Learning Autonomous Quadcopter Trajectories from Demonstration

BY

LUCA GRAGLIA
Laurea, Politecnico di Torino, Turin, Italy, 2013

THESIS
Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2015

Chicago, Illinois

Defense Committee:

Miloš Žefran, Advisor
Brian Ziebart, Chair
Luciano Lavagno, Politecnico di Torino
To the people i love,

thanks to them i am the person i am.
ACKNOWLEDGMENTS

First and foremost i would like to express my sincere gratitude to all the members of the committee and in particular to Prof. Brian Ziebart for the support he gave me during my thesis project. His guidance helped during all the steps of this thesis, from the research to the writing part.

I am grateful and i would like to thank Mathew Monfort for the assistance he has provided me in numerous occasions.

My sincere thanks also goes to my girlfriend, Francesca, and all my friends who have always been close to me even if the material distance among us has been enormous.

Last but not the least, I would like to thank my parents Alessandra Arossa and Francesco Graglia, for supporting me in all the way it has been possible for them throughout my all life.

LG
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2 PREVIOUS WORK</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Corridor and Stair Following</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Road-Following</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Elementary Motion Detectors</td>
<td>5</td>
</tr>
<tr>
<td>2.4 Obstacle avoidance</td>
<td>6</td>
</tr>
<tr>
<td>2.5 RRT Paths Planner</td>
<td>6</td>
</tr>
<tr>
<td>2.6 Scale-Aware Navigation with a Monocular Camera</td>
<td>7</td>
</tr>
<tr>
<td>2.7 Simultaneous localization and mapping</td>
<td>7</td>
</tr>
<tr>
<td>3 BACKGROUND</td>
<td>9</td>
</tr>
<tr>
<td>3.1 Quadcopter</td>
<td>9</td>
</tr>
<tr>
<td>3.1.1 Parrot AR Drone 2.0</td>
<td>10</td>
</tr>
<tr>
<td>3.1.1.1 Components</td>
<td>11</td>
</tr>
<tr>
<td>3.1.1.1.1 Cameras</td>
<td>12</td>
</tr>
<tr>
<td>3.1.1.1.1.1 Frontal Camera</td>
<td>12</td>
</tr>
<tr>
<td>3.1.1.1.1.2 vertical camera</td>
<td>13</td>
</tr>
<tr>
<td>3.1.1.1.2 gyroscope</td>
<td>14</td>
</tr>
<tr>
<td>3.1.1.1.3 accelerometer</td>
<td>15</td>
</tr>
<tr>
<td>3.1.1.1.4 magnetometer</td>
<td>15</td>
</tr>
<tr>
<td>3.1.1.1.5 ultrasound sensor</td>
<td>15</td>
</tr>
<tr>
<td>3.1.1.1.6 Pressure Sensor</td>
<td>15</td>
</tr>
<tr>
<td>3.1.1.1.7 other technical detail</td>
<td>15</td>
</tr>
<tr>
<td>3.1.1.2 Drone communication channel</td>
<td>17</td>
</tr>
<tr>
<td>3.1.1.2.1 AT Commands</td>
<td>17</td>
</tr>
<tr>
<td>3.1.1.2.2 Navigation Data</td>
<td>18</td>
</tr>
<tr>
<td>3.1.1.2.3 Video Stream</td>
<td>20</td>
</tr>
<tr>
<td>3.1.1.2.4 Control Port</td>
<td>21</td>
</tr>
<tr>
<td>3.2 Node</td>
<td>21</td>
</tr>
<tr>
<td>3.2.1 Nodecopter Modules</td>
<td>24</td>
</tr>
<tr>
<td>3.2.1.1 node-ar-drone</td>
<td>24</td>
</tr>
<tr>
<td>3.2.1.2 ardrone-autonomy</td>
<td>25</td>
</tr>
<tr>
<td>3.3 EKF</td>
<td>25</td>
</tr>
<tr>
<td>3.4 PID</td>
<td>27</td>
</tr>
<tr>
<td>3.4.1 The Proportional Term</td>
<td>28</td>
</tr>
<tr>
<td>3.4.2 The Derivative Term</td>
<td>29</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS (continued)

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.3</td>
<td>The Integral Term</td>
</tr>
<tr>
<td>3.5</td>
<td>OpenCV</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Color Filter</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Erode and Dilate</td>
</tr>
<tr>
<td>3.6</td>
<td>Picture Analysis</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Obstacle Distance Estimation</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Camera Coordinate Position</td>
</tr>
<tr>
<td>3.7</td>
<td>Inverse Reinforcement Learning</td>
</tr>
<tr>
<td>3.7.1</td>
<td>Markov Decision Processes</td>
</tr>
<tr>
<td>3.7.2</td>
<td>Heuristic Based Search</td>
</tr>
<tr>
<td>3.7.3</td>
<td>three categories of formalizations</td>
</tr>
<tr>
<td>3.7.3.1</td>
<td>Feature expectation matching</td>
</tr>
<tr>
<td>3.7.4</td>
<td>Inverse Optimal Control and Imitation Learning</td>
</tr>
<tr>
<td>3.7.5</td>
<td>Maxim Entropy Inverse Optimal Control</td>
</tr>
<tr>
<td>3.7.5.1</td>
<td>Deterministic Path Distribution</td>
</tr>
<tr>
<td>3.7.5.2</td>
<td>Non Deterministic Path Distribution</td>
</tr>
<tr>
<td>3.7.5.3</td>
<td>Stochastic Policies</td>
</tr>
<tr>
<td>3.7.5.4</td>
<td>Learning from Demonstrated Behavior</td>
</tr>
<tr>
<td>3.7.5.5</td>
<td>Efficient State Frequency Calculations</td>
</tr>
<tr>
<td>3.7.6</td>
<td>Predictive Inverse Optimal Control</td>
</tr>
<tr>
<td>3.7.7</td>
<td>Softened Value Iteration via Heuristic Based Search</td>
</tr>
<tr>
<td>3.7.7.1</td>
<td>Heuristic-based policy approximation</td>
</tr>
<tr>
<td>3.7.7.2</td>
<td>Greedy Selection of the Approximation Set</td>
</tr>
<tr>
<td>3.7.7.3</td>
<td>Sampling-based Improvement Algorithm</td>
</tr>
<tr>
<td>3.7.7.4</td>
<td>Stochastic Exponentiated Gradient Descent Cost Function Learning</td>
</tr>
<tr>
<td>4</td>
<td>IMPLEMENTATION</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.2</td>
<td>System specification and environment</td>
</tr>
<tr>
<td>4.3</td>
<td>Localization System</td>
</tr>
<tr>
<td>4.4</td>
<td>Collecting Data</td>
</tr>
<tr>
<td>4.5</td>
<td>Inverse Reinforcement Learning algorithm</td>
</tr>
<tr>
<td>4.6</td>
<td>Dynamic Obstacle</td>
</tr>
<tr>
<td>5</td>
<td>TESTING AND ANALYSIS</td>
</tr>
<tr>
<td>5.1</td>
<td>IRL results</td>
</tr>
<tr>
<td>5.2</td>
<td>Test with Dynamic Obstacle</td>
</tr>
<tr>
<td>6</td>
<td>CONCLUSION</td>
</tr>
<tr>
<td>APPENDICES</td>
<td>102</td>
</tr>
<tr>
<td>CHAPTER</td>
<td>PAGE</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------</td>
</tr>
<tr>
<td>Appendix A</td>
<td>103</td>
</tr>
<tr>
<td>Appendix B</td>
<td>105</td>
</tr>
<tr>
<td>CITED LITERATURE</td>
<td>107</td>
</tr>
<tr>
<td>VITA</td>
<td>110</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>FRONTAL CAMERA VALUES</td>
</tr>
<tr>
<td>II</td>
<td>VERTICAL CAMERA VALUES</td>
</tr>
<tr>
<td>III</td>
<td>OBSTACLE CENTER POSITION DIFFERENCES</td>
</tr>
<tr>
<td>IV</td>
<td>OBSTACLE CENTER POSITION DIFFERENCES 1</td>
</tr>
<tr>
<td>V</td>
<td>FLIGHT TRAJECTORIES 1</td>
</tr>
<tr>
<td>VI</td>
<td>FLIGHT TRAJECTORIES 2</td>
</tr>
<tr>
<td>VII</td>
<td>FLIGHT TRAJECTORIES 3</td>
</tr>
<tr>
<td>VIII</td>
<td>FLIGHT TRAJECTORIES 4</td>
</tr>
<tr>
<td>IX</td>
<td>FLIGHT TRAJECTORIES 5</td>
</tr>
<tr>
<td>X</td>
<td>OBSTACLE CENTER POSITION DIFFERENCES</td>
</tr>
<tr>
<td>XI</td>
<td>OBSTACLE CENTER POSITION AFTER CORRECTION 1</td>
</tr>
<tr>
<td>XII</td>
<td>OBSTACLE CENTER POSITION AFTER CORRECTION 2</td>
</tr>
<tr>
<td>XIII</td>
<td>OBSTACLE CENTER POSITION AFTER CORRECTION 3</td>
</tr>
<tr>
<td>XIV</td>
<td>OBSTACLE CENTER POSITION AFTER CORRECTION 4</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>32</td>
</tr>
<tr>
<td>10</td>
<td>33</td>
</tr>
<tr>
<td>11</td>
<td>33</td>
</tr>
<tr>
<td>12</td>
<td>34</td>
</tr>
<tr>
<td>13</td>
<td>36</td>
</tr>
<tr>
<td>14</td>
<td>38</td>
</tr>
<tr>
<td>15</td>
<td>54</td>
</tr>
<tr>
<td>16</td>
<td>56</td>
</tr>
<tr>
<td>17</td>
<td>57</td>
</tr>
<tr>
<td>18</td>
<td>58</td>
</tr>
<tr>
<td>FIGURE</td>
<td>PAGE</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>19</td>
<td>58</td>
</tr>
<tr>
<td>20</td>
<td>59</td>
</tr>
<tr>
<td>21</td>
<td>61</td>
</tr>
<tr>
<td>22</td>
<td>63</td>
</tr>
<tr>
<td>23</td>
<td>63</td>
</tr>
<tr>
<td>24</td>
<td>63</td>
</tr>
<tr>
<td>25</td>
<td>63</td>
</tr>
<tr>
<td>26</td>
<td>63</td>
</tr>
<tr>
<td>27</td>
<td>63</td>
</tr>
<tr>
<td>28</td>
<td>66</td>
</tr>
<tr>
<td>29</td>
<td>66</td>
</tr>
<tr>
<td>30</td>
<td>68</td>
</tr>
<tr>
<td>31</td>
<td>68</td>
</tr>
<tr>
<td>32</td>
<td>70</td>
</tr>
<tr>
<td>33</td>
<td>71</td>
</tr>
<tr>
<td>34</td>
<td>71</td>
</tr>
<tr>
<td>35</td>
<td>72</td>
</tr>
<tr>
<td>36</td>
<td>75</td>
</tr>
<tr>
<td>37</td>
<td>75</td>
</tr>
<tr>
<td>38</td>
<td>77</td>
</tr>
<tr>
<td>39</td>
<td>77</td>
</tr>
<tr>
<td>FIGURE</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>40</td>
<td>IRL c</td>
</tr>
<tr>
<td>41</td>
<td>IRL d</td>
</tr>
<tr>
<td>42</td>
<td>IRL e</td>
</tr>
<tr>
<td>43</td>
<td>IRL f</td>
</tr>
<tr>
<td>44</td>
<td>IRL g</td>
</tr>
<tr>
<td>45</td>
<td>IRL h</td>
</tr>
<tr>
<td>46</td>
<td>IRL i</td>
</tr>
<tr>
<td>47</td>
<td>Distribution 1</td>
</tr>
<tr>
<td>48</td>
<td>IRL j</td>
</tr>
<tr>
<td>49</td>
<td>IRL k</td>
</tr>
<tr>
<td>50</td>
<td>IRL l</td>
</tr>
<tr>
<td>51</td>
<td>Distribution 2</td>
</tr>
<tr>
<td>52</td>
<td>IRL m</td>
</tr>
<tr>
<td>53</td>
<td>IRL n</td>
</tr>
<tr>
<td>54</td>
<td>IRL o</td>
</tr>
<tr>
<td>55</td>
<td>Distribution 3</td>
</tr>
<tr>
<td>56</td>
<td>Trajectory distribution</td>
</tr>
<tr>
<td>57</td>
<td>Trajectory distribution 1</td>
</tr>
<tr>
<td>58</td>
<td>Wrong position estimation</td>
</tr>
<tr>
<td>59</td>
<td>State distribution</td>
</tr>
<tr>
<td>60</td>
<td>Obstacle error observed position a</td>
</tr>
<tr>
<td>FIGURE</td>
<td>PAGE</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>61</td>
<td>91</td>
</tr>
<tr>
<td>62</td>
<td>93</td>
</tr>
<tr>
<td>63</td>
<td>93</td>
</tr>
<tr>
<td>64</td>
<td>94</td>
</tr>
<tr>
<td>65</td>
<td>96</td>
</tr>
<tr>
<td>66</td>
<td>97</td>
</tr>
<tr>
<td>67</td>
<td>99</td>
</tr>
<tr>
<td>68</td>
<td>103</td>
</tr>
<tr>
<td>69</td>
<td>103</td>
</tr>
<tr>
<td>70</td>
<td>103</td>
</tr>
<tr>
<td>71</td>
<td>103</td>
</tr>
<tr>
<td>72</td>
<td>103</td>
</tr>
<tr>
<td>73</td>
<td>103</td>
</tr>
<tr>
<td>74</td>
<td>104</td>
</tr>
<tr>
<td>75</td>
<td>104</td>
</tr>
<tr>
<td>76</td>
<td>104</td>
</tr>
<tr>
<td>77</td>
<td>104</td>
</tr>
<tr>
<td>78</td>
<td>104</td>
</tr>
<tr>
<td>79</td>
<td>104</td>
</tr>
</tbody>
</table>
# LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRL</td>
<td>Inverse Reinforcement Learning</td>
</tr>
<tr>
<td>UAV</td>
<td>Unmanned Aerial Vehicle</td>
</tr>
<tr>
<td>3D</td>
<td>three-dimensional</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>EKF</td>
<td>Extended Kalman Filter</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>PID</td>
<td>Proportional-Integral-Derivative controller</td>
</tr>
<tr>
<td>HSV</td>
<td>Hue, Saturation and Value</td>
</tr>
<tr>
<td>RGB</td>
<td>Red Green Blue</td>
</tr>
<tr>
<td>MAV</td>
<td>Micro Aerial Vehicle</td>
</tr>
<tr>
<td>MDP</td>
<td>Markov Decision Processes</td>
</tr>
<tr>
<td>IOC</td>
<td>Inverse Optimal Control</td>
</tr>
<tr>
<td>MMSL</td>
<td>Maximum Margin Structured Learning</td>
</tr>
<tr>
<td>fps</td>
<td>Frame Per Second</td>
</tr>
<tr>
<td>DOF</td>
<td>Degree Of Freedom</td>
</tr>
<tr>
<td>RGB</td>
<td>Red Green Blue</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>RRT</td>
<td>Rapidly Exploring Random Tree</td>
</tr>
<tr>
<td>SLAM</td>
<td>Simultaneous Localization and Mapping</td>
</tr>
<tr>
<td>IOT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>EMD</td>
<td>Elementary Movement Detectors</td>
</tr>
</tbody>
</table>
SUMMARY

