac{G - C_{min}^r}{C_{\text{step}}} \right)$$ (5.21)

The total number of overnights in the grid is calculated in a similar manner, as shown in (5.22).

$$o_T = \text{ceil} \left( \frac{w \times h (G - C)}{C_{\text{step}} \times sq} \right)$$ (5.22)

If the total number of overnights is bigger than the number of overnights required inside the view, then the heuristic can have an additional cost of movement. The first addition is simply a cost of one, which is a movement different than the overnight to change the view.

$$h = o_T + 1$$ (5.23)

Then for the remaining overnights the following considerations are made: for a given view, the maximum number of overnights spent is the inverse of the confidence step, since for any confidence level this would maximize the confidence; hence dividing the number of external overnights by this number the result is the minimum number of movements to shift the view.

$$h = h + \text{ceil} \left( (o_T - o_v) \times C_{\text{step}} \right)$$ (5.24)
6 Simulations and results

6.1 Sensor calibration

For the accelerometer the variations of the calibrated values, the offset and gain, is not considerable between flights and so the calibration is only performed once before the flight tests. Thus, the values used are here presented in equations (6.1) and (6.2). The accelerometer values are used throughout the control tests here presented.

\[
\text{acc}^{\text{bias}} = \begin{bmatrix} -2052 & -2012 & -2079 \end{bmatrix}^T 
\]

(6.1)

\[
\text{acc}^{\text{scale}} = \begin{bmatrix} -0.01879 & 0.01916 & 0.01909 \end{bmatrix}^T 
\]

(6.2)

As for the gyroscope offset, the variation is larger, and therefore its values are calibrated before each flight. However, the proportions between each component and the order of magnitude of the bias are similar to the example shown in (6.3). The gyroscope values also tend to stabilize after a number of tests, which is one of the expected behaviours of this sensor. The gyroscope gain is fixed, though, and presented in (6.4).

\[
\text{gyro}^{\text{bias}} \approx \begin{bmatrix} -7.35 & -52.89 & -41.82 \end{bmatrix}^T 
\]

(6.3)

\[
\text{gyro}^{\text{scale}} = \begin{bmatrix} 1.0723 & -1.0723 & 1.0723 \end{bmatrix}^T 
\]

(6.4)

The negative values in both gains are a way of adjusting the measures to the desired reference frame, since the gyroscope and accelerometer axis do not match this reference frame.

6.2 PWM modelation

The obtained curves for PWM calibration for the assembly shown in 4.1.3 are shown in figure 41. It is possible to observe that the obtained equations are quite different, which in a hovering point will lead to the different values, as pointed in the graph and shown in table 11. These are obtained considering a mass of approximately 470g.
The curves that correspond to the case with the Raspberry 3 on top of the quadrotor are presented in figure 42. Once again, there is a significant difference between each equation, which reflects in the hovering values, pointed in the graph and shown in table 11. These values consider an approximate mass of 530 g.

Another important point is that the weight distribution is shifted, which is visible on the curves’ equations and the hovering values. This might be a destabilization factor for the quadrotor, since its motors already run at a high PWM value, and it might not be possible to satisfy any adjustment because of power limitations.

Table 11: Rotor hovering point for the quadrotor without (case 1) and with a raspberry Pi 3 (case 2)

<table>
<thead>
<tr>
<th></th>
<th>Mass</th>
<th>Rotor speed</th>
<th>Rotor 1</th>
<th>Rotor 2</th>
<th>Rotor 3</th>
<th>Rotor 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>≈ 470</td>
<td>1.152763</td>
<td>63</td>
<td>70</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>Case 2</td>
<td>≈ 530</td>
<td>1.2998</td>
<td>70</td>
<td>72</td>
<td>69</td>
<td>77</td>
</tr>
</tbody>
</table>
Chapter 6. Simulations and results

6.3 Kalman filter

Before using the Kalman filter estimators for attitude in flight they were tuned by manually rotating the quadrotor and comparing the results before and after the Kalman filter. The tests are performed attached to a board with the rotors functioning at a near-hovering speed, to simulate the vibrations in flight. Figures 43a and 43b show these results.

![Graphs showing results for φ and P, and θ and Q before and after Kalman filter.](image)

Figure 43: Results for φ and θ before (blue) and after (red) Kalman filter

The Kalman filter takes out most of the noise and avoids sudden changes of great amplitude, while following the angles well. The second test performed evaluates the impact of one of the attitude angles in the other, and as such, θ is changed between different amplitudes at different rates, while keeping φ unaltered. Figures 44a and 44b show the obtained results, which show that the Kalman filter decreases the impact observed in the original measures.

![Graphs showing Kalman filter results on the impact of moving θ in φ measurements.](image)

Figure 44: Kalman filter results on the impact of moving θ in φ measurements

Overall, it is visible that the measurements are very affected by vibrations and, although the Kalman estimator can predict the angles well, these vibrations are expected to have greater impact on flight, since the tests were performed with the quadrotor attached to a light board, for better manoeuvering.
Chapter 6. Simulations and results

6.4 Quadrotor stabilization

6.4.1 Continuous hovering

As described in section 3.5, the first control test performed was simply a take-off and a continuous hover state. The angle stabilization is shown in figures 45 and 46. In these it is possible to see that the angle values stay very low, and tend to reduce over time, although slowly.

In both graphs the same is also visible to the velocity, which oscillates around zero. Overall, these are satisfactory results for the attitude control using the tuning values presented in 3.5.
Chapter 6. Simulations and results

Figure 47: Z and W for hover flight test

Regarding the altitude values, the altitude reference tracking is shown in figure 47, together with the corresponding linear velocity. The values are negative since the $z$ axis is pointing down, and so the quadrotor moves in the negative direction. The abrupt change visible is simply the point at which the test was stopped, due to the quadrotor getting dangerously close of some object that might damage it.

The fact that an approximation happens so fast is a drawback, since longer tests would provide more information. It happens because of the quadrotor horizontal drift. Because the quadrotor does not have position control and the error tends to be in the same directions, when compensating it, the quadrotor drifts.

Figure 48: Psi and R for hover flight test

This drift usually happens in a biased direction. In the room used for testing, with about 4 meters width, and taking in account the width of the quadrotor, about 0.5 meters, the drift rapidly takes the quadrotor close to the boundaries. One important point is that this drift is not predictable, mainly because of the results related to $\psi$, and also because of other effects, such as air currents and battery charge impact on the rotor power.
These show that the $\psi$ angle velocity is controlled and approximates the zero reference given, which means the quadrotor does not spin, but the angle does not go to zero, this is, the quadrotor stops moving at a fixed $\psi$ error. Several test flights yield different stopping values. Thus, the drift relative to a fixed reference frame will also vary.

![Figure 49: Forces and torques values for hover flight test](image1)

Finally, the output values of the controller, namely the thrust and torques involved, and the corresponding PWMs, are shown in figures 49 and 50, respectively. Although rotors 2, 3 and 4 show approximate values to the deduced hover values in 6.2, rotor 1 seems to be higher than expected. However it needs to be taken into account that for the stabilization the compensation of errors can shift the PWM. Overall, throughout all the testing it was observed that a good PWM modelation helps in stabilizing the quadrotor.

