University of Birmingham

School of Computer Science

MSc Robotics
Second Semester Mini-Project Report

Visual Odometry and High Level Controller for an Aerial Robot

Author:
Saif Sidhik (1734904)

Supervisor:
Prof. Jeremy L Wyatt

April 24, 2017

Abstract

This report presents the implementation, tuning, experimentations and results thereof of a PID controller on a quadcopter, and a visual odometry algorithm on a synthetic dataset. Basic theory required for their implementation are also explained briefly. The controller developed uses data from a motion capture system to obtain the current pose of the drone in space, and guides it to the destination pose using PID control. The controller was tuned manually and its performance was tested experimentally on the task of following two simple trajectories using two different methods of navigation. One method minimises both the position and orientation of the drone simultaneously, while the other first corrects the position and then the orientation. Though both methods proved to be applicable for navigating simple trajectories, they produced different results depending on the complexity of the trajectory.

The visual odometry algorithm developed can estimate the trajectory of a camera using the sequence of images captured by it along the way. Given a set of keypoints whose 3D positions are known, the algorithm matches these keypoints in the image and uses the 3D-2D correspondences to estimate the pose of the camera at that frame. The features are tracked across the next frames and new correspondences are computed if required. The trajectory of the camera motion is obtained from the camera poses estimated at each frame. The algorithm was tested on a synthetic dataset with and without applying a novel technique called the 'Reprojection Filter' technique, which removes features responsible for high reprojection errors. The technique was tested experimentally and found to reduce the total mean reprojection error by a significantly large margin, improving the trajectory estimation.

Keywords: PID, control, High-level controller, quadcopter, quadrotor, aerial robot, visual odometry, reprojection error, camera trajectory, camera pose estimation.

Acknowledgements

Firstly, I would like to thank Prof. Jeremy L Wyatt for his supervision on this project and for pushing me to achieve beyond the initial objectives. I am also extremely indebted to Ermano Arruda, without whose help and guidance this project would not have been possible. I would like to thank him for providing me with the base codes and resources that helped me get started, for all the hours he had spent to patiently explain theories and clear the silliest of my doubts, and for letting me use his codes for improving and testing the performance of mine. I would also like to thank Diar Karim for introducing me to this topic, for getting me 24-hour-access to the Posture & Balance Lab where the entire project was developed and tested, and for all his support and motivation that was monumental in the completion of this project. Their enthusiasm and passion towards their work are contagious and inspirational.

Contents

Abstract i

List of Figures vii

List of Tables ix

1 Introduction 1

2 Hardware and Workspace 2
2.1 The Drone - Parrot AR.Drone 2.0 . . . 2
2.2 Base Station . . . 3
2.2.1 The Ground Truth System . . . 3
2.2.2 The Control Station . . . 3

I PID Controller for the AR.Drone 5

3 Introduction 7

4 PID Controller - Theory Involved 8
4.1 The Proportional (P) Term . . . 8
4.2 The Differential (D) Term . . . 9
4.3 The Integral (I) Term . . . 9

5 Implementation 10
5.1 Coordinate Frame Transformation . . . 10
5.2 Control Equation . . . 11
5.3 Tuning the PID Controller . . . 11
5.3.1 Tuning of Linear Velocities . . . 12
5.3.2 Tuning for Angular Velocity . . . 13

6 Trajectory Following 15
6.1 Experiments . . . 15
6.1.1 Inclined Circle Trajectory . . . 15
6.1.2 Figure 8 Trajectory . . . 15
6.2 Observations and Inferences . . . 16

7 Conclusion 19

II Visual Odometry 21

8 Introduction 23

9 Theory Involved 24
9.1 The Pinhole Camera Model . . . 24
9.2 Reprojection Error . . . 26
9.3 The Perspective-n-Point Problem . . . 26

9.3.1 Direct Linear Transformation . . . 27
9.3.2 Robust Estimation: Dealing with Outliers . . . 27
9.3.3 Obtaining Camera Pose from the Projection Matrix . . . 27
9.4 Epipolar Geometry and the Fundamental Matrix . . . 28
9.4.1 Epipolar Constraints . . . 28
9.4.2 Triangulation . . . 28
9.4.3 The Fundamental Matrix . . . 29