How to make a drone fly autonomously is a fascinating problem. This is an important field of research nowadays due to the rising usage of drones in numerous different application. Automatizing it would make them do their work autonomously and it can be very useful, for instance for security control or delivery, but this are just two example of their use.

The aim of this thesis is to analyze this problem starting from a drone that can be piloted by a human giving it destination coordinates and find a method to make it fly autonomously. In this thesis this problem has been approached applying an Inverse Reinforcement Learning algorithm. Thank to this, after a training done with expert trajectories the drone started to fly alone given a starting position and a goal. During the development of the project dynamic obstacle have been used is order to make the environment closer to the reality.

In the first chapters there is a description of some works previosly done on similar topic and a description of everything used during the research. Then there is a chapter where the implementation procedure is described. Here a step by step resolution of the problem is given to the reader. Then there is chapter which contains the results of the tests. They are described and explained.

In conclusion the drone learned a cost function from observed trajectory and is able to compute the best path between two point in the map. In the case it sees an obstacle which is not in the map it adds this and change its trajectory in the case there is a risk of collision.
CHAPTER 1

INTRODUCTION

Drone have been used a lot during last years to accomplish a lot of tasks. For instance in the military field for inspection, but there are numerous other way they can be employed. For instance they can be used for deliveries(1), rescue or exploration. The only drawback was that every one of them needed a human driving it remotely since they have not the ability to operate autonomously in the real world yet. So it is very important to find a way to make them fly in autonomy in the real environment where many things can not be predicted.

The skill of navigation in the environment it is used in is an important prerequisite of an autonomous drone. Usually the normal approach used in outdoor environment navigation system is based on the GPS and the paths are planned in advance. The main drawback of the GPS is that it needs to receive data from at least four different satellites. For this reason it is not possible to use the GPS in every environment. For instance in several outdoor areas, such as where there are many obstacles, or inside buildings the GPS signal is not clear enough, therefore it is not convenient for navigation. To avoid this problem other navigation methods have to be used to navigate through these environments. These can be based on different techniques or sensors. Some of them are computer vision techniques or inertial navigation systems which uses the value of accelerometer and gyroscope. In Micro Aerial Vehicle (MAVs) there are only this kind of sensor since they are cheap and light.
The goal of this thesis is to apply an inverse reinforcement learning algorithm to a quadcopter in order to make it fly autonomously. One further problem approached is how to deal with dynamic obstacle in the environment, using the same reward function learned. In this case the problems faced are related to the capacity of being reactive and find solution in a short period of time. This is very hard to solve with the time complexity of the algorithm used.
CHAPTER 2

PREVIOUS WORK

This chapter sums up most of the research work and project done in the field of autonomous navigation for UAV, and in specific with the Parrot AR Drone. This quadcopter is affordable and for this reason it has been used for some research projects. It is easy to test since there are some applications for smart phones that allow the users to control it with two digital joysticks by looking at the video of the camera as a feedback on the display. These applications are sufficient for merely flying the drone, and the power of the quadcopter is not completely used. This is due to, for instance, humans limits, since we are not as fast as computers in making corrections on trajectories or controlling other parameters. In addition since the API is available, it is possible to create higher level of abstraction for the drone and build more complex applications to flight the drone autonomously.

A lot of different navigation system have already been considered in this sector. Here some of these research are listed and briefly described.

2.1 Corridor and Stair Following

One kind of autonomous navigation consists in the ability to follow a corridor avoiding crashes. Basically the drone try to keep the middle of it staying at a similar distance from the two walls on the sides.
Generally this problem is approached creating a 3D model of the environment in advance with respect to planning and control. Anyway another method which is based on the perspective cues of a single image (2) is used to navigate in the indoor environments. The first step done by this approach is to define the type of environment, knowing that it has to be an indoor one. After that the MAV can move around since it has built a map before. This is achieved with some vision algorithms based on perspective cues and this is used to estimate its direction. The discovery of the environment is implemented with a confidence classifier which is able to recognize stairs and corridors. Among the possible one, it is selected the environment which has the highest value of confidence. This algorithm uses some powerful computer vision functions. Some of these are the probabilistic Hough transform and the Canny edge detector in order to acquire the line segments. The position of the vanishing point is a problem solved by the corridor algorithm which tries to evaluate where it is in the image. While on the other hand the position in the middle of the stairs is evaluated by the stair algorithm by observing the horizontal line segments of the same.

These vision-based methods allow MAVs to traverse corridors, stairs and corners on the basis of a small, light-weight camera, which makes them suitable for MAV with payload and power restrictions.

2.2 Road-Following

Differently from the previous work, in this case the environment is outdoor, and the task achieved is to follow roads lines. This project has been developed with small autonomous aircraft (3).
The aircraft is able to visualize and detect the roads and lanes using cameras with a multiple vision algorithm. Instead of the HSV filter the author of this project uses the Bayesian Pixel Classifier. Obliviously noises are removed in order to find only the linked region of the image. This allow it to detect, in the image, the position of the roads and their respective lanes. Then using a Hough transformation the lanes seen are selected among all the possible one. Then the algorithm is able to define the position and the orientation of the central lane.

2.3 Elementary Motion Detectors

Edge detection and Elementary Movement Detectors (to find motion information) are combined together in an approach presented during the IMAV competition(4). Motion and temporal effects are detected by EMD. This characteristic makes EMD be an important development on UAV since it detects the movement done by the drone. Discrepancies in color and contrast are some of the characteristics on which the EMD is implemented, and generates motion information. Border of relevant objects produce higher responses due to the fact that EMD application improves the segmentation of the image. Edge enhancement is caused by the rotation of the drone, while due to translation the closer is an obstacle to the drone the larger is its motion detected. A trustworthy piece of the information is provided by the motion of the drone. Since EMD depends on contrast discrepancies, even if it is useful, it is not providing a total picture of the environment. The segmentation achieved with this method is not enough to detect all the information needed. In order to solve this issue and notice the obstacle in the environment an edge detection algorithm is combined with EMD.
EMDs filter out noise caused by the edge detection algorithm due to the contrast differences in the image. From this cleaned image objects were detected by using a standard Hough transform.

2.4 Obstacle avoidance

The aim of this kind of autonomous flight is to avoid collision with obstacles. A recent approach to this problem uses a motion detection based algorithm to avoid obstacles in a corridor(5). The approach is based on monocular stereo vision that computes the disparity map between two consecutive images. The disparity map shows the difference in motion in the environment. Obstacles that are close by have a greater motion than objects that are further away. Due to the different distances of objects the platform is able to detect and avoid them.

2.5 RRT Paths Planner

This algorithm generates a tree whose root is the starting position. From this it grows randomly generating new edges in the free space until it reaches the goal(6). The input of this algorithm are positions of start, goal and obstacles. The initial state of the tree is made by the starting position as a root and single vertex. Every vertex generated is associated with its position in the space. At each step the algorithm randomly select a new free location in the environment, of course that is not overlapping with any obstacle. Then the tree grows from the vertex that is nearest to the random position to that position by a certain increment. This increment defines a new edge between this two point and the weight given to this edge is equal to the Euclidian distance. This process is reiterated until one of the new vertex is close to the goal. If this happens the last edge is added to link this node with the goal position. When the
tree is built in order to find the shortest way that links start and end node Dijkstra’s shortest path algorithm is applied.

2.6 Scale-Aware Navigation with a Monocular Camera

Another research develops a scale aware solution for the visual navigation in unknown environments.(7) This approach relies solely on a monocular camera as the main sensor, and therefore does not need external tracking aids such as GPS or visual markers. This needs three main components:

- a monocular SLAM system
- an EKF to correct the prediction
- a PID controller

In order to develop this project it is necessary to define accurately the model of the flight dynamics of the quadcopter. In addition it has to be implemented a method to evaluate the proportion of a monocular SLAM system. They use an external motion capture to evaluate pose estimation accuracy and flight accuracy.

2.7 Simultaneous localization and mapping

In this approach a real-time Simultaneous Localization and Mapping (SLAM) algorithm has been developed with the down-looking camera.(8) Thank to this, the quadcopter manage to create its own map representing the environment and find its position in the same. Basically it aggregates information collected with the various sensors in order to build the environment. The information regarding its position is useful for the other functions of the drone, for instance
all the functions that have to interact with the environment. This is implemented using the information received by the sensor of the drone, such as camera or altimeter.
CHAPTER 3

BACKGROUND

In this chapter a summary and a description of the sub-fields approached during the development of the project is given to the reader. Since the topic related are different the background covers more areas.

3.1 Quadcopter

A quadcopter is a flying object moved by four rotors. It can change its position in a 3D space just by using the four moment generated by the four engine. Thus this is achievable changing each engine speeds in order to generate different balance of the forces. Every rotor gives a contribution to the upwards thrust, but the forward left and the backward right rotor are rotating clockwise, while the other two are rotating counter-clockwise in order to delete and balance the torques. In this way (at least theoretically) running all engines at equal speed, with an acceleration equal to the gravity, allows the quadcopter to stay in its position without moving. For example to increase vertical acceleration and move to an upper altitude all the four rotor have to increase equally their speed, while to change the yaw, the angle around the z axis, without changing the altitude the quadcopter has to increase the speed of two opposite rotors and decrease the speed of the two complementary ones. Horizontal movement can be achieved by increasing the speed of one engine, while decreasing the speed of the opposing one. In this way there is a change of the roll or pitch angle, and thereby inducing horizontal acceleration.
3.1.1 Parrot AR Drone 2.0

The Parrot AR Drone 2.0(9) is a UAV created for home entertainment. It is widely used in the academic field mainly due to its low weight and the low price. The AR Drone has a lot of functions available and it is possible to use it with a lot of platforms, for instance smart phones, where the user can maneuver it using the feedback of the camera and two digital joystick for the six degree freedom movement. It comes with some useful components:

- Frontal camera
- Bottom camera
- Gyroscope
• accelerometer

• magnetometer

• ultrasound sensor for altitude measurement (emission frequency 40 Khz range 6 meter)

• Pressure sensor

In order to develop applications based on this quadcopter, some APIs are available. However there is no access to the control software embedded on the drone, which is proprietary and neither open-source nor documented. Drone state values are given by four components always communicated when the battery is connected to the drone:

• AT commands are strings used to control its actions. They are sent on the command channel (UDP port 5556).