![Figure 50: PWM values for hover flight test](image2)
6.4.2 Takeoff and landing

The results for $\phi$ and $\dot{\phi}$ and $\theta$ and $\dot{\theta}$ are shown in figures 51 and 52. It is possible to see that the values remain small throughout the test, excepting the final part of the landing, which is discussed further on. Unlike the previous test, however, it is not visible in these graphs the damping of the oscillations, since there was no hovering period. This is related to the short length of the tests, already mentioned.

![Figure 51: Phi and P for take off and landing flight test](image1)

![Figure 52: Theta and Q for take off and landing flight test](image2)
Chapter 6. Simulations and results

As for the $\psi$ angle and its velocity $\dot{\psi}$, the results are very similar to the previous test, with the same explanation as before. One point that is worth pointing out, though, is the wide variation in the final moments of the landing, as observed also in $\phi$ and $\theta$. The altitude results, shown in figure 54, seem to track the reference well until the final moments of the landing.

![Figure 53: Psi and R for take off and landing flight test](image)

This moment of instability in the landing is probably related to the dead-zone of the ultrassound sensor, which cannot return measurements when too close to the obstacle. Figure 54 shows the results for $Z$ and $W$ in this same test, for which the measurements show a smooth descent till zero, which supports the idea that the sensor returns a zero value before it is exactly on the floor, thus leading to that instability.

![Figure 54: Z and W for take off and landing flight test](image)
The thrust and torques generated and the corresponding PWM values are shown in figures 55 and 56. These results are quite similar to the previous test, with the exception of having more variations on the thrust, as expected.

Figure 55: Forces and torques values for take off and landing flight test

Figure 56: PWM values for take off and landing flight test
6.5 Image processing

For analyzing the image processing results, two different sets of tests were made: the first consisting of simply keeping the camera still while pointing to the same set of ArUcos and see the variation in the results and the second while moving the camera a given distance while observing ArUcos, to retrieve the Kalman measurements and check its quality vs. the processing time required to get the measurements.

6.5.1 Still camera

The test with the still camera was done with the raspberry Pi camera facing a set of 11 ArUcos, each with the dimensions 3.9x3.9cm, at a distance of about 40cm. The results regarding the computing time and the variation of each coordinate are shown in table 12, for several different values of resolution and different numbers of iterations.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Iterations</th>
<th>Processing time</th>
<th>FPS</th>
<th>Variation Range</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1280x720</td>
<td>5</td>
<td>0.408</td>
<td>2.45</td>
<td>.05236 .04014 .01550</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.413</td>
<td>2.42</td>
<td>.04210 .03192 .01658</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>0.428</td>
<td>2.34</td>
<td>.04249 .03051 .01462</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>0.466</td>
<td>2.15</td>
<td>.04431 .03024 .01982</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.512</td>
<td>1.95</td>
<td>.03855 .03314 .01783</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0.636</td>
<td>1.57</td>
<td>.03053 .02494 .01324</td>
</tr>
<tr>
<td>640x480</td>
<td>5</td>
<td>0.190</td>
<td>5.26</td>
<td>.06331 .27690 .1707</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.205</td>
<td>4.88</td>
<td>.02576 .02549 .01353</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>0.215</td>
<td>4.65</td>
<td>.02255 .01895 .01299</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>0.221</td>
<td>4.52</td>
<td>.02417 .02033 .01266</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.221</td>
<td>4.52</td>
<td>.02581 .02033 .01067</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0.221</td>
<td>4.52</td>
<td>.02570 .01583 .01087</td>
</tr>
<tr>
<td>320x240</td>
<td>5</td>
<td>0.086</td>
<td>11.63</td>
<td>.06508 .3748 .1954</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.086</td>
<td>11.63</td>
<td>.04868 .04237 .0288</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>0.087</td>
<td>11.49</td>
<td>.04868 .04237 .0288</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>0.088</td>
<td>11.36</td>
<td>.04868 .04237 .0288</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.088</td>
<td>11.36</td>
<td>.04868 .04237 .0288</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0.088</td>
<td>11.36</td>
<td>.04868 .04237 .0288</td>
</tr>
</tbody>
</table>

It is possible to see that for the smallest resolution the measurements and processing time almost stay the same throughout most of the test, with the exception of the smallest number of iterations. This probably happens because in a smaller resolution the pixel variation between frames will be insignificant and the estimate of the previous frame given will already be very close to the pose. Thus, the stop criteria will probably not be the iterations but the difference between each iteration pose.
Chapter 6. Simulations and results

Another observation is that for any resolution there is a "knee" in the 10 iterations, as seen in figure 57. This already points to the fact that when selecting the parameters, there will be a minimum number of iterations to get satisfactory results. As for the remaining results, since for control applications it is better to have a higher rate, the most satisfactory set of parameters would then be a resolution of $320 \times 240$ with 100 iterations in solvePnP. However, the final parameters are derived also from the moving test.

![Figure 57: Rasperry Pi's ArUco pose estimation accuracy for a still camera](image)

6.5.2 Moving camera

The second test consists of moving the camera in the $x$ axis about 14cm while maintaining a partial view of the same map of 11 ArUcos with 3, 9x3, 9cm. The measurements then pass through a tuned Kalman filter that reduces the noise of the estimates. The results for $1280 \times 720$, $640 \times 480$ and $320 \times 240$ resolutions are shown in figures 58, 59 and 60, respectively.

In this test one of the observed results is that the measured time for the algorithms is very close to the one in the still test, with a variation smaller than 20ms in the highest resolution, and smaller than 5ms in the lowest resolution. This is probably due to the test having a very slow movement. Although for the lowest resolution the iterations seem to not matter, for the higher resolution this value increases the time significantly. Extrapolating from this, a moderate value for the iterations is chosen, namely 30 iterations.
Chapter 6. Simulations and results

Figure 58: Kalman filter results for the raspberry camera (1280 × 720) moving 14 cm in the $x$ coordinate.

Figure 59: Kalman filter results for the raspberry camera (640 × 480) moving 14 cm in the $x$ coordinate.
Chapter 6. Simulations and results

Figure 60: Kalman filter results for the raspberry camera (320 × 240) moving 14 cm in the $x$ coordinate

The first result, coherent with the static results, is that for a small number of iterations there is clearly a larger variation that is not so easily compensated by the Kalman filter. However, the results for more than 30 iterations seem similar in all tests, and this is the value chosen for the parameter. Regarding resolutions, there is a visible difference from 640 × 480 to 320 × 240, especially in the $y$ and $z$ coordinates, that ideally would be null.

Table 13 shows the ranges results for the ”static” variables, $x$ and $y$. However, it is necessary to consider that the three tests might have small variations caused by different camera motions, since they are performed manually. They still give a good base for tuning the parameters and of the tradeoff between accuracy and effective rate, and as such the chosen values are a resolution of 320 × 240 and 30 iterations in solvePnP.