10 Algorithm **30**
10.1 Matching Initial Patches . . . 31
10.2 Tracking Features across Images . . . 31
10.2.1 Lucas-Kanade Method . . . 32
10.3 Making up for Lost Features . . . 33
10.3.1 Feature Detection using SIFT . . . 33
10.3.2 Keypoint Matching . . . 35

11 Algorithm Tuning **37**
11.1 The Dataset . . . 37
11.2 Algorithm Parameters for the Dataset . . . 38

12 The 'Reprojection Filter' Technique **39**
12.1 Experiments and Results . . . 39
12.2 Comparison With the Winning Code of ISMAR 2015 . . . 41

13 Conclusion and Future Work **43**

References **45**

A Additional Notes **49**
A.1 Camera Calibration . . . 49
A.2 Levenberg-Marquardt Optimization . . . 50

B Code Repositories **51**

C Statement of Information Search Strategy **52**

D Mini-Project Declaration **53**

List of Figures

2.1 The AR.Drone used for the Project. The 5 Spherical Markers fitted for making it detectable to the Motion Capture System can be seen on its hull. . . . 2
2.2 Image of the Project Workspace. Some of the Oqus cameras of the Qualisys Motion Capture System can be seen around the Workspace. . . . 3

4.1 Block Diagram of a PID Controller . . . 8

5.1 The World and Local Coordinate Frames as seen in the QTM Software. . . . 10
5.2 Effect of Different Values of K_p on Change in Error . . . 12
5.3 Effect of Different Values of K_d on Change in Error . . . 12
5.4 Effect of Different Values of K_i on Change in Error . . . 12
5.5 Linear and Angular Drifts Observed when the PID Controller was used to move the Drone a Distance of 1m along the y Direction . . . 13
5.6 Performance of PID Controller for Linear Velocity in 3D Space . . . 13
5.7 Performance of the tuned Controller for Angular Velocities . . . 14
5.8 Drift in Position along the 3 Axes due to Rotation . . . 14

6.1 Circle Trajectory Followed Using 'move&turn' Method . . . 16
6.2 Circle Trajectory Followed Using 'reach&turn' Method . . . 16
6.3 Mean Error at Each Waypoint for the Circular Trajectory . . . 16
6.4 Figure-8 Trajectory Followed Using 'move&turn' Method . . . 16
6.5 Figure-8 Trajectory Followed Using 'reach&turn' Method . . . 16
6.6 Mean Error at Each Waypoint for the Figure-8 Trajectory . . . 17

9.1 The Pinhole Camera Model . . . 24
9.2 Effect of Intrinsic and Extrinsic Parameters on a 3D Point . . . 26
9.3 Epipolar Geometry . . . 28

10.1 SIFT Image Pyramid . . . 34
10.2 SIFT Descriptor Vector Formulation . . . 35

11.1 World Coordinate System for the Dataset . . . 37
11.2 Effect of the Error Threshold of Lucas-Kanade Optical Flow Method on the Mean Reprojection Error at the end of the Algorithm . . . 38

12.1 Mean Reprojection Error per Frame for the 2 Methods . . . 40
12.2 Number of Features Tracked per Frame for the Two Methods . . . 40
12.3 Two Views of the Trajectory Estimated Using the 'Reprojection Filter' Technique . . . 41
12.4 Two Views of the Trajectory Estimated Without Using the 'Reprojection Filter' Technique 41
12.5 A Frame from the Video Comparing the performance of the Developed VO Code with that of the Winning Code of ISMAR 2015 Tracking Competition . . . 42

List of Tables

5.1 Optimum Control Parameter Values for the PID Controller 14

6.1 Observations for the Inclined Circle Trajectory Experiment 17
6.2 Observations for the Figure-8 Trajectory Experiment 17

11.1 Optimum VO Parameter Values for the ISMAR Dataset 38

Chapter 1

Introduction

Aerial robotics is a fast-growing field, and multirotor aircraft such as the quadrotor are rapidly growing in popularity. Quadcopters have in fact become a standard platform for robotics research all around the world due to some of its unique advantages such as vertical take-off and landing. One of the most popular topics among researchers is the autonomous flight of these vehicles. Simultaneous localization and mapping (SLAM) [1], and image-based visual servoing (IBVS) [2, 3] are techniques that have been studied and used to achieve this goal. The literatures mentioned use camera as the main sensor for perception and control as GPS may not be suitable for autonomous navigation, owing to the fact that they do not operate well indoors or in cluttered areas. Another method usually adopted for automating robot navigation is by using external off-board sensors such as a motion capture system [4, 5], or by collaboration with another cooperative robot [6].