• Navigation data, on navigation channel (UDP port 5554)

• video stream, on video channel (UDP port 5555)

• control port (TCP port 5559)

3.1.1.1 Components

3.1.1.1 Cameras

The AR Drone has two onboard cameras, one pointing forward and one pointing downward.

They have the ability to recognize automatically some specific symbol such as the roundel (Figure 17), the cocarde (Figure 2) or the shell tag on the hull (Figure 3). The values sent in the navdata packet are distance of the object, identified position with respect to the left high
corner of the camera, thus $x$ and $y$. Other values given are the size of the edges along the $x$ and $y$ axis. With all these information it is possible to calculate the position of the object seen using some simple formulas, as shown later in this chapter.

![Figure 2. Parrot AR Drone 2.0 cocarde](image1)

![Figure 3. Parrot AR Drone 2.0 outdoor](image2)

3.1.1.1.1 Frontal Camera

The frontal camera has a diagonal lens with a wide-angle of about ninety degree. A CMOS sensor is used for this scope. It is placed in front of the drone and it looks forward. Its video
frequency is 30 fps and the maximum resolution of the same is 1280 x 720 pixel, but it is possible
to change this configuration to a lower resolution of 640 x 360.

### TABLE I

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>focal length</td>
<td>4 mm</td>
</tr>
<tr>
<td>resolution</td>
<td>1280x720</td>
</tr>
<tr>
<td>angle</td>
<td>92 degree</td>
</tr>
</tbody>
</table>

3.1.1.1.1.2 **vertical camera**

The vertical camera is a sixty-four degrees wide-angle diagonal lens camera done with a
CMOS sensor. It is placed on the bottom of the structure and it points to the floor, or generally
under the drone. Its video frequency is 60 fps and the maximum resolution of the same is 320
x 240 pixel.
This camera makes automatic ground speed measures for automatic hovering and trimming. If the floor under the drone does not present any differences between different location this functionality works poorly, while if there are a lot of references this estimation is much better.

TABLE II

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>resolution</td>
<td>640x360</td>
</tr>
<tr>
<td>angle</td>
<td>64 degree</td>
</tr>
</tbody>
</table>

3.1.1.1.2 gyroscope

This quadcopter has a 2-axis gyroscope (measuring pitch and roll angle) and a one-axis yaw precision gyroscope. The measured roll and pitch angles are very accurate, since they have a deviation of only up to 0.5 degree, and they are not subject to drift over time. On the other hand, the yaw measurements drift significantly over time, it can be up to 60 degrees per minute in the worst observed cases.
3.1.1.1.3 **accelerometer**

It has a 3 axis accelerometer which gives the value of the acceleration along every axis.

3.1.1.1.4 **magnetometer**

There is a three axis magnetometer for absolute control mode. In this mode the direction of the drone when receiving the command to move forward is the same direction faced by the smart phone. Therefore the coordinate system are relative to the phone position.

3.1.1.1.5 **ultrasound sensor**

The ultrasound sensor is used as an altimeter. It has a maximal range of 6 meters and an emission frequency of 40 kHz.

3.1.1.1.6 **Pressure Sensor**

A pressure sensor allows altitude measurements at any height, when the altimeter is not working over 6 meters.

3.1.1.1.7 **other technical detail**

The Parrot AR.Drone has dimensions of 51,5 cm × 51,5 cm with the indoor hull, and 45,2 cm × 45,2 cm with the outdoor hull. A removable Styrofoam hull (Figure 4) protects the drone and particularly the rotors during indoor flights, allowing the drone to avoid damage caused by minor crashes such as flying into various types of room furniture. Another hull is available for flight outdoors(Figure 5). This allows for better maneuverability and higher speeds. Its weight is 436 g with the indoor hull and 400 g with the outdoor hull.
The AR Drone has four rotors with a 20 cm diameter propeller, fastened to a robust carbon-fiber skeleton cross which provide stability. They are four brush-less 15 watt 28500 rpm in-runner motor.

It is equipped with a lithium polymer battery with three cells. There are different configuration of mA/hour.

The embedded computer system has an 1 Ghz ARM Cortex A8 with 128 MB SDRAM, 128 MB of NAND flash memory, and a wifi module. It runs Linux OS.

The drone IP in its network is 192.168.1.1, and the DHCP Range vary from 192.168.1.2 to 192.168.1.5.

Figure 4. Parrot arDrone 2.0 indoor hull     Figure 5. Parrot arDrone 2.0 outdoor hull
3.1.1.2 Drone communication channel

3.1.1.2.1 AT Commands

This channel is used to command the drone through a set of commands that set some drone parameters:

- **ROLL**: This value range is between minus 1 and 1 and it is proportional to the maximum angle achievable by the drone, which is 12 degrees. It is the rotation around the y axis. Y in the drone coordinate system is where the horizontal camera points, thus the forward direction.

- **PITCH**: It is controlled in the same way roll is. The value is in a range between minus 1 and 1 which is proportional to the 12 degrees maximum angle reachable by the rotation around the x axis, so the one perpendicular to the y axis that lay on the same plane of the drone.

- **GAZ**: This value is proportional to the vertical speed and like the other vary in the range of -1 to 1. The maximum achievable vertical speed is 0.7 m/s.

- **ROTATION**: It has the same value of the others and it refers to the maximum angular speed achievable by the drone, which is 100 degrees. This is the yaw rotational speed.

- **HOVER MODE**: There is one bit to switch between hover mode (the drone tries to keep its position, ignoring any other control commands) and manual control mode.

- **ERROR**: There is one bit indicating whether the drone is supposed to enter or exit an error state. If so it immediately switches off all engines.
• START-END: There is one bit indicating whether the drone is supposed to take off or land.

3.1.1.2.2 Navigation Data

This service provided by the drone gives a lot of information on its status, for instance altitude, speed, measures from the sensors of the quadcopter. If it is set to demo mode it sends data 15 times per second, while if demo is disabled it does it 200 times per second.

The most important parameters and sensor values are underlined here:

• drone orientation: roll, pitch and yaw angles. Roll and pitch values are drift-free and very accurate, while the measured yaw-angle is subject to significant drift over time.

• horizontal velocity: in order to enable the drone to keep its position in spite of wind, an optical-flow based motion estimation algorithm utilizing the full 60 fps from the floor camera is performed on board, estimating the drone’s horizontal speed. The exact way these values are determined however is not documented. Experiments have shown that the accuracy of these values strongly depends on whether the ground below the drone is textured or not: when flying above a textured surface these values are extremely accurate, while flying above a poorly textured surface, the quality of these speed estimates can be very poor, deviating from the true value by up to 1 m/s above completely untextured surfaces.

• drone height in millimeter: this value is based solely on the ultrasound altimeter measurements. As long as the drone is flying over a flat, reflecting surface, this value is quite
accurate, with (at a height of 1 m) a mean error of only 8.5 cm. As this sensor measures relative height, when flying over uneven surfaces or close to walls strong fluctuations will occur. This often induces sudden and undesired vertical acceleration, as the drone tries to keep its relative height as supposed to its absolute height. This value is measured only every 40 ms. When the drone is beyond the six meters of altitude instead of the altimeter the pressure sensor is used to give evaluation about height.

- battery state: an integer between 100 and 0, obviously 100 when it is totally charged and 0 when the battery is out of energy.

- control state: a 32-bit bit field, indicating the drone’s current internal status. This might for example be LANDED, HOVERING, ERROR, TAKEOF.

- internal timestamp: at which the respective data was sent, in microseconds. This is not necessarily the time at which the sensor values were taken, experiments have shown that within the same package, some parameters are up to 60 ms older than others.

- detected tag: the number of detected tags or oriented roundel. It gives additional information such as an array containing all the detected items and the kind of item detected. The different object detected are classified in an enumeration. For each of this it sends the coordinates of the left high corner of the tag in the camera point of view and the dimension of the same. In addition to this there are value for distance from the object and the orientation of the drone with respect to the object expressed in degree.
3.1.1.2.3 Video Stream

A video stream sends the video from the AR Drone to the client connected to its wifi. It can be one of four different channels. These are:

- **Horizontal camera**
- **Vertical camera**
- **Horizontal camera wide screen with a little image of the vertical camera in the top left corner**
- **Vertical camera wide screen with a little image of the horizontal camera in the top left corner**

The maximum frame rate value for the vertical camera is 60 fps, while for the horizontal camera it is 30 fps.
3.1.1.2.4 Control Port

It is used to send configuration data, for instance switching among the four video channels described above, or acknowledge signals. It uses a TCP connection in order to have ordered data. Some of the configurable options are:

- navdata_demo: this configuration sets the amount of information the drone sends. If it is set to true it sends 15 packets per second, if it is false the number of packets sent per second are 200.
- navdata_options: there are a lot of different navdata packets defined in the library, with this option it is possible to select the ones of interests.
- it is possible to set all the values of gyroscopes or accelerometers offset or gain.
- it is possible to set maximum values for altitude or yaw speed.

3.2 Node

There are several framework and libraries available to control and program the AR.Drone. Some of them are for instance Robot operating system (ROS), c++ or Python. In this project Node.js is used because because for the purpose of the same it is easy to develop, it is portable in different operating system and it is light weight. Node is a great tool for real-time updating, handling lots of users, and in some cases using a lot of data.

Node.js is a platform which run in different Operating System(OS X, Microsoft Windows, Linux, FreeBSD, and IBM i) and is open source. It is a runtime environment for server-side and networking applications. It is commonly used for real-time web applications. Google V8
run-time JavaScript engine is used to execute the code, and most of the modules are written in JavaScript. One of the libraries that is part of it makes simple applications act like a Web server. It is not necessary for them to use other applications such as Apache to act like this. The library libuv allows Node.js to handle asynchronous events. Network and file system functionality are abstracted with this library. While it waits for requests it runs in a loop. Any request can submit other requests, such as sending signals or commands to a robot or reading files from the disk. That loop, known as the event loop, is the runtime part and with this platform it is easy to build a network or develop event-driven applications.

Figure 7. Node structure
The event-driven architecture of node.js has a non-blocking I/O API in order to enhance throughput and scalability of applications. Non-blocking servers usually use only one thread to service all requests, although on Node.js 0.10+ the execution in multi-thread is available, but it uses multiple threads for network events and files. A single listener thread which delegates I/O work to a pool of threads can be important to have an highly available application. I/O requests are asynchronous tasks so that they return before it is finished their job, and some work is executed in background. When it finishes it calls for instance a callback or a signal. With these characteristics it can handle some thousands of concurrent connections, and it has not to concern about the cost of context-switching between threads. Therefore with this implementation is possible to build highly concurrent applications, and to manage everything it is necessary that any function which performs I/O tasks have to use a callback.

Whenever the single listening thread receives a request it executes the event related to the same. In this moment there are two possible job options available:

- quick, non-CPU intensive work  
- long-running I/O bound operations

In the first case the listener thread blocks for the duration of the request the execution of the main loop, but since the request is very quick delays are negligible. On the other hand in the second case node.js delegates the I/O works to a thread taken among the ones in the pool of native C++ threads thank to V8 and libuv. When the listening thread calls the worker thread, it passes to it a callback so that in the moment it finishes its work, the same is executed signaling the end of it. After it delegate the work immediately comes back to listen for the
next connection. When the main thread receives the callback, it returns the result to the client. Basically the Event loop waits for and dispatches events or messages in a program. External events are converted to callback invocations. Node.js can switch among requests when I/O calls is executed. When an I/O call happens, after the callback is saved it returns the control to the runtime environment. Either if the multi thread functionality is not directly available to the user node.js in the back-end side is effectively a multi-threaded platform.

3.2.1 Nodecopter Modules

Nodecopter modules(10) are a collection of client programs already developed which have implemented some functionalities related to the AR Drone 2. The node-ar-drone module(11) is an implementation of the networking protocol and the SDK functionalities. Thanks to this the drone can be controlled using the received video and sensor data and write autonomous program(12) on top of it. The client API returns a client object that can be used to maneuver the drone directly. It also provides a configuration layer, where some of the AR.Drone initial configuration values can be set up. Some of them are providing more navigational data, setting animation led values, toggling on/off emergency, emitting event status like hovering, flying, landing, battery change and altitude change. It is written completely in JavaScript, but since it is an open project there are some bugs and functionality to be fixed or added.

3.2.1.1 node-ar-drone

This module basically translate all the SDK APIs(13) for node.js. Therefore it is possible to send to the drone numerous different commands and settings. For instance after a client is created it is possible to get the video, or gives command such as go up/down, left/right,
forward/backward specifying the speed to impress in that direction. Then the user has to manually stop the previous command, if not the drone will keep the same speed in the specified direction until it meets an obstacle.