Table 13: Rasperry Pi’s ArUco pose estimation accuracy for a still moving camera

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Iterations</th>
<th>Variation Range</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$y$</td>
</tr>
<tr>
<td>1280x720</td>
<td>30</td>
<td>.04891</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>.05379</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>.0463</td>
</tr>
<tr>
<td>640x480</td>
<td>30</td>
<td>.02868</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>.02941</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>.03498</td>
</tr>
<tr>
<td>320x240</td>
<td>30</td>
<td>.074833</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>.07687</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>.0742</td>
</tr>
</tbody>
</table>
6.6 Vision outer loop

A final test for the image-based localization and the outer loop is to feed the position from the image processing algorithms over a larger map to the outer loop and see the the body frame coordinates computed with $x$, $y$ and $\psi$ from the camera, as well as the control torques fed to the inner loop. Figures 61a and 61b show the results for the image coordinates with $\psi$ compared against the angle given by integration with the rotors shutdown this is, without vibrations.

![Figure 61: Camera localization results for larger map, with baseline integrated $\psi$](image)

The results for $\psi$ and the coordinates $x$ and $y$ follow the actual route performed quite well. Figures 62a and 62b then present the corresponding bodyframe coordinates and the torque provoked by them.

![Figure 62: Body frame coordinate transformation and outer loop results](image)

The bodyframe coordinates are computed well, transforming the $x$ movements with $+90$ degrees and $-90$ degrees into $y$ movements. As for the torques, they opposite the bodyframe coordinates proportionally, an expected result. These results are satisfactory and benefit of an important aspect, its error is not time-dependent, thus staying similar over time.
6.7 Covering algorithms

6.7.1 Ground coverage

To test the algorithms presented in 5.3 in the example of ground coverage, which can easily be applied to the quadrotor exploration problem, several tests are performed. The variables that can be set for each scenario are the grid size, the initial level of confidence in each cell, the confidence step, this is, the increase in confidence an overnight in a cell brings to the visible cells, the view range and the goal confidence.

For simplicity purposes and to compare with the optimal solution, which is only computable in useful time for non-complex scenarios, the first test consists of using several different grid sizes where all cells have 0.8 confidence, the confidence step is 0.2 and the desired confidence is 1 in every cell, requiring only one overnight to get a range of cells into the goal. The view range is considered to be 1, this is, if the camera is on top of one cell, it sees every cell around that one, a $3 \times 3$ square.

The three algorithms were applied, namely the optimal A* algorithm, the best-first search in horizon with initial horizon of 6 steps - both with the heuristic described in 5.3.1 - and the random based coverage algorithm with nearest neighbour. The random-based algorithm is ran four times for each test and the best solution is picked. The results for small grids are shown in Table 14.

<table>
<thead>
<tr>
<th>Grid size</th>
<th>A*</th>
<th>BFS Horizon</th>
<th>Random based</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x4</td>
<td>7 steps</td>
<td>14 steps</td>
<td>7 steps</td>
</tr>
<tr>
<td></td>
<td>14 ms</td>
<td>12 ms</td>
<td>18 ms</td>
</tr>
<tr>
<td>4x5</td>
<td>8 steps</td>
<td>14 steps</td>
<td>10 steps</td>
</tr>
<tr>
<td></td>
<td>38 ms</td>
<td>14 ms</td>
<td>15 ms</td>
</tr>
<tr>
<td>5x5</td>
<td>10 steps</td>
<td>15 steps</td>
<td>12 steps</td>
</tr>
<tr>
<td></td>
<td>243 ms</td>
<td>106 ms</td>
<td>19 ms</td>
</tr>
<tr>
<td>5x6</td>
<td>11 steps</td>
<td>21 steps</td>
<td>11 steps</td>
</tr>
<tr>
<td></td>
<td>723 ms</td>
<td>130 ms</td>
<td>18 ms</td>
</tr>
<tr>
<td>6x6</td>
<td>13 steps</td>
<td>15 steps</td>
<td>15 steps</td>
</tr>
<tr>
<td></td>
<td>15.787 s</td>
<td>35 ms</td>
<td>24 ms</td>
</tr>
<tr>
<td>7x6</td>
<td>17 steps</td>
<td>37 steps</td>
<td>20 steps</td>
</tr>
<tr>
<td></td>
<td>1h18m36.877s</td>
<td>335 ms</td>
<td>24 ms</td>
</tr>
<tr>
<td>7x7</td>
<td>Could not compute</td>
<td>31 steps</td>
<td>25 steps</td>
</tr>
<tr>
<td></td>
<td>in useful time (¿24h)</td>
<td>1.548s</td>
<td>19 ms</td>
</tr>
</tbody>
</table>

It is clear from this analysis that the optimal algorithm has exponential time, as expected, and both others suboptimal do not. However, the best first search within horizon can fall into the loop described earlier, and because of that, when increasing the horizon, it can increase its computation time a lot, which explains the jump in time observed for both grids $5 \times 5$ and $5 \times 6$. As for the random, the computed times are the total time to run the algorithm four times, and the best solution clearly produced good solutions for these scenarios.

An example of the $6 \times 6$ grid is shown for further analysis. Figures 63a and 63b present the initial setting of the problem and the optimal solution, respectively. This scenario is very intuitive and thus easy to analyze. The corner numbers presented in each cell are the number of computed overnights. The optimal solution serves as a base for comparison.
Chapter 6. Simulations and results

(a) Initial setup of the algorithm
(b) Optimal (A*) solution of the problem

Figure 63: Ground coverage problem for a 6 × 6 grid

Figure 64a shows the result for the best first in horizon algorithm, and figure 64b the results for the random based coverage. Both approximate the optimal result well, but each as an unnecessary deviation.

(a) Best first in horizon
(b) Random-based coverage (best of four)

Figure 64: Ground coverage sub-optimal algorithms results for a 6 × 6 grid

It is also interesting to analyze some of the other solutions the random based coverage produces, since unlike its counterpart, it is not a deterministic algorithm. Figures 65a and 65b present two other random based results. These results are not as good as the presented best random based solution. The algorithm is thus uncertain, since it could produce four bad solutions. However, statistically it is probable that at least one of the solutions produced is not a worst case scenario.

(a) Random-based coverage medium scenario
(b) Random-based coverage worst scenario

Figure 65: Ground coverage random-based coverage results for a 6 × 6 grid
Chapter 6. Simulations and results

Other tests for comparison analyzed more complex scenarios just for the two approximation algorithms, since the optimal A* algorithm does not present a solution in useful time. The first is simply a much larger grid, with $12 \times 9$, thus 108 cells, with results shown in figures 66 and 67, where once again the random based coverage produced a better result than the best first in horizon, although both presented some overlapping.