The initial aim of the project was to develop a high-level PID controller to control the position and orientation (yaw) of a Parrot AR.Drone 2.0 quadcopter, using a Qualisys Motion Capture System as the sensor. This was to be achieved by observing the pose of the drone from the motion capture system and guiding it to the destination pose (position and orientation) using the developed PID controller. A PID controller for the task was developed, its parameters tuned, and its performance experimentally tested. The controller was applied for the task of trajectory following using two methods of navigation that were developed. The effectiveness of the two methods in navigating a circular and a 'figure-8' trajectory have been analysed and compared. The PID control theory, its implementation on the AR.Drone, and the experiments involved make up the first part of this report.

An additional part – 'Visual Odometry' – was implemented and has been included in the report as the second part of the project, exceeding the initial aims presented in the 'Mini-Project Declaration Form' (Appendix D). The topic was examined with the intention of implementing 'Visual Servo Control' on the AR.Drone which, along with the PID controller developed, was to be the first step for developing a visual SLAM system on the aerial robot (proposed topic for the summer project).

Visual odometry (VO) is the process of estimating the path moved by a vehicle using just the sequence of images captured by an on-board camera. Given a few feature patches and their positions in the 3D world frame, the developed VO algorithm estimates the camera pose at each image frame using the 3D-2D correspondences of tracked features. Once all the images in the sequence are analysed, the algorithm plots the estimated trajectory of the camera in the world frame. The total mean reprojection error was used as the measure of performance.

Since the best parameters of the VO algorithm depends on the task at hand, a consistent dataset was required for its testing and development. In this project, the dataset provided by the International Symposium on Mixed and Augmented Reality (ISMAR) for their Offline Tracking Competition of 2015, was used. The various parameters of the algorithm were experimentally tuned to minimise the reprojection error for the task. A method to improve the performance of the algorithm by discarding features with high reprojection error, called the 'Reprojection Filter Technique' was developed and tested. The performance of the algorithm with and without the implementation of this technique have been experimentally analysed and their results compared. Finally, the performance of the algorithm was compared with the code that had won the ISMAR 2015 Tracking Competition.

The report is organised in the following way: Chapter 2 gives an overview about the task environment and the hardware involved in the project. The remainder of the report is divided into two parts: 'PID Controller' and 'Visual Odometry', containing the theory and experimentations involved in their implementations.

Chapter 2

Hardware and Workspace

The main platform used for this project was the Parrot AR.Drone 2.0 quadcopter. Both the PID controller and the visual odometry algorithm were developed with the intention of implementing them on the AR.Drone. Another integral part of the project, especially for the development of the PID controller, was the Qualisys Motion Tracking System. The data from the motion tracking system was used as the ground truth for most of the experiments. Moreover, the PID controller developed depends intensively on the information obtained from the motion capture system as it uses this data for correcting the pose of the drone. In addition to the drone and the motion capture system, two computers were used for the processing and running of the codes for the project.

2.1 The Drone - Parrot AR.Drone 2.0

The quadrotor vehicle or quadcopter, is a very simple machine. It consists of four individual rotors attached to a rigid cross airframe, each rotor rotating in the direction opposite to that of its adjacent rotors to counter the torque on the body of the vehicle. Control of a quadrotor is achieved by differential control of the thrust generated by each rotor. Pitch, roll, and thrust (lift and linear motion) control is straightforward to conceptualize. Yaw control is obtained by adjusting the average speed of the clockwise and anticlockwise rotating rotors [7].

The AR.Drone 2.0 is a relatively cheap quadcopter which has a wide range of on-board sensors which can be used for various tasks. It has a size of 55 cm from rotor-tip to rotor-tip, and weighs less than 500 grams. It can be controlled using various android and iOS applications.

Figure 2.1: The AR.Drone used for the Project. The 5 Spherical Markers fitted for making it detectable to the Motion Capture System can be seen on its hull.

The AR.Drone 2.0 has a motherboard containing the main processing unit (an ARM Cortex-A8 running at 1 GHz), two cameras, and a Wi-Fi module[8]. One of the cameras is oriented vertically, pointing to the ground. It has a frame rate of 60 frames per second (fps) and a resolution of 320 $\times$ 240 pixels, and is used for estimating the horizontal speed of the drone. The second camera is pointing forwards and can capture images of 720p resolution at 30 fps. A second board, the navigation board, contains all the sensors necessary for the state estimation of the quadrotor. The sensors include GPS, an IMU consisting of a gyroscope and accelerometer, pressure sensors, ultrasound sensors, and a barometric sensor.