### 3.2.1.2 ardrone-autonomy

This module develops some interesting and useful functionalities. With some PID it controls the speeds along the three axis and the rotational speed of the yaw. Therefore it allows the user to specify instead of the speed rate at which it should move the distance in meter to fly in the specified direction. With these functions it is possible to develop a lot of different projects since it is easier to control. In addition to this this module created a first version of a Kalman filter to correct the position of the drone above its starting position using the bottom camera.

### 3.3 EKF

The extension for nonlinear systems of the Kalman filter is called Extended Kalman Filter. Current mean and covariance are estimated and used to linearize. Since the problems in the real world usually are not linear, the possibility to apply this filter to non linear problems make it usable in more situations. If the transition model among states is defined correctly the EKF is considered the standard in nonlinear state estimation. It is used to approach problems such as navigation systems and GPS.

The state observation and transition models do not have to be necessary linear functions, it is enough for them to be differentiable.

$$x_k = f(x_{k-1}, u_{k-1}) + w_{k-1}$$
\[ z_k = h(x_k) + v_k \]

Where \( v_k \) is the observation noise and \( w_k \) is the process noise. This two are zero mean multivariate Gaussian noises with covariance \( R_k \) and \( Q_k \) respectively. \( u_k \) is the control vector.

With the function \( f \) it is possible to estimate the current state given as parameter the estimation of the previous state and the previous control. Analogously with the function \( h \) it is possible to compute a prediction on the measurement given the state. In order to apply the covariance, \( f \) and \( h \) have to be partially derived with the Jacobian. This is done every time step with the current predicted states. With this matrices in the Kalman Filter equations it is possible to linearize the function around the current estimate. The difficulty now lies in the fact that when we apply a nonlinear transformation to a Gaussian random variable, the resulting random variable is no longer Gaussian: in order to still make the above framework applicable, \( h \) and \( f \) are approximated by a first-order Taylor approximation, which however leads to the result no longer being optimal. Let

\[
F_{k-1} = \frac{\partial f}{\partial x} |_{\hat{x}_{k-1}|_{k-1}, u_k} \\
H_k = \frac{\partial h}{\partial x} |_{\hat{x}_{k}|_{k-1}}
\]

Update and prediction can then be approximated as follows:

**PREDICTION:**

\[
\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k)
\]
\[ P_{k|k-1} = F_{k-1}P_{k-1|k-1}F_{k-1}^T + Q \]

where \( \hat{x} \) and \( P \) are respectively the predicted state estimate and the covariance estimate.

**UPDATE:**

\[ y_k = z_k - h(\hat{x}_{k|k-1}) \]

\[ S_k = H_kP_{k|k-1}H_k^T + R \]

\[ K_k = P_{k|k-1}H_k^T S_k^{-1} \]

\[ \hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k y_k \]

\[ P_{k|k} = (I - K_k H_k)P_{k|k-1} \]

where \( y \) is the measurement residual, \( S \) is the residual covariance.

### 3.4 PID

Control theory deals with the problem of controlling the behavior of a dynamic system. The general goal is to calculate system input values \( u(t) \), such that the system reaches and holds a desired state. In other words, the measured error \( e(t) \) between a given setpoint \( w(t) \) and the measured output of the system \( y(t) \) has to be minimized over time. In particular, the goal is to quickly reach the desired setpoint and hold it without oscillating around it, counteracting any random disturbances introduced into the system by the environment. Therefore a PID is a controller based on a control loop mechanism. It is divided into three separate control mechanisms:
• the proportional part depends on the current error $e(t)$

• the integral part depends on the accumulated past error $\int_0^t e(x)dx$

• the derivative part depends on the predicted future error, based on the derivative of the error with respect to time $\dot{e}(t)$

If integral and derivative of the error cannot be measured directly, they are approximated by numeric integration and differentiation:

$$\int_0^t e(x)dx \approx \sum_{x=1}^{t} e(x)$$

$$\dot{e}(t) \approx \frac{1}{\delta t}(e(t) - e(t - \delta t))$$

The PID controller now calculates the system input, control signal, values according to their sum:

$$u(t) = k_p e(t) + k_i \int_0^t e(x)dx + k_d \dot{e}(t)$$

where $K_p$, $K_i$ and $K_d$ are tunable parameters which define the weight of the three different response.

3.4.1 The Proportional Term

The proportional part is always required, and is the part responsible for reducing the error: the bigger the error, the stronger the control signal. In real-world systems however, a purely proportional controller causes severe overshoot, leading to strong oscillations.
3.4.2 The Derivative Term

The derivative part has the effect of damping occurring oscillations: the higher the rate of change of the error, the more this term contributes towards slowing down this rate of change, reducing overshoot and oscillations.

3.4.3 The Integral Term

The integral part is responsible for eliminating steady-state errors: for a biased system requiring a constant control input to hold a state, a pure PD-controller will settle above or below the set point. Depending on accumulated past error, the integral term compensates for this bias. It needs to be treated with caution as it may increases convergence time and cause strong oscillations.

3.5 OpenCV

OpenCV is an open source computer vision software and machine learning software library. This library has been ported in Node.js environment so it is suitable for the scope of this work. Using this library it is possible to process real time images. There are algorithm in this library for objects identification, movements tracking, faces detection and other purposes. Over the years various computer vision techniques have been developed to detect objects in images. One of the fundamental problems is finding the most applicable algorithm for the defined problem. In order to follow a line in flight, fast feedback from the camera is required in order for the system to adjust its trajectory. For this reason, a real-time algorithm is favorable. Furthermore, the algorithm should be able to handle movement noise caused by the motions of the flying platform.
3.5.1 Color Filter

RGB is an additive color model. The three main color Red, Green and Blue are added in different combination to reproduce other colors.

![RGB](image)

Figure 8. RGB

HSV is one of the way to represent point of the RGB color model in cylindrical coordinate. As it is shown in the picture (Figure 9) the angle of rotation around the vertical axis represents the H (hue). The radius from the same axis is S (saturation). The height represent the V (value) which is the perceived luminance in relation to the saturation. The Hue stands for the color. Basically it is the combination of two primary color. It differs among all different shades of colors. For instance yellow, red, green. Usually it is represented on a scale from 0 to 360, but opencv normalizes it in the range written below. Saturation is the colorfulness of an image, characterized by the amount of gray in the color of the same. Value represents the brightness and the intensity of the color. The three parameters have the following ranges:
• Hue: [0:179]

• Saturation: [0:255]

• Value: [0:255]

Using one of these different colors representation it is possible to define a lower and an upper threshold in order to filter an image. The image is usually converted to HSV and every pixel is checked whether it is within this range, if so the pixel in the result image is set to 1 otherwise to 0. This result image is a binary image, where the colors in range of the color are set to white, while the others are black. Color filtering reduces the amount of noise caused by other colors in the image. This makes very easy to extract a color from an image.
3.5.2 Erode and Dilate

This is the picture obtained after the color filter has been applied:
The erode command makes the white area being smaller because it erodes its shape on the contour.

Figure 10. HSV filter original figure

Figure 11. Figure after erode command
On the other hand the dilate command makes the selected area be bigger by increasing its shape size.

Figure 12. Figure after dilate command

These two operation are very important to reduce some issue like disturbance in the image.

3.6 Picture Analysis

This section describes how to find useful information from pictures or images, knowing some parameters.
3.6.1 Obstacle Distance Estimation

The camera calibration matrix contains information useful for the computation of the distance of objects. 

\[
\begin{bmatrix}
  f_x & 0 & \sigma_x \\
  0 & f_y & \sigma_y \\
  0 & 0 & 1
\end{bmatrix}
\]

where \( f_x = f \cdot m_x \) and \( f_y = f \cdot m_y \). \( m_x \) and \( m_y \) are the values of pixel per millimeter on the sensor. With this value it is possible to convert the size of the image in pixel given by opencv to its real size on the sensor. Since this two formulation are identical for the purpose of this problem it is possible to make an average between them. \( f_{xy} = \frac{(f_x + f_y)}{2} \). The result of the inverse formula is the one which gives the parameter for how many pixel fits in one millimeter of the camera, \( m = \frac{f_{xy}}{f} \). The only missing parameter is the size of the image on the screen since it is supposed that the real size of the object is known. These value is easily found in opencv. Then the last step is to use all the values found in the previous passages in the formula:

\[
distance = \frac{S \cdot f}{s}
\]

where \( S \) is the size of the real object, \( s \) is the size of the object on the sensor and \( f \) is the focal length. These three value have to be expressed in the same unity of measure, usually in millimeter.
3.6.2 Camera Coordinate Position

This is the operation of mapping between the 2D screen of the camera and the 3D world taken by the camera. The model used for this aim is the pinhole camera model.

\[(X, Y, Z, 1)^T \rightarrow (x, y, z)^T\]

Figure 13. Pinhole

Where the coordinates \((x, y)\) are the 2D coordinates in the screen, while \((X, Y, Z)\) are the 3D coordinates in the environment and \(f\) is the focal length.

Since the coordinate of the image on the screen are not centered in most of today camera (like the AR Drone 2 cameras Figure 6), differently from what is shown in Figure 13, it is
necessary to translate the origin. To do so it is added an offset called principal-point offset. This method is used when the pixel coordinate system origin is in the top-left corner of the image. In this case it is necessary a conversion of coordinate. The principal-point position can be assimilated into the projection matrix just with the addition of two elements in the matrix. This is identified by \((\sigma_x, \sigma_y)^T\).

\[
\begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix} =
\begin{pmatrix}
  fX \\
  fY \\
  Z
\end{pmatrix} =
\begin{bmatrix}
  f & 0 & \sigma_x & 0 \\
  0 & f & \sigma_y & 0 \\
  0 & 0 & 1 & 0
\end{bmatrix}
\begin{pmatrix}
  X \\
  Y \\
  Z \\
  1
\end{pmatrix}
\]

where the first three rows and the first three columns of the projection matrix form a matrix \(K\) that is known as the camera calibration matrix.

\[
\begin{pmatrix}
  X \\
  Y \\
  Z
\end{pmatrix} \rightarrow \begin{pmatrix}
  fX/Z + p_x \\
  fY/Z + p_y
\end{pmatrix}
\]

where \((p_x, p_y)^T\) represents the offset’s coordinates.
One of the assumptions done is that the pixels on the image sensor are square, so the aspect ratio is 1 : 1 and pixels are not skewed, otherwise other terms would be added to the matrix.

\[
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix} =
\begin{bmatrix}
  fX \\
  fY \\
  Z
\end{bmatrix} =
\begin{bmatrix}
  f & \tau & \sigma_x & 0 \\
  0 & \eta f & \sigma_y & 0 \\
  0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
  X \\
  Y \\
  Z \\
  1
\end{bmatrix}
\]

where the parameters \( \tau \) and \( \sigma \) model the pixel aspect ratio and skew of the pixels, respectively.

### 3.7 Inverse Reinforcement Learning

IRL consists of recovering the reward function \( R \) starting from a policy \( \pi^* \) and knowing the model dynamics. It learns to imitate from observations of examples of human behavior. Some
expert gives a demonstration and from their behavior the Reward function, $R$, is built. The input of this model are, as stated above, teacher demonstrations, transition model, state and action space.

The main aim is to try to recover $R$ from the information stated above and then use it to find a good policy (apprenticeship learning). This is contrast with methods that try to directly learn the teacher’s policy using supervised learning (behavioral cloning). This last case is an estimation of the policy obtained from some training examples. Many supervised learning techniques, such as neural networks and decision trees, can be employed.

In this thesis I am not interested in behavioral cloning, so I will focus on the apprenticeship learning.

### 3.7.1 Markov Decision Processes

The definition of a MDP (29) is based on a tuple $(S, A, \tau, R)$. Where $S$ is the state set, $A$ is the action set, $\tau : S \times A \rightarrow \Delta_s$ is the distribution of the next state and $R$ is the reward function. The optimal policy $\pi$ is the sequence of state action which maximize the global reward.

$$E\left[\sum_{t=1}^{T} R(s_t, a_t) | \pi, s_1\right]$$

Given the policy the agent decides what to do by consulting its current percepts, which tells it the current state $s$, and then executing the action suggested by the policy itself. A policy represents the agent function. MDPs are used to compute decision in a sequential environment.
3.7.2 **Heuristic Based Search**

With a bigger knowledge the state space to be analyzed shrinks significantly. Using for instance an $A^*$ algorithm it is possible to explore only some path, since it chooses according to the accumulated cost and the heuristic value. In this case the accumulated cost is the optimal reward to reach the actual state, while the heuristic is an upper bound on the future reward obtained taking an action 'a' from the actual state.

$$f(s_t, a_t) = r^*(s_t) + Q^+(s_t, a_t)$$

If the heuristic is admissible the algorithm is guaranteed to find the optimal solution. An admissible heuristic has to not overestimate the cost to reach the goal. In addition it has to be consistent, therefore the estimated cost to reach the goal from a successor of a node has to be greater or equal then the estimated cost to reach the goal from that node, $h(x_t) \leq cost(x_t, x_{t+1}) + h(x_{t+1})$.