![Base scenario](image1)
![Random-based coverage result](image2)

Figure 66: Ground coverage test in a $12 \times 9$ grid

![Middle of execution](image3)
![Final result](image4)

Figure 67: Ground coverage best first in horizon result sequence for a $12 \times 9$ grid

Other increase in complexity is the use of different steps of confidence values. The simple initialization in a value that requires more than one confidence step update per cell increases the complexity in a visible way. Table 15 presents the results for a grid of medium size, $9 \times 9$, with different confidence levels.

Table 15: Algorithm path length and time results for ground coverage problem with higher complexity

<table>
<thead>
<tr>
<th>Initial confidence</th>
<th>BFS Horizon</th>
<th>Random based</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>63 steps / 3.841s</td>
<td>58 steps / 41 ms</td>
</tr>
<tr>
<td>0.6</td>
<td>84 steps / 5.112s</td>
<td>65 steps / 45 ms</td>
</tr>
<tr>
<td>0.4</td>
<td>86 steps / 9.135s</td>
<td>81 steps / 44 ms</td>
</tr>
<tr>
<td>0.2</td>
<td>81 steps / 2.916s</td>
<td>77 steps / 30 ms</td>
</tr>
<tr>
<td>0.0</td>
<td>81 steps / 6.430s</td>
<td>96 steps / 29 ms</td>
</tr>
</tbody>
</table>
Chapter 6. Simulations and results

The random based coverage seems to produce better and faster results for the presented cases. This result is very positive for the algorithm, especially because there are still improvements that can be done, described in 7. However, this is not conclusive for the Best first search in horizon, because even though its results are worse than the random based coverage, it is highly dependent on the heuristics and it can also be improved with other strategy to break cycles.

Extending the algorithm to a more interesting case, it is possible to have different areas with different levels of interest, by specifying different initial levels of confidence for each cell and a more loose goal, where the average confidence is high, but does not require all cells to have maximum confidence. Figure 68 presents an example for which the goal confidence is 0.9 and the darker and lighter areas have 0.2 and 0.7 of confidence, respectively. The confidence step is still 0.2. Figure 69 presents the results for both algorithms.

![Figure 68: Ground coverage test with different levels of confidence in the same map](image)

Although random base coverage still gives a better result, with 46 steps obtained in 30ms, as opposed to the 54 steps of the best first in horizon obtained in 2.212s, the path it takes seems to be less directed than its counterpart. In fact, the best first in horizon seems to be attracted to the areas with less confidence, and the random based coverage simply picks a path and adjusts its number of overnights.

![Figure 69: Ground coverage results for different levels of confidence in the same map](image)
7.1 Discussion and future work

The present thesis focuses on the exploration problem, this is, the task of guiding a robot in such a way that it covers an entire environment with its sensors. It proposes an implementation on a quadrotor, with the use of control and image processing algorithms for stabilization and guidance. The first contribution is the bottom-up approach presented, which can be summarized by the scheme in figure 70. Each module was firstly implemented and tested and, according to the results, they achieved the desired behavior, separately.

The control strategy described in chapter 3 is the PID-LQR method that combines the PID robustness to steady-state error and the closed loop stability of the LQR. The results show that the quadrotor stabilizes promptly, without substantial oscillations. A simple PID strategy could also achieve the same results, however the use of the PID-LQR made the tuning process more intuitive. Another important aspect is that the $\psi$ angle only stabilizes at a given offset, because of the incomplete sensing information. This offset can be corrected with the image measurements.
Chapter 7. Conclusions

Since the image acquisition is done with a Raspberry Pi 3, rotor modulation was applied also to the quadrotor with the Raspberry assembled, and stabilization was achieved. This new configuration required the rotors working close to their maximum thrust, which prevents the correction of greater disruptions. In particular, an increase of this effect was observed at the end of the thesis. Because the rotors showed degradation, supported payload diminished. Thus, the rotors could no longer support the Raspberry, since even smaller disruptions with its weight destabilize the quadrotor, which detained the initially planned implementation. To join all modules, a possible future implementation should consider a quadrotor with higher payload.

The image algorithms were implemented firstly in the quadrotor cameras, and then in the Raspberry, in three stages:

- use of the Video4Linux2 API for C code inside Simulink® blocks through the Legacy tool, which showed delay and rate values not adequate to a real time application;
- installation of the Gstreamer program for ARM processors, which improved with respect to the previous implementation, but still lacked lower delay values to prove useful in real time;
- assembly of a Raspberry Pi 3 with a Camera Board, for acquisition and processing, which showed a good balance between delay and rate values and the resolution and its localization estimates.

Because of the performance achieved with the improvements to the Simulink® blocks and the gstreamer program, which showed better results than the previous implementations, Mathworks adopted some of the ideas. However, the available on-board hardware does not enable real time localization. The future work should focus on the improvement of the algorithms with this hardware, either by enriching the blocks with the ability of down-sampling and downsampling the image, or by exploiting and optimizing the use of the Gstreamer application. Both strategies can also benefit from the use of encoding in the transmission process.

As referred, the final strategy produced nice pose estimates with satisfactory delay and framerate. The estimate errors are small and they do not increase over time. Thus, another natural step in image processing is to replace the visual markers by real objects and test the performance of module. This requires real-time object recognition and pose estimation methods for the new objects.

Regarding exploration, the thesis presented an innovative formulation of the coverage path planning problem, with the insertion of a measure of utility of each portion of the environment. This measure of utility introduces a degree of knowledge of each cell, that makes the algorithms favor less known areas and map the environment more efficiently.

The presented coverage approximation algorithms compute satisfactory solutions in useful time with the desired behavior, in particular for the best first in horizon algorithm. However, the strategies could be improved: in the best first in horizon algorithm a better heuristic can improve velocity; and in the random-based coverage a better path planner to go through the chosen nodes could improve the result, since the nearest neighbour is not an efficient algorithm. The coverage algorithms should be implemented on-board in a low cost quadrotor, which involves expanding the coverage path planning algorithms to on-line strategies and to study the problem without assuming full knowledge of the state, accounting for some uncertainty.


PWM measurements
### Appendix A. PWM measurements

Table 16: PWM calibration for quadrotor without payload (case 1)