Due to the availability of a large range of sensor data, the AR.Drone has been successfully used for various tasks such as object following, position stabilization and autonomous navigation [9]. For this project, the ROS (Robot Operating System) driver for the AR.Drone, '*ardrone_autonomy*', was used for controlling the drone using ROS commands [10]. The driver can also be used to access various information from the drone sensors including images from cameras, IMU data, linear and angular velocities, and the current state of drone (landing, take off etc.). Being equipped with onboard sensors and controllers, the AR.Drone can stabilize itself while flying and can hover on a spot, awaiting the next command. The low-level controllers within the drone are developed from the dynamic model of the drone, which makes it able to counter most of the external dynamic forces such as wind or gravity.

2.2 Base Station

The base station consists of the motion capture system and the two computers used for running the algorithms for the project.

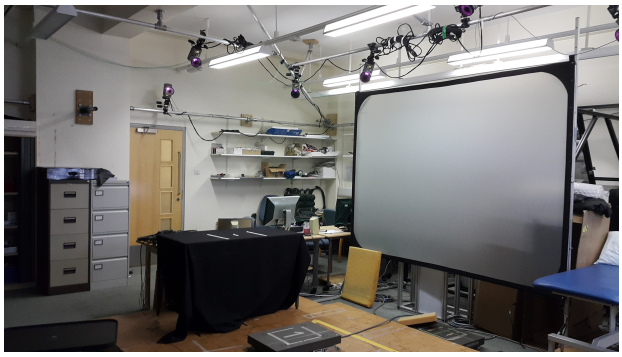


Figure 2.2: Image of the Project Workspace. Some of the Oqus cameras of the Qualisys Motion Capture System can be seen around the Workspace.

2.2.1 The Ground Truth System

The data obtained from the Qualisys Motion Capture System is used as the ground truth for the project. It uses 12 Oqus cameras placed around the workspace, which can capture frames of 1.3 megapixels at the rate of upto 10,000 fps [11]. These cameras can detect specially designed markers (polystyrene hemispheres covered in a special retro-reflective tape). Five small spherical markers were fixed on the AR.Drone for making it detectable to the motion capture system (Figure 2.1).

Qualisys provides a user-interface software called *Qualisys Track Manager* (QTM) which makes it convenient and straightforward to collect data from the motion capture system. It can provide 6 degrees-of-freedom (6DoF) data (3 translation and 3 rotation) of the tracked body. It was also used to visualise the movement of the body in real-time.

The QTM software was run on an i5-2500 CPU @ 3.6 GHz, 10 GB RAM computer running Windows 7. The software was used to define a coordinate frame in the workspace (which was used as the *world frame* for the project) by fixing the origin at a point on the ground of the workspace, and specifying the directions of the axes. The markers on the AR.Drone were used to define another coordinate frame at the centre of the drone (the *local frame*). The motion capture system could now define the position and orientation of the local frame with respect to the world frame.

The 6DoF data from QTM was streamed real-time into MATLAB using *QTM Connect*, another Qualisys supporting software. A MATLAB code was used to transmit the position (as a 3D point vector) and orientation (as a rotation matrix) of the local frame in the world frame, wirelessly to the Control Station (section 2.2.2). The Qualysis Motion Capture System along with the computer running the QTM software and the MATLAB code together formed the Ground Truth System for this project.

2.2.2 The Control Station

An i7-5500U CPU @ 2.40GHz, 6GB RAM laptop running Ubuntu 16.04 was used as the Control Station. It ran all the codes required for the PID controller as well as the visual odometry algorithm. It was wirelessly connected to both the AR.Drone and the Ground Truth System.

A python code was used by the Control Station to parse the data transmitted by the Ground Truth System. It converts the 6DoF information obtained into data that can be used by the controller algorithm to guide the drone to the required pose.

The Control Station was connected to the AR.Drone to wirelessly control its motion using the PID controller developed. It was also possible to obtain the images from the two on-board cameras on the drone using the wireless connection. Additionally, the wireless connection between the drone and the control station was necessary to manually override the drone using ROS commands in case of any potential emergency situations.