3.7.3 **three categories of formalizations**

There are three different mathematical formulations for IRL:

- Max margin
- Feature expectation matching
- Interpret reward function as parameterization of a policy class

All these three methods are used in order to find a reward function $R^* = w^T f(x, y)$ which describe in the most optimal way the behavior of the expert. Doing this needs some assumption
such as: the expert demonstrations have to be optimal and all the policies are enumerable. The Reward function has different features that characterize the different policies. Obviously the more feature there are the more sample are needed.

3.7.3.1 Feature expectation matching

In this method(20) we assume the reward function to be $R(s) = w^T \phi(s)$, where $w$ are the weight of the feature and $\phi$ are the feature.

$$E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi \right] = w^T E\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) | \pi \right] = w^T \mu(\pi)$$

Therefore the expected cumulative discounted sum of the reward function is equal to the weighted expected cumulative discounted sum of feature values, $\mu(\pi)$. Since the optimal policy has an expected sum of the rewards that has to be greater or equal to the expected sum of the reward function of every other policy, it is possible to derive

$$w^*^T \mu(\pi^*) \geq w^*^T \mu(\pi)$$

The problem in this case is to find $w^*$. The feature expectation can be estimated from the sample trajectories. The number of demonstration doesn’t depend on the complexity of the policy or the size of the space, but from the number of features.
In order to guarantee that a policy \( \pi \) behaves like the expert \( \pi^* \) it is sufficient to have a match between feature expectations:

\[
\| \mu(\pi) - \mu(\pi^*) \|_1 \leq \epsilon
\]

therefore

\[
\| w^T \mu(\pi) - w^T \mu(\pi^*) \| \leq \epsilon
\]

### 3.7.4 Inverse Optimal Control and Imitation Learning

One way this problem is approached is attempting to create a reward function starting from a known sequence of actions. They define vectors of reward factors \( f_{s,a} \), and assume the reward function to be linear with respect to this feature vectors.

\[
R(s, a) = \Theta^T f_{s,a}.
\]

where \( \Theta \) are the reward weights. IOC formulation(14) was defined as the process to find the reward weights \( \Theta \) which describe the optimal demonstrated behavior. This problem is not well-posed since different combinations of weights can lead to more than one optimal result, for instance degeneracy, all zeros, is one solution.

A successive approach tries to find the reward weights having as goal the same result for demonstrated trajectories and a planner based on these weights(15). This formulation reduces
to find a match between the planner result and the expected feature count of the demonstrated trajectories, \( f_\zeta = \sum_{s,a \in \zeta} f_{s,a} \).

\[ \sum_\zeta P_{plan}(\zeta|\theta)f_\zeta = f_\tilde{\zeta} \]

where \( \zeta \) is the path and \( f_{s,a} \) is the vector of reward features. The solution of the optimal MDP for trajectories' distribution is used to define a series of deterministic controls. If a behavior is shown to be sub-optimal, because of unobserved reward function or agent’s imperfection, more policies are mixed in order to match the feature counts. A lot of mixture will match the feature counts, but they do not resolve this ambiguity.

This was solved by posing IRL as a maximum margin problem with loss-augmentation.(16) This method leads to a unique solution, but it has some drawbacks when there are no reward functions that make the demonstrated behavior optimal and better then the others. This is a quite common situation which appears for example when the behavior demonstrated by the agent is imperfect.

An imitation learning approach to the problem relaxes the assumption of optimal MDP by using the solution policy’s reward of the same, \( Q_\theta(a, s) = \max_\zeta \theta^T f_\zeta \), within a Boltzmann probability distribution.(17)

\[ P(action \ a|s) = \frac{e^{Q_\theta(a, s)}}{\sum_{action} a' e^{Q_\theta(a', s)}} \]

In order to penalize the squared difference of probability distribution among demonstrated actions and model actions, in this method this distribution is used in the loss function.
3.7.5 Maxim Entropy Inverse Optimal Control

In order to match feature counts and resolve ambiguities in the selection of the distribution, maximum entropy IRL uses the principle of maximum entropy. This principle releases it from the commitment to any specific path required by the constraint and leads to the distribution over behaviors constrained to match feature expectations. The result obtained is a single stochastic policy.

3.7.5.1 Deterministic Path Distribution

In this approach it is considered an entire class of possible behaviors and a distribution over this. As a consequence deterministic MDPs have paths of different length. If the demonstrated behavior is sub-optimal a lot of various distributions of paths match feature counts. Some of the path can be preferred over other without being implied by the path features. This ambiguity is solved with the use of the principle of maximum entropy which selects the distribution which match the feature expectations and has not any other preferences.

\[ \sum_{\text{path} \zeta_i} P(\zeta_i) f_{\zeta_i} = \tilde{f} \]

For deterministic MDPs reward weights are used to parameterized the resulting distribution over paths.

\[ P(\zeta_i|\theta) = \frac{1}{Z(\theta)} e^{\theta^T f_{\zeta_i}} = \frac{1}{Z(\theta)} e^{\sum_{s_j \in \zeta_i} \theta^T f_{s_j}} \]

In this way there are equal probabilities for plans with the same rewards, while on the other hand higher rewards are exponentially more preferred. There are two cases:
• infinite horizons problems with discounted reward weights
• finite horizon problems

In both of these the partition function $Z(\theta)$ always converges if the parameter weights are given. The reward weights that maximize entropy must be convergent if the demonstrated trajectories have a finite number of steps.

3.7.5.2 Non Deterministic Path Distribution

Stochastic transitions among states are produced by actions in general MDPs. The random outcomes of the MDP after the agent chooses the action to do determine the paths. this randomness is taken into consideration in order to determine the distribution over paths.

The MDP is deterministic given an outcome sample, $o$, and a compatible distribution over paths. Thus there is a match between $o$ and the action outcomes of the path.

$$P(\zeta|\theta, T) = \sum_{o \in \text{action outcome}} P_T(o) \frac{e^{\theta^T f_\zeta}}{Z(\theta, o)} I_{\zeta \in o}$$

The value of $I_{\zeta \in o}$, the indicator function, is 1 in the case $o$ and $\zeta$ are compatible, 0 in the other case. This computation is intractable in general. However if the partition function is constant for all $o$ in the space action outcomes and if the behavior suffers within given limits the transition randomness, as a consequence the approximate distribution over paths obtained is tractable.

$$P(\zeta|\theta, T) \approx \frac{e^{\theta^T f_\zeta}}{Z(\theta, T)} \prod_{s_{t+1}, a_t, s_t \in \zeta} P_T(s_{t+1}|a_t, s_t)$$
3.7.5.3 Stochastic Policies

If the partition function seen above converges a distribution over actions of each state (stochastic policy) is provided by the previous distribution over paths. The expected exponentiated rewards that weights the probability of an action is generated considering all the paths which start doing the same action considered.

\[ P(action_{a}|\theta,T) \propto \sum_{\zeta \in \zeta_{a} \cap T} P(\zeta|\theta,T) \]

3.7.5.4 Learning from Demonstrated Behavior

The entropy of the distribution over paths depends on values recovered from the observed data, therefore on the feature constraints. In order to have the maximum probability of the data observed with the same distribution, this value is maximized, knowing its distribution of maximum entropy described and derived above.

\[ \theta^* = \arg\max_{\theta} L(\theta) = \arg\max_{\theta} \sum_{example} \log P(\tilde{\zeta}|\theta,T) \]

The gradient-based optimization methods is used to compute the optima for deterministic MDPs since the function is convex. This method basically computes the discrepancy between learner’s expected feature counts and expected empirical feature counts. It is possible to express

\[ \theta^* = \arg\max_{\theta} L(\theta) = \arg\max_{\theta} \sum_{example} \log P(\tilde{\zeta}|\theta,T) \]
the learner’s expected feature count with the expected visitation frequency of the states. \( D_{s_i} \) represents this value.

\[
\nabla L(\theta) = \tilde{f} - \sum_{\zeta} P(\zeta | \theta, T) f_\zeta = \tilde{f} - \sum_{s_i} D_{s_i} f_{s_i}
\]

The best is represented by a match between the feature expectations. This situation guarantees that the behavior demonstrated by the agent is equivalent to the learner’s one, without taking into consideration the reward weights the agent is trying to enhance. Expectations of the feature values are measured from sample collected from the teacher demonstration. Instead of considering the true values of the agent, these empirical value are considered to be imitated. Saying that it is feasible to bound the magnitude of the features, it is possible to have an error on feature expectations in the order of the logarithm on the number of sample.

### 3.7.5.5 Efficient State Frequency Calculations

The gradient can be computed knowing the expected state frequencies. One way to do this is by enumerating all the existing paths, but this approach is not feasible because they are growing exponentially with the MDP’s time horizon. The approach undertaken by this method consists in a dynamic programming version for the computation of the expected state occupancy frequencies. The techniques used are for instance the forward-backward algorithm for Conditional Random Fields.

\[
Z(\theta, s) = \sum_{\zeta_s} e^{\theta^T f_{\zeta_s}} = \sum_{\text{action}} \sum_a [e^{\theta^T f_{s,a}} \sum_{\zeta_s,a} e^{\theta^T f_{\zeta_s,a}}]
\]
where $\zeta_{s,a}$ is the path starting from state $s$ and acting with action $a$.

In this algorithm the infinite time horizon’s state frequencies are approximated defining a fixed time horizon which is large enough.

### 3.7.6 Predictive Inverse Optimal Control

Maximum entropy inverse optimal control algorithms\(^{(20)}\) try to find a probabilistic action policy.

On the other hand this formulation gives priority to a probability distribution over state-action sequences.

\[
P(a_{1:T}, s_{1:T}) \propto e^{\text{reward}_\theta(a_{1:T}, s_{1:T})} 1_C(a_{1:T}, s_{1:T})
\]

where $\text{reward}_\theta(a_{1:T}, s_{1:T}) = \sum_{t=1}^{T} \theta^T f(a_t, s_t)$, and $1_C$ is an indicator function that return 1 if the sequence belongs to set $C$ of sequences satisfying certain requirements, 0 otherwise. One of this requirement can be for instance reaching a target state.

This distribution over state-action sequences factors into a stochastic policy,

\[
\pi(s_t | a_t) = e^{Q(s_t, a_t) - V(s_t)}
\]

according to the recurrence relationship that is a softened relaxation of the Bellman equation:

\[
Q_{\text{soft}}(s_t, a_t) = E[V_{\text{soft}}(s_{t+1}) | a_t, s_t] + \text{reward}_\theta(a_t, s_t)
\]

\[
V_{\text{soft}}(s_t) = \text{softmax}_{a_t} Q(a_t, s_t)
\]
\[ softmax_x f(x) \equiv \log \sum_x e^{f(x)} \]

The softmax function is a smoothed relaxation of the max function and the terminal state value is either 0 if the terminal state-action sequences satisfies the constraints of set C or \(-\infty\).

### 3.7.7 Softened Value Iteration via Heuristic Based Search

#### 3.7.7.1 Heuristic-based policy approximation

This approach\(^{(21)}\) try to reduce the space of the decision process by truncating the softmax value iteration recurrence, defined in the previous paragraph, at different state \(s \in S_{\text{approx}}\).

This is an heuristic based sampling method, which reduces the state space explored. Some of the state nodes in the map are part of the approximation set. These lead to other state, and their values are approximated with an heuristic function, \(V^+(s)\), instead of computing them recursively which define an upper bound to the softmax value function.

\[ V_{soft}(s) \leq V^+_{soft}(s) \]

In this work it is assumed a monotonous heuristic value functions for the analysis.

\[ V^+_{soft}(s) \geq softmax_a E[V^+_{soft}(s_t)|a, s] + reward_\theta(a, s) \]

The approximation error of the resulting policy is described here\(^{(22)}\): the expected difference in cumulative reward, said approximation loss, between the estimated heuristic-based softmax
policy $\pi_{soft}^+$ and the true softmax policy $\pi_{soft}$ is a difference between the log loss of sequence distribution and the entropy of the true softmax sequence distribution.

$$E_{\pi_{soft},\text{actionoutcomes}}[\text{reward}(s_1:T, a_1:T)] - E_{\pi_{soft}^+,\text{actionoutcomes}}[\text{reward}(s_1:T, a_1:T)] =$$

$$E_{\pi_{soft}^+,\text{actionoutcomes}}[-\log(\pi_{soft}\text{actionoutcomes})] - H(\pi_{soft}\text{actionoutcomes})$$

The approximation state set can be chosen and directly employ this approximate policy for rejection sampling or importance sampling. However, since the true solution policy $\pi_{soft}$ is unknown, this may be extremely sample-inefficient. Therefore the problem is approached refining iteratively the policy estimation.