<table>
<thead>
<tr>
<th>PWM</th>
<th>Rotor 1</th>
<th></th>
<th>Rotor 2</th>
<th></th>
<th>Rotor 3</th>
<th></th>
<th>Rotor 4</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W</td>
<td>T</td>
<td>W</td>
<td>T</td>
<td>W</td>
<td>T</td>
<td>W</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td>446.6</td>
<td>.126549</td>
<td>446.3</td>
<td>.130473</td>
<td>447.0</td>
<td>.122625</td>
<td>447.3</td>
<td>.118701</td>
</tr>
<tr>
<td>2</td>
<td>445.8</td>
<td>.134397</td>
<td>445.6</td>
<td>.137340</td>
<td>446.3</td>
<td>.129492</td>
<td>446.6</td>
<td>.125568</td>
</tr>
<tr>
<td>3</td>
<td>445.0</td>
<td>.142245</td>
<td>444.8</td>
<td>.145188</td>
<td>445.8</td>
<td>.134397</td>
<td>445.7</td>
<td>.134397</td>
</tr>
<tr>
<td>4</td>
<td>444.2</td>
<td>.150093</td>
<td>443.8</td>
<td>.154998</td>
<td>445.0</td>
<td>.142245</td>
<td>445.0</td>
<td>.141264</td>
</tr>
<tr>
<td>5</td>
<td>443.4</td>
<td>.157941</td>
<td>443.0</td>
<td>.162846</td>
<td>444.2</td>
<td>.150093</td>
<td>444.4</td>
<td>.147150</td>
</tr>
<tr>
<td>6</td>
<td>442.6</td>
<td>.165789</td>
<td>442.0</td>
<td>.172656</td>
<td>443.4</td>
<td>.157941</td>
<td>443.7</td>
<td>.154017</td>
</tr>
<tr>
<td>7</td>
<td>441.8</td>
<td>.173637</td>
<td>441.1</td>
<td>.181485</td>
<td>442.6</td>
<td>.165789</td>
<td>442.9</td>
<td>.161865</td>
</tr>
<tr>
<td>8</td>
<td>440.8</td>
<td>.183447</td>
<td>440.2</td>
<td>.190314</td>
<td>441.7</td>
<td>.174618</td>
<td>442.2</td>
<td>.168732</td>
</tr>
<tr>
<td>9</td>
<td>439.7</td>
<td>.194238</td>
<td>439.1</td>
<td>.201105</td>
<td>440.9</td>
<td>.182466</td>
<td>441.3</td>
<td>.177561</td>
</tr>
<tr>
<td>10</td>
<td>438.6</td>
<td>.205029</td>
<td>438.3</td>
<td>.208953</td>
<td>439.7</td>
<td>.194238</td>
<td>440.3</td>
<td>.187371</td>
</tr>
<tr>
<td>11</td>
<td>437.7</td>
<td>.213858</td>
<td>437.2</td>
<td>.219744</td>
<td>438.9</td>
<td>.202086</td>
<td>439.3</td>
<td>.197181</td>
</tr>
<tr>
<td>12</td>
<td>436.8</td>
<td>.222687</td>
<td>436.1</td>
<td>.230535</td>
<td>438.1</td>
<td>.209934</td>
<td>438.5</td>
<td>.205029</td>
</tr>
<tr>
<td>13</td>
<td>435.6</td>
<td>.234459</td>
<td>434.8</td>
<td>.243288</td>
<td>437.2</td>
<td>.218763</td>
<td>437.6</td>
<td>.213858</td>
</tr>
<tr>
<td>14</td>
<td>434.7</td>
<td>.243288</td>
<td>433.7</td>
<td>.254079</td>
<td>436.2</td>
<td>.228573</td>
<td>436.7</td>
<td>.222687</td>
</tr>
<tr>
<td>15</td>
<td>433.7</td>
<td>.253098</td>
<td>433.0</td>
<td>.260946</td>
<td>435.2</td>
<td>.238383</td>
<td>435.5</td>
<td>.234459</td>
</tr>
<tr>
<td>16</td>
<td>432.5</td>
<td>.264870</td>
<td>431.9</td>
<td>.271737</td>
<td>434.2</td>
<td>.248193</td>
<td>434.3</td>
<td>.246231</td>
</tr>
<tr>
<td>17</td>
<td>431.4</td>
<td>.275661</td>
<td>430.8</td>
<td>.282528</td>
<td>433.2</td>
<td>.258003</td>
<td>433.6</td>
<td>.253098</td>
</tr>
<tr>
<td>18</td>
<td>430.2</td>
<td>.287433</td>
<td>429.9</td>
<td>.291357</td>
<td>432.3</td>
<td>.266832</td>
<td>432.6</td>
<td>.262908</td>
</tr>
<tr>
<td>19</td>
<td>428.8</td>
<td>.301167</td>
<td>428.4</td>
<td>.306072</td>
<td>430.9</td>
<td>.280566</td>
<td>431.3</td>
<td>.275661</td>
</tr>
<tr>
<td>20</td>
<td>427.6</td>
<td>.312939</td>
<td>427.0</td>
<td>.319806</td>
<td>429.8</td>
<td>.291357</td>
<td>430.3</td>
<td>.285961</td>
</tr>
<tr>
<td>21</td>
<td>426.5</td>
<td>.323730</td>
<td>425.5</td>
<td>.334521</td>
<td>428.6</td>
<td>.303129</td>
<td>429.8</td>
<td>.290866</td>
</tr>
<tr>
<td>22</td>
<td>424.9</td>
<td>.339426</td>
<td>424.5</td>
<td>.344331</td>
<td>427.6</td>
<td>.312939</td>
<td>428.8</td>
<td>.300677</td>
</tr>
<tr>
<td>23</td>
<td>423.8</td>
<td>.350217</td>
<td>422.7</td>
<td>.361989</td>
<td>426.5</td>
<td>.323730</td>
<td>427.7</td>
<td>.311468</td>
</tr>
<tr>
<td>24</td>
<td>422.6</td>
<td>.361989</td>
<td>421.3</td>
<td>.375723</td>
<td>425.5</td>
<td>.333540</td>
<td>426.6</td>
<td>.322258</td>
</tr>
<tr>
<td>25</td>
<td>421.2</td>
<td>.375723</td>
<td>420.3</td>
<td>.385533</td>
<td>424.2</td>
<td>.346293</td>
<td>425.5</td>
<td>.333049</td>
</tr>
<tr>
<td>26</td>
<td>419.6</td>
<td>.391419</td>
<td>419.2</td>
<td>.396324</td>
<td>423.0</td>
<td>.358065</td>
<td>424.4</td>
<td>.343841</td>
</tr>
<tr>
<td>27</td>
<td>418.2</td>
<td>.405153</td>
<td>417.7</td>
<td>.411039</td>
<td>421.6</td>
<td>.371799</td>
<td>423.3</td>
<td>.354631</td>
</tr>
<tr>
<td>28</td>
<td>416.6</td>
<td>.420849</td>
<td>416.0</td>
<td>.427716</td>
<td>420.1</td>
<td>.386514</td>
<td>422.0</td>
<td>.367384</td>
</tr>
<tr>
<td>29</td>
<td>415.1</td>
<td>.435564</td>
<td>414.5</td>
<td>.442431</td>
<td>419.2</td>
<td>.395343</td>
<td>420.8</td>
<td>.379156</td>
</tr>
<tr>
<td>30</td>
<td>413.8</td>
<td>.448317</td>
<td>413.2</td>
<td>.455184</td>
<td>417.8</td>
<td>.409077</td>
<td>419.3</td>
<td>.393871</td>
</tr>
<tr>
<td>31</td>
<td>412.0</td>
<td>.465975</td>
<td>411.2</td>
<td>.474804</td>
<td>416.4</td>
<td>.422811</td>
<td>418.0</td>
<td>.406625</td>
</tr>
<tr>
<td>32</td>
<td>410.5</td>
<td>.480690</td>
<td>409.3</td>
<td>.493443</td>
<td>414.9</td>
<td>.437526</td>
<td>416.5</td>
<td>.421340</td>
</tr>
<tr>
<td>33</td>
<td>409.2</td>
<td>.493443</td>
<td>408.0</td>
<td>.506196</td>
<td>413.6</td>
<td>.450279</td>
<td>415.2</td>
<td>.434093</td>
</tr>
<tr>
<td>34</td>
<td>407.5</td>
<td>.510120</td>
<td>406.5</td>
<td>.520911</td>
<td>412.3</td>
<td>.463032</td>
<td>413.9</td>
<td>.446846</td>
</tr>
<tr>
<td>35</td>
<td>405.8</td>
<td>.526797</td>
<td>404.2</td>
<td>.543474</td>
<td>410.8</td>
<td>.477747</td>
<td>412.4</td>
<td>.461561</td>
</tr>
<tr>
<td>36</td>
<td>404.1</td>
<td>.543474</td>
<td>402.8</td>
<td>.557208</td>
<td>409.3</td>
<td>.492462</td>
<td>410.6</td>
<td>.479218</td>
</tr>
<tr>
<td>37</td>
<td>402.0</td>
<td>.564075</td>
<td>400.4</td>
<td>.580752</td>
<td>407.6</td>
<td>.509139</td>
<td>409.3</td>
<td>.491972</td>
</tr>
<tr>
<td>38</td>
<td>399.9</td>
<td>.584676</td>
<td>398.6</td>
<td>.598410</td>
<td>406.2</td>
<td>.522873</td>
<td>408.2</td>
<td>.502763</td>
</tr>
<tr>
<td>39</td>
<td>397.9</td>
<td>.604296</td>
<td>396.3</td>
<td>.620973</td>
<td>404.0</td>
<td>.544455</td>
<td>406.2</td>
<td>.522383</td>
</tr>
<tr>
<td>40</td>
<td>396.3</td>
<td>.619992</td>
<td>394.4</td>
<td>.639612</td>
<td>403.2</td>
<td>.552303</td>
<td>405.0</td>
<td>.534155</td>
</tr>
<tr>
<td>41</td>
<td>394.4</td>
<td>.638631</td>
<td>392.3</td>
<td>.660213</td>
<td>401.6</td>
<td>.567999</td>
<td>403.4</td>
<td>.549851</td>
</tr>
</tbody>
</table>