Part I

PID Controller for the AR.Drone

Chapter 3

Introduction

A quadcopter has six degrees of freedom, but only four controls (the velocities of the four motors). Therefore, the system is under-actuated, and the remaining degrees of freedom corresponding to the translational velocity in the plane of the quadcopter must be controlled through the system dynamics [7]. However, the dynamics of the quadrotor are highly non-linear and several uncertainties are encountered during its missions, thereby making its flight control a challenging venture [12]. Various controllers for simple [13] and complicated [14, 5, 7] dynamic models for a quadcopter have been developed. The simple models usually linearise the model around hover conditions, and are therefore stable only under reasonably small roll and pitch angles. More complicated models, which try to account for the full state space of a quadcopter, were able to achieve more aggressive and complicated manoeuvrability of the vehicles [5, 15].

Performance of the controller could be improved by using the on-board sensors for state and pose estimation. Controllers designed by modelling the quadcopter dynamics usually consider these information for more accurate state estimations. A framework for autonomous flying of the AR.Drone using just the on-board sensors have been shown in [8], making the system independent of any base station or remote control.

All the controllers mentioned above achieve the stability and control of a vehicle by modelling its dynamics. There is no abstraction and the motion of the quadcopter is controlled directly by varying the voltage supplied to the individual motors. These types of controllers are called *low-level controllers*. For this project, this is done autonomously by the on-board processor of the AR.Drone, using the informations from the various sensors to stabilize the drone.

On the other hand, *high-level controllers* are used to achieve some high-level functionalities on a robot (such as situation awareness) by making use of the low level controllers. There is a high level of abstraction from the inner workings of the actual control. It is used to perform various functions needed to carry out a specific operational task. The task is achieved by making use of some sensor information for giving commands to the low level controllers for the required motion. Vision based controllers using on-board cameras have been developed and tested for various tasks [16, 2, 17]. Drones which use GPS-based controllers to fly outdoors are available as commercial products [18].

For this project, a high-level PID controller was developed to change and control the pose of the AR.Drone using the data obtained from a Qualisys Motion Capture System. The controller makes use of the inbuilt low-level controller for stabilizing the drone in the air, and for guiding it to the destination pose.

The remainder of this part of the report is organized as follows: Chapter 4 briefs the basic theory of PID control. Chapter 5 explains the implementation of the PID controller on the AR.Drone and the tuning of its control variables. Chapter 6 contains the experiments and results of comparing two navigation methods that were developed for trajectory following. Part I is concluded in Chapter 7 with a review and assessment of the controller developed.

Chapter 4

PID Controller - Theory Involved

A proportional-integral-derivative (PID) controller is a feedback control mechanism which is very commonly used in industries due to its simplicity and effectiveness. Most industrial controllers are still implemented based around PID algorithms, as no other controllers match the simplicity, clear functionality, applicability, and ease of use offered by the PID controller [19]. In [13], the author compared PID controller with a Linear-Quadratic controller, and proved that the former was able to perform better even under minor perturbations in the system. Various types of controllers are compared and their advantages and disadvantages explored in [20]. The authors concludes by claiming that no single algorithm presented the best required results. They proposed a hybrid control scheme that have the best features of the different controllers. However, such hybrid systems would make the response slower, and do not guarantee good performance.

A PID controller uses a feedback mechanism which continuously calculates an error value $e(t)$ as the difference between a desired setpoint (desired value of the control variable) and a measured process variable, and applies a correction based on the proportional, integral, and derivative terms.

Figure 4.1 shows the block diagram of a PID controlled system. The plant feedback is subtracted from the command signal r to generate the error signal $e(t)$ which is to be minimized. The control signal (u) to the plant is adjusted to a new value so as to minimize this error. The control signal is determined by:

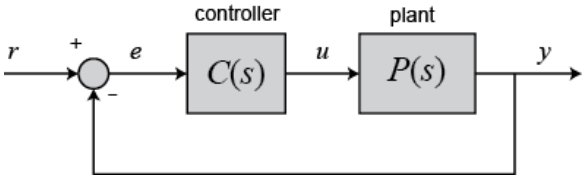


Figure 4.1: Block Diagram of a PID Controller
Source: Control Tutorials for MATLAB and Simulink,*'Introduction: PID Controller Design'*
Retrieved from: `http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID`

$$u(t) = K_p e(t) + K_i \int e(t)\,dt + K_d \frac{de(t)}{dt} \tag{4.1}$$

where K_p, K_i and K_d are non-negative coefficients for the proportional, integral, and derivative terms respectively. They are also called gains .The error signal (e) is sent to the PID controller, which computes both the derivative and the integral of this error signal.