### 3.7.7.2 Greedy Selection of the Approximation Set

One way to select the approximation set is to expand the most potentially beneficial state. If rewards are not bounded this is a possible optimization:

$$\arg\max_{s_q \in S} E_{\pi^+,\text{actionoutcomes}}[s_q \in s_1:T] V(s_q)^+$$

where the first part, $E_{\pi^+,\text{actionoutcomes}}[s_q \in s_1:T]$, is the expected occurrence of state $s_q$ in the approximate sequence distribution.

This is similar to $A^*$ but the priority used to expand the state $s_q$ depends on the softmax of all paths to $s_q$, thus the first term in the algorithm.
If the heuristic is monotone and admissible the minimum cost path to a state found with an optimal path planning algorithm does not change, but the first part of this algorithm depends on all paths to each state \(s_q\), therefore every path created by the modification of \(S_{approx}\) have to be considered.

### 3.7.7.3 Sampling-based Improvement Algorithm

Instead of using the greedy approach, which is onerous, it is used the random approach. The softmax policy estimate, \(\pi_{soft}\), is iteratively improved refining in a selective way the set \(S_{approx}\) with the sample of the trajectories. They are sampled according to the actual policy estimate and the transition function. The possible successor of a state are added to the approximation set whenever an approximated state is visited. This one after that is removed from the set. This process continues until the maximum path length or the goal is reached. The values along this sampled trajectory are then updated using softened maximum value iteration as described above. The state in the \(S_{approx}\) set have a value estimated by the heuristic \(V^+_{soft}\). The states are expanded from higher to lower probability of having large approximation loss. When it is expanded policy and value are updated.

### 3.7.7.4 Stochastic Exponentiated Gradient Descent Cost Function Learning

State transition cost is computed with the following:

\[
R(s_t, a_t) = \gamma \sum_{j=0}^{f} \theta_j f_j
\]
where \( R \) is the reward, or cost, \( -f- \) is the total number of features and \( \gamma \) is a scaling parameter which defines the general strength of the cost function.

Since \( \theta \) is strictly negative the method chosen is the stochastic exponentiated gradient descent. In this way the reward function behave as a cost function.

\[
\theta_{t+1} = \frac{1}{Z_j} \theta_t e^{\eta (E_p[f] - E_{\gamma p}[f])}
\]

This is the equation used to update the parameters. \( \eta \) is the learning rate of the \( i \)th feature, \( E_p[f] \) is the sum of the values in the path found, \( E_{\gamma p}[f] \) is the sum of value in the demonstrated sequence, \( Z_j \) is a normalizing factor for the feature of type \( j \).
CHAPTER 4

IMPLEMENTATION

4.1 Introduction

The following chapter describes the step by step implementation of the project.

The project is developed on Linux Ubuntu 14.04 lts 64 bit and it is based on Node.js for the reasons described in the background chapter. The starting point of the project is the module described above, ardrone-autonomy, which is based on node-ar-drone previous work. As it is described above, in the Background chapter, the second one basically maps the command of the API of the quadcopter to node, while the first create some command that allows to move the drone without considering the velocity but just the space. From this point i improved the localization system and developed an autonomous flight, since the drone is able to move from a starting coordinate to a destination one selecting the best trajectory according to some features. In the end i added a dynamic obstacle and i tested the implementation.

4.2 System specification and environment

The operating system used is Linux Ubuntu 14.04 lts 64 bit. The installation process for setting up the environment can be found in the appendix B.

Basically the main steps are the installation of Node.js, opencv, and the nodecopter modules. The drone has its wifi connection, therefore the first step is to connect to it. The drone is acting like a server and sends all the data through the network.
The flight environment is a 3 by 3 by 2 meters cube indoor. The static obstacles are located in the following point:

![Figure 15. Flying area](image)

Since the value of the position of the drone refers to the center of gravity the obstacles size is the real size of themselves plus half size of the drone. In this way it is possible to consider the dimensions of the drone in the distance from the obstacles.

4.3 **Localization System**

The localization of a quadcopter is an important problem and many factors affect its precision. One of these is for example inertia, since it flies in the air the friction is not enough to control it precisely. For instance if it goes in a direction and it receives the command to turn
right due to inertia it will not just turn right but its position will keep going in the previous
direction for a while. In addition to this the quadcopter’s sensors are not perfect and the error
growth is significant and increases continuously. An estimate of the quadcopter position is done
using the data received from accelerometer and gyroscope and using the kinematics formulas.

\[ x_{t+1} = x_t + v_x \cdot dt \cdot \cos(yaw) - v_y \cdot dt \cdot \sin(yaw) \]

\[ y_{t+1} = y_t + v_x \cdot dt \cdot \sin(yaw) - v_y \cdot dt \cdot \cos(yaw) \]

\[ \text{yaw}_{t+1} = \text{yaw}_t + \text{dyaw} \]

Where \( x_{t+1}, \ y_{t+1} \) and \( \text{yaw}_{t+1} \) are the actual estimation of the position of the drone.
\( x_t, \ y_t \) and \( \text{yaw}_t \) are the last known position estimated, \( v_x \) and \( v_y \) are the value sent
by the quadcopter and \( \text{dyaw} \) is the difference the actual value of the yaw and the last known
one, since the yaw position is important with respect to something, and not its absolute value.
The time frame \( dt \) is set to one-fifteenth of second because in demo mode it receives the naviga-
tion data fifteen time in a second. This is another reason of imprecision, since it is improbable
that the speed is the same for that fraction of second. This fact is not relevant for a single
fraction, but after a lot of samples the error grows. In addition, since the packets are sent
through a UDP channel there could be some loss. The average of received packets is around 15
per second, but it is not always the same. this can influence significantly the approximation.
Figure 16. The blue line represents the direction of the frontal camera, while the red line the perpendicular direction.

The Figure 16 shows the reason of the formulas written above. The blue line is the direction of the camera so in the drone coordinate system the velocity on the x axis, while the red line in the y axis. Therefore it is possible to understand that they derive by a rotation.
The error generated by this estimation can be partially corrected with some external observations of the environment. Knowing the drone relative position to some external objects, which positions are known, it is possible to do this correction. For example in this thesis I used some tag that I laid on the ground in the following coordinates (x,y):

- (0,0)
- (0.42,1)
• (0.61, 2.64)
• (1.8, 0)
• (1.3, 1.4)
• (2.192)
• (1.52, 2.75)
• (2.75, 0.7)
• (2.75, 2.32)

These positions are selected in order to have more control in proximity of the static obstacles along the principal paths. Since the drone can correct its position when it is over them they are placed mainly near the obstacles.

Figure 18. Roundel tag position

Figure 19. Roundel tag area
Through the bottom camera the quadcopter can see the tags on the floor. Since these tags (Figure 17) have no differences among themselves the coordinates where the quadcopter thinks to be are used to locate it in a range where the closest tag is. In this way the tag seen by the camera is taken from a matrix that contains all the tags positions. Tags are positioned at a distance that, flying at a low altitude (up to 2 meters) since we are indoor, avoid the quadcopter to see more than one of them in the same time (Figure 19). Although it can manage to see up to four tag I preferred not to do so because the way in which the drone gives the value of them in the array is not easy to use. All the information about the tag seen by the drone are stored in an array. This special tag shown in Figure 17 is automatically recognized by the drone.

![Figure 20. This image shows how the drone store the tags it sees in the array.](image-url)
They are ordered according to their relative position to the drone. The Figure 20 shows schematically four different positions of the tags. The figure colored magenta is the drone, while the other are tags. Their information are stored in a vector giving priority to the tags detected in front of the drone, therefore the first element of the array would contains the information about the tag i have colored red in this figure. The second priority parameter is the y position. Therefore after the red one there should be the blue or the yellow according to x position. Since the value of y for the yellow tag is higher it will be the second in the array and so forth.

With the image information received from the bottom camera it has been possible to understand the relative position of the quadcopter with respect to the tag. Basically the information the drone sends are distance and relative position of the tag on the screen. Therefore x is positive if it is in front of the drone (the high part of the screen), while negative if it is behind (the lower part of the screen), and y is positive if it is on the right of the drone (analogously the right part of the screen), and obviously negative on the left. The quadcopter is able to calculate the distance from the tag because it knows the real size of the object seen (Figure 17), the focal length of the camera, the size of the image in pixel and the value of pixel per millimeter of the camera. The way in which the distance is computed is not known since the drone software is not public, but probably the formula used is similar to the one described in the previous chapter.

In order to determine the position in the coordinate it is possible to use the position on the screen, the distance and the camera intrinsic matrix as described above in the Background chapter. Doing this operation I ignore the drone translation and yaw since i want X,Y in the
drone coordinate system. This because i compare it to the relative position with respect to the tag instead that with the absolute position of the tag.

Figure 21. Roundel coordinate system

Therefore the module of the camera receives as input the coordinates \((x,y)\) on the screen and the distance from the tag and returns the position of the drone with respect to the tag roundel positioned on the floor in the drone coordinate system. Thus the \(x\) axis is perpendicular to the frontal camera, while the \(y\) axis lays on it. For this reason in order to map the coordinates to our environment they are rotated by 90 degree.
After the position of the drone and the position of the tag are determined in the same coordinate system it is possible to use the second step of the EKF, the correction. It calculates the difference between the position where it believes to be and the position where it is, calculating the error. Thank to the information received by this process we are able to correct the position following the process described in the Background chapter.

In this way, it is possible to reduce the position estimation error and have an acceptable control over it. Thank to this it is possible to proceed on an higher level, where the aim is develop an autonomous flight through the cost function he will learn.

4.4 Collecting Data

In order to apply the algorithm of IRL a lot of data have to be collected. Therefore i run some flights along different trajectories that are considered optimal by the expert, me in this case, according to different features. In specific the features considered are flying high or low for three different trajectories. One passes on one side of the flying area, close to the wall, another passes on the other side of the flying area close to the net, which delimit the area itself, and the last one passes among the obstacles.
A lot of data have been collected for each one of the trajectories shown above. All these data collected are used to build the cost function and to determine the best path. From Figure 22
to Figure 27 there is one sample of each one of the trajectories showed to the drone. Other samples of each one of these paths can be found in the appendix A.

4.5 **Inverse Reinforcement Learning algorithm**

In the next lines I am going to describe how the algorithm works, knowing everything that is described in the background chapter.

The algorithm has to know the starting position and the ending position. These can be chosen according to the needs. Stochastically it computes the path which links the two consecutive positions until it reaches the destination. In each step it chooses a random action from a probabilistic policy distribution generated from the estimated state-action values, where the state-action value is the cost of transitioning to the next state plus the value of the next state. Each state has a computed value or a value estimated with the heuristic. Among this the value selected is random, but it is more likely to be selected one with higher probability. This probability depends on the path, therefore the neighbors, and the features computed with the training paths. The heuristic is a simple softmax algorithm which basically chooses the state with the lowest step cost. The step cost is determined by the direction and angle of travel, assuming for instance more onerous change a direction instead of keeping one. Once the destination is reached or the maximum path length is exceeded, it returns to the beginning updating the value of each node in the path. For each node in the path it updates the value according to the cost transition and the neighbor’s value. This sequence of action is repeated more times and at the end the algorithm returns a set of paths with their probability. Of course
the one with highest probability is more likely to be chosen. This algorithm basically build a policy.

The policy returned by the same depends, of course, by the features expressed. The drone has to define the characteristics it wants to maximize in order to have the best path between two point. This behavior is normal in nature. For example a person who drive home usually follows the same way if the requests are the same (features). But sometimes it has other features, for example this person has to buy something in a store that is not on his normal way, or due to a car incident(dynamic obstacle) it has to change street. This denotes how, with highly probability there is a preferred trajectory, but there are probabilities of him following others according to new features expectation or new obstacles.

The features used are 100. They are chosen considering, for each of the parameters used in the expert demonstrations (altitude of flight and the three main trajectories: fly near the right or the left side of the flying area or among the obstacles), different combinations of angle and direction of the flight, distance from walls and relative position to the goal.

The action class is composed by any kind of movement the quadcopter can do as a six degree freedom vehicle. These are up, down, right, left, forward and backward.

The algorithm has been trained for different cell sizes in order to try and evaluate different situations. In specific the environment has been divided in three different cubic cell sizes:

- 20 cm
- 30 cm
- 40 cm
In order to consider a cell totally occupied by an obstacle it is enough that a small part of it is containing the obstacle or a part of the same. This may lead to a problem. Different cell sizes can have different solutions in some circumstances.