(Continued on next page)
<table>
<thead>
<tr>
<th>PWM</th>
<th>Rotor 1</th>
<th>Rotor 2</th>
<th>Rotor 3</th>
<th>Rotor 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W</td>
<td>T</td>
<td>W</td>
<td>T</td>
</tr>
<tr>
<td>42</td>
<td>392.8</td>
<td>.654327</td>
<td>391.6</td>
<td>.667080</td>
</tr>
<tr>
<td>43</td>
<td>390.5</td>
<td>.676890</td>
<td>390.4</td>
<td>.678852</td>
</tr>
<tr>
<td>44</td>
<td>389.3</td>
<td>.688662</td>
<td>387.4</td>
<td>.708282</td>
</tr>
<tr>
<td>45</td>
<td>386.3</td>
<td>.718092</td>
<td>385.9</td>
<td>.722997</td>
</tr>
<tr>
<td>46</td>
<td>384.2</td>
<td>.738693</td>
<td>384.6</td>
<td>.735750</td>
</tr>
<tr>
<td>47</td>
<td>382.3</td>
<td>.759294</td>
<td>381.4</td>
<td>.767142</td>
</tr>
<tr>
<td>48</td>
<td>379.8</td>
<td>.783819</td>
<td>379.8</td>
<td>.782838</td>
</tr>
<tr>
<td>49</td>
<td>378.4</td>
<td>.797553</td>
<td>377.7</td>
<td>.803439</td>
</tr>
<tr>
<td>50</td>
<td>375.0</td>
<td>.830907</td>
<td>374.4</td>
<td>.835812</td>
</tr>
</tbody>
</table>
### Appendix A. PWM measurements

#### Table 17: PWM calibration for quadrotor with raspberry payload (case 2)

| PWM | Rotor 1 | | Rotor 2 | | Rotor 3 | | Rotor 4 | |
|-----|---------|---|---------|---|---------|---|---------|
|     | W       | T |         |   |         |   |         |   |
| ... | ...     |  | ...     |   | ...     |   | ...     |   |
| 7   | 509.3   | .176188 | 509.9 | .170302 | - | - | - | - |
| 8   | 508.5   | .184036 | 509.0 | .179131 | - | - | - | 510.0 | .169321 |
| 9   | 507.9   | .189922 | 508.2 | .186979 | 509.4 | .175207 | 508.8 | .181093 |
| 10  | 506.6   | .202675 | 507.0 | .198751 | 507.8 | .190903 | 508.2 | .186979 |
| 11  | 505.3   | .215428 | 506.0 | .208561 | 507.0 | .198751 | 507.3 | .195808 |
| 12  | 504.4   | .224257 | 505.0 | .218371 | 506.2 | .206599 | 506.4 | .204637 |
| 13  | 503.4   | .234067 | 504.3 | .225238 | 505.3 | .215428 | 505.8 | .210523 |
| 14  | 502.6   | .241915 | 503.2 | .236029 | 504.5 | .223276 | 504.8 | .220333 |
| 15  | 501.0   | .257611 | 502.0 | .247801 | 502.3 | .244858 | 503.5 | .233086 |
| 16  | 500.2   | .265459 | 500.8 | .259573 | 501.4 | .253687 | 502.8 | .239953 |
| 17  | 499.0   | .277231 | 500.0 | .267421 | 500.7 | .260554 | 502.0 | .247801 |
| 18  | 497.8   | .289003 | 499.0 | .277231 | 499.6 | .271345 | 501.4 | .253687 |
| 19  | 496.9   | .297832 | 497.6 | .290965 | 498.0 | .287041 | 499.8 | .269383 |
| 20  | 495.6   | .305859 | 496.0 | .306661 | 497.5 | .291946 | 499.0 | .277231 |
| 21  | 493.9   | .327262 | 495.0 | .316471 | 495.8 | .308623 | 498.7 | .280174 |
| 22  | 492.9   | .337072 | 493.7 | .329224 | 494.0 | .326281 | 498.0 | .287041 |
| 23  | 491.5   | .350806 | 492.3 | .342958 | 493.8 | .328243 | 496.6 | .300775 |
| 24  | 489.9   | .366502 | 491.5 | .350806 | 492.0 | .345901 | 495.5 | .311566 |
| 25  | 488.0   | .385141 | 490.5 | .360616 | 490.8 | .357673 | 494.2 | .324319 |
| 26  | 486.2   | .402799 | 489.8 | .367483 | 489.5 | .370426 | 493.0 | .336091 |
| 27  | 485.0   | .414571 | 488.7 | .378274 | 488.2 | .383179 | 492.0 | .345901 |
| 28  | 483.7   | .427324 | 487.3 | .392008 | 485.7 | .407704 | 491.0 | .355711 |
| 29  | 481.8   | .445963 | 486.5 | .399856 | 484.3 | .421438 | 489.5 | .370426 |
| 30  | 479.8   | .465583 | 484.6 | .418495 | 483.6 | .428305 | 488.0 | .385141 |
| 31  | 478.8   | .475393 | 483.0 | .434191 | 481.6 | .447925 | 487.5 | .390046 |
| 32  | 477.6   | .487165 | 481.6 | .447925 | 480.0 | .463621 | 486.0 | .404761 |
| 33  | 475.0   | .512671 | 480.3 | .460678 | 479.0 | .473431 | 485.0 | .414571 |
| 34  | 472.5   | .537196 | 479.0 | .473431 | 477.8 | .485203 | 483.8 | .426343 |
| 35  | 471.5   | .547006 | 477.5 | .488146 | 476.5 | .497956 | 482.5 | .439096 |
| 36  | 470.5   | .556816 | 475.8 | .504823 | 474.3 | .519538 | 481.7 | .446944 |
| 37  | 469.0   | .571531 | 473.8 | .524443 | 472.9 | .533272 | 480.0 | .463621 |
| 38  | 466.4   | .597037 | 472.2 | .540139 | 471.0 | .551911 | 479.0 | .473431 |
| 39  | 465.0   | .610771 | 471.0 | .551911 | 469.0 | .571531 | 477.5 | .488146 |
| 40  | 463.3   | .627448 | 469.8 | .563683 | 466.9 | .592132 | 476.0 | .502861 |
| 41  | 461.0   | .650011 | 467.5 | .586246 | 465.5 | .605866 | 475.6 | .506785 |
| 42  | 459.5   | .664726 | 465.8 | .602923 | 463.3 | .627448 | 473.4 | .528367 |
| 43  | 458.3   | .676498 | 464.5 | .615676 | 460.8 | .651973 | 472.0 | .542101 |
| 44  | 456.5   | .694156 | 462.5 | .635296 | 458.0 | .679441 | 470.3 | .558778 |
| 45  | 453.5   | .723586 | 460.9 | .650992 | 457.0 | .689251 | 468.8 | .573493 |
| 46  | 452.9   | .729472 | 458.0 | .679441 | 453.8 | .720643 | 466.5 | .596056 |

(Continued on next page)
Table 17 – continued from previous page

<table>
<thead>
<tr>
<th>PWM</th>
<th>Rotor 1</th>
<th></th>
<th>Rotor 2</th>
<th></th>
<th>Rotor 3</th>
<th></th>
<th>Rotor 4</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W</td>
<td>T</td>
<td>W</td>
<td>T</td>
<td>W</td>
<td>T</td>
<td>W</td>
<td>T</td>
</tr>
<tr>
<td>47</td>
<td>450.0</td>
<td>.757921</td>
<td>456.0</td>
<td>.699061</td>
<td>452.0</td>
<td>.738301</td>
<td>465.0</td>
<td>.610771</td>
</tr>
<tr>
<td>48</td>
<td>448.5</td>
<td>.772636</td>
<td>454.0</td>
<td>.718681</td>
<td>450.0</td>
<td>.757921</td>
<td>462.7</td>
<td>.633334</td>
</tr>
<tr>
<td>49</td>
<td>446.0</td>
<td>.797161</td>
<td>452.5</td>
<td>.733396</td>
<td>448.0</td>
<td>.777541</td>
<td>460.7</td>
<td>.652954</td>
</tr>
<tr>
<td>50</td>
<td>444.7</td>
<td>.809914</td>
<td>450.8</td>
<td>.750073</td>
<td>446.0</td>
<td>.797161</td>
<td>460.0</td>
<td>.659821</td>
</tr>
<tr>
<td>51</td>
<td>443.8</td>
<td>.818743</td>
<td>448.0</td>
<td>.777541</td>
<td>443.0</td>
<td>.826591</td>
<td>458.0</td>
<td>.679441</td>
</tr>
<tr>
<td>52</td>
<td>441.0</td>
<td>.846211</td>
<td>447.0</td>
<td>.787351</td>
<td>442.0</td>
<td>.836401</td>
<td>455.0</td>
<td>.708871</td>
</tr>
<tr>
<td>53</td>
<td>440.0</td>
<td>.856021</td>
<td>445.0</td>
<td>.806971</td>
<td>440.0</td>
<td>.856021</td>
<td>452.5</td>
<td>.733396</td>
</tr>
<tr>
<td>54</td>
<td>437.8</td>
<td>.877603</td>
<td>443.5</td>
<td>.821686</td>
<td>437.0</td>
<td>.885451</td>
<td>450.0</td>
<td>.757921</td>
</tr>
<tr>
<td>55</td>
<td>434.0</td>
<td>.914881</td>
<td>442.0</td>
<td>.836401</td>
<td>434.7</td>
<td>.908014</td>
<td>448.3</td>
<td>.774598</td>
</tr>
<tr>
<td>56</td>
<td>431.8</td>
<td>.936463</td>
<td>440.0</td>
<td>.856021</td>
<td>430.9</td>
<td>.945292</td>
<td>446.3</td>
<td>.794218</td>
</tr>
<tr>
<td>57</td>
<td>430.0</td>
<td>.954121</td>
<td>438.0</td>
<td>.875641</td>
<td>430.0</td>
<td>.954121</td>
<td>442.5</td>
<td>.831496</td>
</tr>
<tr>
<td>58</td>
<td>427.8</td>
<td>.975703</td>
<td>436.0</td>
<td>.895261</td>
<td>427.0</td>
<td>.983551</td>
<td>441.0</td>
<td>.846211</td>
</tr>
<tr>
<td>59</td>
<td>426.0</td>
<td>.993361</td>
<td>433.0</td>
<td>.924691</td>
<td>425.0</td>
<td>1.003171</td>
<td>439.0</td>
<td>.865831</td>
</tr>
<tr>
<td>60</td>
<td>423.2</td>
<td>1.020829</td>
<td>430.0</td>
<td>.954121</td>
<td>420.0</td>
<td>1.052221</td>
<td>438.0</td>
<td>.875641</td>
</tr>
<tr>
<td>61</td>
<td>421.0</td>
<td>1.042411</td>
<td>427.0</td>
<td>.983551</td>
<td>-</td>
<td>-</td>
<td>437.0</td>
<td>.885451</td>
</tr>
<tr>
<td>62</td>
<td>417.5</td>
<td>1.076746</td>
<td>426.0</td>
<td>.993361</td>
<td>-</td>
<td>-</td>
<td>434.0</td>
<td>.914881</td>
</tr>
<tr>
<td>63</td>
<td>414.0</td>
<td>1.111081</td>
<td>423.0</td>
<td>1.022791</td>
<td>-</td>
<td>-</td>
<td>430.0</td>
<td>.954121</td>
</tr>
<tr>
<td>64</td>
<td>411.5</td>
<td>1.135606</td>
<td>420.0</td>
<td>1.052221</td>
<td>-</td>
<td>-</td>
<td>429.0</td>
<td>.963931</td>
</tr>
<tr>
<td>65</td>
<td>409.8</td>
<td>1.152283</td>
<td>417.0</td>
<td>1.081651</td>
<td>-</td>
<td>-</td>
<td>426.0</td>
<td>.993361</td>
</tr>
<tr>
<td>66</td>
<td>408.0</td>
<td>1.169941</td>
<td>414.0</td>
<td>1.111081</td>
<td>-</td>
<td>-</td>
<td>424.0</td>
<td>1.012981</td>
</tr>
<tr>
<td>67</td>
<td>404.6</td>
<td>1.203295</td>
<td>412.0</td>
<td>1.130701</td>
<td>-</td>
<td>-</td>
<td>422.0</td>
<td>1.032601</td>
</tr>
<tr>
<td>68</td>
<td>403.0</td>
<td>1.218991</td>
<td>410.0</td>
<td>1.150321</td>
<td>-</td>
<td>-</td>
<td>419.0</td>
<td>1.062031</td>
</tr>
<tr>
<td>69</td>
<td>400.0</td>
<td>1.248421</td>
<td>408.0</td>
<td>1.169941</td>
<td>-</td>
<td>-</td>
<td>418.0</td>
<td>1.071841</td>
</tr>
<tr>
<td>70</td>
<td>397.0</td>
<td>1.277851</td>
<td>405.0</td>
<td>1.199371</td>
<td>-</td>
<td>-</td>
<td>415.0</td>
<td>1.101271</td>
</tr>
<tr>
<td>71</td>
<td>396.0</td>
<td>1.287661</td>
<td>402.0</td>
<td>1.228801</td>
<td>-</td>
<td>-</td>
<td>412.0</td>
<td>1.130701</td>
</tr>
<tr>
<td>72</td>
<td>393.6</td>
<td>1.311205</td>
<td>397.0</td>
<td>1.277851</td>
<td>-</td>
<td>-</td>
<td>410.0</td>
<td>1.150321</td>
</tr>
<tr>
<td>73</td>
<td>389.0</td>
<td>1.356331</td>
<td>393.0</td>
<td>1.317091</td>
<td>-</td>
<td>-</td>
<td>406.0</td>
<td>1.189661</td>
</tr>
<tr>
<td>74</td>
<td>385.5</td>
<td>1.390666</td>
<td>389.0</td>
<td>1.356331</td>
<td>-</td>
<td>-</td>
<td>404.0</td>
<td>1.209181</td>
</tr>
<tr>
<td>75</td>
<td>382.0</td>
<td>1.425001</td>
<td>383.0</td>
<td>1.415191</td>
<td>-</td>
<td>-</td>
<td>402.0</td>
<td>1.228801</td>
</tr>
<tr>
<td>76</td>
<td>379.5</td>
<td>1.449526</td>
<td>379.0</td>
<td>1.454431</td>
<td>-</td>
<td>-</td>
<td>398.0</td>
<td>1.268041</td>
</tr>
<tr>
<td>77</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>395.0</td>
<td>1.297471</td>
</tr>
<tr>
<td>78</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>390.0</td>
<td>1.346521</td>
</tr>
<tr>
<td>79</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>388.8</td>
<td>1.358293</td>
</tr>
<tr>
<td>80</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>387.6</td>
<td>1.370065</td>
</tr>
<tr>
<td>81</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>385.3</td>
<td>1.392628</td>
</tr>
<tr>
<td>82</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>382.0</td>
<td>1.425001</td>
</tr>
<tr>
<td>83</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>380.0</td>
<td>1.444621</td>
</tr>
<tr>
<td>84</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>378.0</td>
<td>1.464241</td>
</tr>
<tr>
<td>85</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>376.0</td>
<td>1.483861</td>
</tr>
<tr>
<td>86</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>370.0</td>
<td>1.542721</td>
</tr>
</tbody>
</table>
Image acquisition measures
### Appendix B. Image acquisition measures