For example there could be enough space for the drone to pass between two obstacles but it sees as there is not as it is shown in Figure 28. This is due to the fact that the cells between the obstacles are partially occupied, but they result totally occupied, therefore the algorithm try to find another way. The bigger the cells are the more probable is to be in a situation like this. On the other hand reducing the cells size it is possible to minimize or avoid this problem. An
example is in Figure 29 where it is possible to see that this time there is a cube free for the drone to pass. Anyway the drawback here is that the number of possible paths grows significantly and the time needed for the computation increases a lot. The time needed by the algorithm changes according to the length of the path and the size of the cells. The shorter it is the path the fast it is the execution. A possible optimization of this problem in order to have a correct path but an acceptable execution time is to use larger cells in the space where there are no obstacle and smaller ones in the middle of the obstacle. This approach has been used for some of the test done, but in an area relatively small, like this one, the time difference is significant, since it is in the order of ten seconds, but it does not change the magnitude.

Applying the result of this algorithm to the drone in the static environment is easy. Before taking off the program waits the result of the same, it adds all the instructions to the tasks queue and then the process starts until it reach the end of the path. This path is built with 30 centimeter cell, and since it knows all the sequence of steps before taking off, and in this initial configuration the obstacle are static, it is possible to evaluate in advance if the drone is flying near some obstacles. If this is the case the part of the path close to the obstacle is recomputed with smaller cell size (20 centimeter). Doing so the time does not increase a lot and the error described above can be minimized. This distance parameter has been set to 1 cell. If it is closer then it is recalculated.

Here there is a picture of one flight (Figure 30) done with this method. The red line is the trajectory followed by the quadcopter, while the black line is the trajectory suggested by the algorithm. The result of other trajectories are in the Testing and analysis chapter.
This is the flight from the position of coordinates (0,0,0) to the coordinates (275,232,0) flying among the obstacles at a low altitude and the cell size considered is 30 centimeter. Figure 31 shows the state visit density probability for this trajectory. It is possible to see that the states with higher probability are the ones along the trajectory coordinates, but there is probability different from zero either in other states close to the main trajectory.

4.6 Dynamic Obstacle

As we can see by the results shown in the previous section the drone is able to flight alone and autonomously, according to some features, in a static environment given starting and ending position. Therefore the next step is to add dynamic obstacles. This changes deeply the problem
and affects significantly the work to do. In order to develop this part of the thesis i used a green colored cube of 30 cm of edge as an obstacle. The steps i have followed are the following:

- set the frontal camera and elaborate the images in order to recognize the green color.
- set the frontal camera parameter in order to calculate distances from the object using the same method described above for the bottom camera.
- when it sees the object it emits an event which is collected by the main program. If the drone is close enough to the obstacle the program stops all the tasks of the drone.
- the new obstacle is added to the map.
- the new best trajectories is computed considering the new obstacle.
- the drone proceeds to the destination when the new path is computed.

Some problems appears during the processing of these stages. The first one consists in locating the obstacle, since the distance calculated from the camera of the drone to the obstacle is the line which links these two points, the Euclidean distance, so the sum of the distances along all three axis. In order to find the offset of the obstacle with respect to the drone camera along the x axis it is necessary to make some calculus.
As shown in Figure 32 the y and z value are the result of the computation described in the background done with the camera calibration matrix. Having this two values and the distance from the drone to the obstacle it is easy to compute the offset on the x axis.

\[ x_{\text{off}} = \sqrt{\text{dist}^2 - z_{\text{off}}^2 - y_{\text{off}}^2} \]

The value obtained is not perfect, because here more error are summed up together. The value of the y is affected by the yaw of the drone because the precision in the control of it is 5 degree, therefore if it is in the same position, it can be with a yaw in the range from -5 degree to +5 degree, fact that creates some imprecision, specially on the value of the y. This reflects on the value on the x axis as a consequence of the computation described above. This is evident in the difference between Figure 33 and Figure 34, where the drone is in the same position, but has different yaws, both of them are recognized as zero.
The value on the z is one of the most accurate but when the drone change direction it as a range of oscillation around 10 centimeter. In addition to these it is necessary to remember that either if it is small there is always an error on the drone estimation of its position. All this value together can create some problem, but the results obtained are enough precise for the context of this thesis, since usually the total error is around 10 or 15 centimeter and the cube size is bigger. In this way beside precise situations where this error allows one cube close to the obstacle to be free instead of occupied there are no drawbacks. As it is shown in Figure 35 different estimated positions of the obstacle do not affect the real behavior of the system, beside the case where a cell results free instead to be occupied.
After all the parameter relative to the camera were set a first tentative of capturing the distance has been done with the quadcopter stopped on the floor, in order to avoid the error caused by its motion. In this way i evaluate its precision in a static situation in order to check if the measurement done are correct.

| TABLE III |

| OBSTACLE CENTER POSITION DIFFERENCES |
|-----|-----|-----|
|     | x   | y   | z   |
| real| 220 | 100 | 75  |
| seen| 223 | 105 | 74  |

Figure 35. Obstacle x offset
As it is possible to see by the Table III the values measured are precise in a situation where the delay of the images does not matter because the drone does not move, so every picture is the same. The distance on the x position summed up in the table is computed from the position of the frontal camera.

The second is that the camera has a delay of about one second, therefore it is not possible to proceed at high speed or either flying at a low speed in some cases. If the obstacle appears in a position close to the drone while it is moving it will not have enough time to stop since it sees it after this delay. In addition this increases the error on the estimation of the position of the obstacle, because the drone keeps moving during that delay while the picture is elaborated. The partial solution developed for this issue, caused by the decoder of the images, is to make the drone stop farther with respect to the obstacle. In this way either if it keeps moving for one second it will not hit the obstacle. However with this method the additional error generated on the x coordinate for the position of the obstacle is not detected. Therefore when it adds the position of the obstacle relative to the drone to the position of the same in order to get the general coordinates of the position of the obstacle in the environment the computed position will be closer to the obstacle since it moves for another second. However this error can be partially avoided. For this reason i made an estimate of the space flown in that time slice in order to subtract it to the value computed and reduce the error. Basically this is the final value:

\[ X_{obf} = x_{drone} + dist_{camera,dronecenter} + \sqrt{dist_{camera,obstacle}^2 - z_{off}^2 - y_{off}^2} - X_{flown} \]
The third one concerns the computation cost of the best path that take a lot of time. The main reason of this computation cost is the heuristic, since the one used is simple and its not the optimal one. In this time slice, between the instant when the drone sees the obstacle and decides to stop and the moment it starts moving again, it is difficult to stabilize the drone because as written above node.js is a single thread platform and when it has to make a lot of computations on images it suffer and the control of the drone has less dedicated time during the execution of the program. In addition to that if the obstacle is moving for the time the algorithm completes there will be no obstacle anymore. Since the problem of a totally dynamic environment can be a further improvement of this project i started approaching this issue whit a dynamic obstacle which is static. Therefore the obstacle is static, but the quadcopter does not know its position at priori. In this way it is possible to simulate the dynamic situation with an affordable approach considering the computation capability of this drone and the algorithm time complexity. Since the time for it to wait for the results is high i made an easy optimization. When it sees the obstacle height, if the drone altitude for the next steps of the path is enough to not worry about the obstacle the call to the path algorithm is avoided since the obstacle does not intersect the quadcopter path. In addition to that when it calls the path algorithm after it sees an obstacle, it can switch the cell size to 30 centimeter instead of 20 if it is in an area not too close to other obstacle.

In Figure 36 and in Figure 37 is illustrated the trajectory followed from the point with coordinates (0,0,0) to the one with coordinates (275,232,0). In this simulation the feature
asked to the drone are to fly low in the middle of the obstacles and the size of the cell is set to 30 centimeter.

Figure 36. Obstacle observed position

Figure 37. Object real position

In this case the position seen by the camera of the drone (Figure 36) and the real position of the obstacle (Figure 37) differ for a few centimeter, therefore the estimation of the position is almost the same.
Anyway this (Table IV) is fortuitous because in this case i do not consider the motion during
the delay of the camera and i do not consider the fact that the camera is not in the center of
the quadcopter, but is located in front. In this case the space flown by the drone during that
time was very similar to the distance from the camera to the center of the drone, so around 18
centimeters.

The next step is to add the correction described above. The results obtained are summed
up in the next chapter.
CHAPTER 5

TESTING AND ANALYSIS

In this chapter some of the result obtained are shown and briefly described.

5.1 IRL results

For the following figure the black lines are the outputs path of the algorithm, while the red lines are the trajectories followed by the drone.

Figure 38. IRL a  Figure 39. IRL b  Figure 40. IRL c
The starting point for these first three examples is on the bottom left corner of the flying area, which coincides with the coordinate with x and y equal to zero, or about zero. While the destination is the same as the coordinate of the farthest tag on the floor (275,232). As summed in Table V different configuration of altitude are tested keeping the same feature, flying among the obstacles, and cell size. The results obtained are coherent since it changes altitude if the destination is on a different altitude. Even if the feature asks for a flight high, it would be too onerous to change two times altitude for a short path like this if the destination is on the same z. In Figure 40 it is possible to see the change in altitude since it is requested to flight high.

<table>
<thead>
<tr>
<th>figure</th>
<th>(x_{in})</th>
<th>(y_{in})</th>
<th>(z_{in})</th>
<th>(x_{goal})</th>
<th>(y_{goal})</th>
<th>(z_{goal})</th>
<th>cell size</th>
<th>altitude</th>
<th>trajectory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 38</td>
<td>25</td>
<td>45</td>
<td>100</td>
<td>275</td>
<td>232</td>
<td>100</td>
<td>30</td>
<td>low</td>
<td>middle</td>
</tr>
<tr>
<td>Figure 39</td>
<td>25</td>
<td>45</td>
<td>100</td>
<td>275</td>
<td>232</td>
<td>100</td>
<td>30</td>
<td>high</td>
<td>middle</td>
</tr>
<tr>
<td>Figure 40</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>275</td>
<td>232</td>
<td>150</td>
<td>30</td>
<td>high</td>
<td>middle</td>
</tr>
</tbody>
</table>
Figure 41. IRL d

Figure 42. IRL e

Figure 43. IRL f

TABLE VI

<table>
<thead>
<tr>
<th>figure</th>
<th>(x_{in})</th>
<th>(y_{in})</th>
<th>(z_{in})</th>
<th>(x_{goal})</th>
<th>(y_{goal})</th>
<th>(z_{goal})</th>
<th>cell size</th>
<th>altitude</th>
<th>trajectory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 41</td>
<td>170</td>
<td>60</td>
<td>100</td>
<td>275</td>
<td>232</td>
<td>100</td>
<td>30</td>
<td>low</td>
<td>middle</td>
</tr>
<tr>
<td>Figure 42</td>
<td>170</td>
<td>60</td>
<td>100</td>
<td>275</td>
<td>232</td>
<td>100</td>
<td>20</td>
<td>low</td>
<td>middle</td>
</tr>
<tr>
<td>Figure 43</td>
<td>200</td>
<td>192</td>
<td>100</td>
<td>275</td>
<td>70</td>
<td>100</td>
<td>20</td>
<td>low</td>
<td>middle</td>
</tr>
</tbody>
</table>
Figure 41 and Figure 42 show the problem described in the previous chapter generated by the cell size. In the first case the cell size is 30 and for that trajectory the algorithm prefers to choose the direction close to the wall since there are not a lot of possible paths which pass through the obstacles. While in the second picture the drone passes between the two obstacle since more room is detected for him to pass in that point.

Figure 44. IRL g  
Figure 45. IRL h  
Figure 46. IRL i
TABLE VII

FLIGHT TRAJECTORIES 3

<table>
<thead>
<tr>
<th>figure</th>
<th>$x_{in}$</th>
<th>$y_{in}$</th>
<th>$z_{in}$</th>
<th>$x_{goal}$</th>
<th>$y_{goal}$</th>
<th>$z_{goal}$</th>
<th>cell size</th>
<th>altitude</th>
<th>trajectory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 44</td>
<td>200</td>
<td>192</td>
<td>100</td>
<td>275</td>
<td>70</td>
<td>100</td>
<td>20</td>
<td>low</td>
<td>middle</td>
</tr>
<tr>
<td>Figure 45</td>
<td>200</td>
<td>192</td>
<td>100</td>
<td>275</td>
<td>70</td>
<td>100</td>
<td>30</td>
<td>low</td>
<td>middle</td>
</tr>
<tr>
<td>Figure 46</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>275</td>
<td>232</td>
<td>100</td>
<td>20</td>
<td>low</td>
<td>net</td>
</tr>
</tbody>
</table>

Figure 43, Figure 44 and Figure 45 show different configuration for a flight with different starting position and different goal.

An interesting results is shown in Figure 46. In this case the algorithm prefers to pass among the obstacles instead of passing close to the net, even if the feature asked is the last one.
This is explained by Figure 47 where the probability distribution highlight that the one followed would be the favorite trajectory. These case is analyzed better at the end of this section.
**TABLE VIII**

<table>
<thead>
<tr>
<th>figure</th>
<th>$x_{in}$</th>
<th>$y_{in}$</th>
<th>$z_{in}$</th>
<th>$x_{goal}$</th>
<th>$y_{goal}$</th>
<th>$z_{goal}$</th>
<th>cell size</th>
<th>altitude</th>
<th>trajectory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 48</td>
<td>61</td>
<td>264</td>
<td>100</td>
<td>275</td>
<td>232</td>
<td>100</td>
<td>30</td>
<td>low</td>
<td>net</td>
</tr>
<tr>
<td>Figure 49</td>
<td>61</td>
<td>264</td>
<td>100</td>
<td>275</td>
<td>232</td>
<td>100</td>
<td>30</td>
<td>low</td>
<td>net</td>
</tr>
<tr>
<td>Figure 50</td>
<td>180</td>
<td>30</td>
<td>100</td>
<td>200</td>
<td>192</td>
<td>100</td>
<td>30</td>
<td>low</td>
<td>middle</td>
</tr>
</tbody>
</table>
Figure 48 and Figure 49 show a configuration where the path is close to the net, and the heatmap above (Figure 51) strengthen the reason why it is selected.
Figure 52. IRL m

Figure 53. IRL n

Figure 54. IRL o

TABLE IX

<table>
<thead>
<tr>
<th>figure</th>
<th>$x_{in}$</th>
<th>$y_{in}$</th>
<th>$z_{in}$</th>
<th>$x_{goal}$</th>
<th>$y_{goal}$</th>
<th>$z_{goal}$</th>
<th>cell size</th>
<th>altitude</th>
<th>trajectory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 52</td>
<td>180</td>
<td>30</td>
<td>100</td>
<td>200</td>
<td>192</td>
<td>100</td>
<td>30</td>
<td>low</td>
<td>middle</td>
</tr>
<tr>
<td>Figure 53</td>
<td>180</td>
<td>30</td>
<td>100</td>
<td>200</td>
<td>192</td>
<td>100</td>
<td>20</td>
<td>low</td>
<td>middle</td>
</tr>
<tr>
<td>Figure 54</td>
<td>180</td>
<td>30</td>
<td>100</td>
<td>200</td>
<td>192</td>
<td>100</td>
<td>20</td>
<td>low</td>
<td>middle</td>
</tr>
</tbody>
</table>
Last four images show another trajectory where the path which passes among the obstacles is selected even if the drone is close to the wall. This happens because the feature states its preference to pass in the middle. Figure 55 shows the distribution for the case with cell size equal to 20 centimeter.

Figure 55. Distribution 3

All the previous examples show different paths, with different starting points, different goals, different parameters set. In all the example it is possible to see how the algorithm works. It is evident to see, for instance, how it changes its trajectory according to the requests. If the feature
asked is middle (among the obstacles), it gives priority to the path among them (Figure 38). On the other hand if the same is set to net it goes on the side of the flying area close to the net (Figure 48), but in some circumstances either if net is set it could prefer to pass in the middle (Figure 46). For example the following distribution highlight this situation. Figure 56 is the case where the starting point and the goal are the two opposite corners, therefore (x,y) equal to (0,0) and (300,300). The features are to fly near the net at a low altitude. The cell size is 20 centimeter. Figure 57 has the same settings as the previous one, the only difference is the cell size, which is set to 30 centimeter.

This difference is probably caused by the issue you mentioned in the previous chapter where the smaller cell size gives more room for the path to travel between obstacles. It is possible to
see in the second image that there is a small distribution that takes a similar path to the first image. But, the larger cell size may be prohibiting a strong set of paths through that space. Maximum entropy ioc has this result where paths through an area with a larger distribution of other possible paths will be more probable.

5.2 Test with Dynamic Obstacle

The huge error obtained in the following two examples (Figure 60 and Figure 61) is caused by the fact that half the size of the drone is added to the measure as explained before, but the estimate of the space flown by the same during the camera delay time has not been computed yet. As a consequence in these cases the drone evaluates a wrong position in the axis parallel to the flight direction (Table X) because the space flown during the delay of the camera has to be subtracted from its measure.

In particular this two example are taken in calculating the trajectory between the coordinates (0,0,0) and (275,232,0) following the features fly low in the middle of the obstacles. In both the cases to speed up the calculus of the new trajectory 30 centimeter cells are used, instead of the 20 centimeter one used from the starting point, after it sees the dynamic obstacle.
TABLE X

OBSTACLE CENTER POSITION DIFFERENCES

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>real</td>
<td>215</td>
<td>192</td>
<td>75</td>
</tr>
<tr>
<td>seen</td>
<td>236</td>
<td>186</td>
<td>76</td>
</tr>
<tr>
<td>seen</td>
<td>225</td>
<td>197</td>
<td>75</td>
</tr>
</tbody>
</table>

With the values returned by the cost function the suggested path in this case makes the drone keep the same altitude and flight around it, on his side, because it imagines the obstacle to be farther from the original place where it really is.
In Figure 58 it is possible to see the situation described above. In particular the green obstacle is where it really is, the gray and orange one are the two places where the drone thinks the obstacle is in the two different flight shown. The black line is the trajectory suggested by the algorithm which generates it having has parameter the location of the obstacle, but not the real one. In Figure 59 it is possible to see that the probability of a state to be visited is similar in the two cases: fly over it or go around. Since change altitude is more onerous it decides to fly around it.

The real position of the obstacle is closer than the one seen, thus the drone comes very close to the obstacle. It did not touch it but a little oscillation would have been enough for making that happen. Looking at Figure 58 it is evident to understand why the drone did not
crash against the obstacle either if its estimate was wrong. In Figure 60 and in Figure 61 are displayed the trajectories followed by the quadcopter in the two cases.

![Figure 60. Obstacle error observed position a](image1)

![Figure 61. Obstacle error observed position b](image2)

In order to reduce this error the following step, as described above, is to make an estimation on the space flown by the drone in that time slice and use that estimate to correct the position of the obstacle.

A lot of different flight have been done to collect an amount of data big enough to find a good average to predict the space flown during the delay of the camera. Of course different situations have different results, for instance if the drone sees the obstacle while it is moving at full speed to the next step in the path, or if the drone is slowing down because it has just reached the
partial goal and it is waiting for the next step in the path. This two situations denote that this estimate can not be always perfectly predicted, but averagely speaking from the data collected this value is around 30 centimeter. For this reason i used this as correction parameter. A farther approach can make more precise estimation, maybe updating and keeping memorize the space flown during the last delay. Therefore since the camera delay is always around one second it is possible to keep the value of the space flown in the last one second and subtract it from the estimate. Anyway, since this is not the main focus of my thesis and the estimate i have done is precise enough, i have not implemented this for now.

After this correction the results obtained are way more precise. The following four example shown the behavior of the drone in these cases (where the correction path is always computed using 30 centimeters cells in order to reduce the time to wait for the drone to move):

- From (0,0,0) to (275,232,0) flying low in the middle of the obstacle, using a cell size of 20 centimeters.
- From (0,0,0) to (275,232,0) flying low in the middle of the obstacle, using a cell size of 30 centimeters.
- From (60,240,0) to (275,232,0) flying low close to, using a cell size of 30 centimeters.
- From (60,240,0) to (275,232,0) flying low close to, using a cell size of 20 centimeters.

In the following examples the black line represents the initial trajectory suggested by the algorithm, the gray line is the new trajectory after it sees the obstacle and the red one is the real trajectory done by the drone.
In this case the obstacle position is more precise with respect to the previous situation. In Table XI the measured value are compared with the real ones. In this case the error is acceptable.
As a consequence, it is showed in Figure 63 that the distribution has an higher density in the path over the obstacle, all the other are almost zero.
Similar to the previous example either in the case of Figure 64 (Table XII) the error is acceptable, since the cell occupied by the obstacle are almost the same.

**TABLE XII**

<table>
<thead>
<tr>
<th>OBSTACLE CENTER POSITION AFTER CORRECTION 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>real</td>
</tr>
<tr>
<td>seen</td>
</tr>
</tbody>
</table>

The next two are the same kind of test, done with the obstacle in another position. This time the obstacle is positioned on the side of the area, close to the net. The center of it is summed in Table XIII. Like the previous cases the black line is the original trajectory, the gray line is the trajectory after the re-computation and the red line is the trajectory followed.
Figure 65. Obstacle avoidance 3

TABLE XIII

OBSTACLE CENTER POSITION AFTER CORRECTION 3

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>real</td>
<td>200</td>
<td>275</td>
<td>75</td>
</tr>
<tr>
<td>seen</td>
<td>195</td>
<td>261</td>
<td>82</td>
</tr>
</tbody>
</table>
The following is the same test done with a smaller cell size, 20 centimeter.

Figure 66. Obstacle avoidance 4
The results obtained are compatible with the problem approached and the instruments used. Anyway it is possible to make another easy improvement. Since we know in advance the trajectory to follow, when the drone sees the obstacle before calling the function to compute the new path it is possible to check if the trajectories of the drone does intersect the obstacle position or not. In the case there is not any intersections the drone can avoid considering the obstacle seen and keep following its path. To check this intersection the obstacle is considered bigger then what it really is because the drone does not follow precisely the path since it has some error. In this way it makes sure there are no possibilities it will crash. While in the case there is an intersection it acts as described above. Therefore it stops, adds the new obstacle to the map and calculates the new best path to the destination.
In the following figure this improvement is tested and the result is shown. The starting coordinate is (0,0,0) and the destination one is (275,232,0). It is asked to fly low among the obstacles.

Figure 67. This are the expert trajectories
CHAPTER 6

CONCLUSION

This work demonstrates that the aim stated in the introduction is achievable. The work done in this thesis is a first step in the direction of applying this kind of algorithm to drones in real environment, of course a lot of work has to be done but it is evident how a drone can learn from an expert.

The problem found with dynamic obstacles are solvable integrating this method with some obstacle recognition and computer vision algorithm. In my idea seen the time requested by the algorithm i used, it would be a good approach to use this one to generate the global path of the flight and then use the other ways to avoid dynamic obstacle basing it on the analysis of the picture, or integrating other cheap instrument like for instance a sonar. This probably will not return the best path, but it is a good compromise in my opinion.

In a world where the Internet of Things theory is totally or mainly applied this kind of approach would be perfect, since the trajectory of the other objects can be predicted or at least known before the drone has to deal with them. In this way it can recalculate the trajectory in advance avoiding loosing a lot of time. In this kind of approach in order to keep the time complexity under control it is possible, like i have partially done in this thesis, to analyze different part of the trajectory with different precision. For instance the algorithm should use smaller cells around cities were there are more problem and huge cell in area around deserts where there are almost no obstacle. In addition after it has computed the main trajectory
while it is flying it can start analyzing better (with smaller cells) the next step. In this way the autonomous navigation would be totally achieved.
APPENDICES
Appendix A

TEACHER FLIGHT

Figure 68. Example 1  
Figure 69. Example 2  
Figure 70. Example 3

Figure 71. Example 4  
Figure 72. Example 5  
Figure 73. Example 6
Appendix A (continued)

Figure 74. Example 7
Figure 75. Example 8
Figure 76. Example 9

Figure 77. Example 10
Figure 78. Example 11
Figure 79. Example 12
Appendix B

ENVIRONMENT INSTALLATION AND SETTINGS

B.1 Install Sublime

- sudo add-apt-repository ppa:webupd8team/sublime-text-2
- sudo apt-get update
- sudo apt-get install sublime-text

B.2 Install Node.js

- sudo apt-get install g++ curl libssl-dev apache2-utils
- sudo apt-get install git-core
- download node
- extract
- ./configure
- make
- sudo make install
- make test
- make doc
- which node
Appendix B (continued)

- sudo apt-get update
- sudo apt-get install nodejs npm

B.3 Install ar-drone

sudo npm install ar-drone

B.4 Install ardroned-autonomy

- sudo git clone https://github.com/eschnou/ardrone-autonomy.git
- sudo npm install ardrone-autonomy

B.5 Install opencv

- sudo apt-get install libopencv-dev python-opencv
- npm install opencv

B.6 Install ffmpeg

- sudo apt-add-repository ppa:jon-severinsson/ffmpeg
- sudo apt-get update
- sudo apt-get install ffmpeg
CITED LITERATURE


VITA

NAME: Luca Graglia


EXPERIENCE: Research on the storage platform for cloud computing systems, Istituto Superiore Mario Boella, 2013.