Table 18: Image acquisition results for quadrotor front camera (case 1)

<table>
<thead>
<tr>
<th>Tuning</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>0.050</td>
<td>0.050</td>
</tr>
<tr>
<td>0.050</td>
<td>0.050</td>
</tr>
<tr>
<td>0.050</td>
<td>0.005</td>
</tr>
<tr>
<td>0.050</td>
<td>0.005</td>
</tr>
<tr>
<td>0.005</td>
<td>0.050</td>
</tr>
<tr>
<td>0.005</td>
<td>0.050</td>
</tr>
<tr>
<td>0.005</td>
<td>0.005</td>
</tr>
<tr>
<td>0.005</td>
<td>0.005</td>
</tr>
<tr>
<td>0.250</td>
<td>0.250</td>
</tr>
<tr>
<td>0.250</td>
<td>0.250</td>
</tr>
<tr>
<td>0.250</td>
<td>0.050</td>
</tr>
<tr>
<td>0.250</td>
<td>0.050</td>
</tr>
<tr>
<td>0.050</td>
<td>0.250</td>
</tr>
<tr>
<td>0.050</td>
<td>0.250</td>
</tr>
<tr>
<td>0.050</td>
<td>0.050</td>
</tr>
</tbody>
</table>

Table 19: Image acquisition results for quadrotor front camera (case 2)

<table>
<thead>
<tr>
<th>Tuning</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>0.050</td>
<td>0.050</td>
</tr>
<tr>
<td>0.050</td>
<td>0.050</td>
</tr>
<tr>
<td>0.050</td>
<td>0.050</td>
</tr>
<tr>
<td>0.050</td>
<td>0.050</td>
</tr>
<tr>
<td>0.050</td>
<td>0.005</td>
</tr>
<tr>
<td>0.005</td>
<td>1/200</td>
</tr>
<tr>
<td>0.005</td>
<td>1/200</td>
</tr>
<tr>
<td>0.250</td>
<td>0.250</td>
</tr>
<tr>
<td>0.250</td>
<td>0.250</td>
</tr>
</tbody>
</table>
Table 20: Image acquisition results for quadrotor front camera (case 3)

<table>
<thead>
<tr>
<th>Tuning</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive sample time</td>
<td>Grab and send sample time</td>
</tr>
<tr>
<td>0.050</td>
<td>0.050</td>
</tr>
<tr>
<td>0.050</td>
<td>0.050</td>
</tr>
<tr>
<td>0.050</td>
<td>0.050</td>
</tr>
<tr>
<td>0.050</td>
<td>0.050</td>
</tr>
<tr>
<td>0.050</td>
<td>0.005</td>
</tr>
<tr>
<td>0.005</td>
<td>1/200</td>
</tr>
<tr>
<td>0.005</td>
<td>1/200</td>
</tr>
<tr>
<td>0.250</td>
<td>0.250</td>
</tr>
<tr>
<td>0.250</td>
<td>0.250</td>
</tr>
</tbody>
</table>