Each gain term in the control equation (4.1) has specific functionalities and the behavior of the system changes depending on their values [21].

4.1 The Proportional (P) Term

The proportional term in the PID controller is responsible for providing an overall control action proportional to the error signal. If the other gains are assumed to be zero, we get the proportional controller which is the error signal (e) multiplied by a constant (K_p). The value of K_p determines how quickly the error reduces (*response rate*). It accounts for present values of the error. Therefore, if the error is large and positive, the control output will also be large and positive, which in turn reduces the error. The proportional term is given by:

$$P_{out} = K_p e(t)$$

However, for most systems using just the proportional controller, the measured process variable will overshoot past the desired setpoint. This is because all real plants have some inherent delay in their

system response, i.e. they do not give the output for a specific input instantaneously. Therefore even when the process variable actually reaches the desired value, the measured error will not be exactly zero. This results in overshoot. In the next instant, however, the sign of the error value changes, bringing the process variable back to the setpoint. Again, it overshoots due to the plant delay, this time in the opposite direction. This results in an oscillatory behavior of the system variable. It is obvious from the control equation that the overshoot and oscillation amplitude will be more pronounced for higher values of K_p.

Therefore, there is a need for some kind of 'damping' which can slow down the response when the process variable approaches the desired value. This can be achieved using the derivative term.

4.2 The Differential (D) Term

The derivative term of the PID control equation improves the transient response of the system by taking into account the current rate of change of the error e. It is obtained by determining the slope of the error over time and multiplying this rate of change by the derivative gain K_d. It is given by:

$$D_{out} = K_d \frac{de(t)}{dt}$$

This derivative action predicts the system behavior by considering the rate of change of the error, and thus improves the stability and settling time of the system [22]. Therefore, including the derivative term in the control equation (4.1) can result in decreasing the response rate as the error approaches zero. Proper tuning of the P and D terms of the PID controller can theoretically result in reducing the error to zero over time.

The final difference between the process variable and setpoint is called the steady state error. The desired value of the steady state error is zero. However, if there is a bias or an inherent offset in the system, the steady state error will not be zero, i.e. the process variable will never reach the desired setpoint value. This is because the effect of the proportional term, which is to drive the error to zero, is diminished by the bias in the system. It will stop at a value where the process variable is equal to the effect of the bias. Therefore, if there is some bias which continuously influences the control variable, it can never reach the setpoint value. In this case, another parameter that can detect the presence of a system bias, and can counteract its effect, is required. This is the main role of the integral term.

4.3 The Integral (I) Term

The integral term in the PID control equation (4.1) adjusts the controller's output in accordance with both the size of the deviation from setpoint and the time it lasts. While the proportional term considers the current size of e only at the time of the controller calculation, the integral term considers the history of the error, or how long and how far the measured process variable has been from the set point over time [22]. The integral term is given by:

$$I_{out} = K_i \int e(t)\, dt$$

The integral response will increase continuously over time unless the error is zero. The result is that even a small error term will cause the integral part to increase slowly. The effect of the integral term is to drive the steady state error to zero. It causes the controller output to move in the direction of the setpoint by an amount equal to the difference between the loop output when setpoint is equal to control point, and the actual loop output caused by offset. So it eliminates the residual steady-state error that occurs with a pure proportional controller.

However, due to its accumulative behaviour which keeps summing up the errors as long as there is a steady state error, it is mathematically possible for the integral term to grow very large. In cases where the setpoint is a value that is impossible to reach, the integral term will keep increasing to a value that would become nonsensical. The result is called *integral windup* [21]. This can cause overshooting and further build-up of the integral term until the accumulated error is unwound (offset by the errors in the opposite direction).

Chapter 5

Implementation

The PID controller developed for this project works by minimising the error between its current pose and the desired pose. This involved converting the position and orientation of the destination pose in the world coordinate frame to the local frame of the drone. The controller then corrects the pose of the local frame to align with the destination pose.

5.1 Coordinate Frame Transformation

The coordinate frames used for defining positions and orientations in this project follow the right-hand system. The two coordinate frames used for this part of the project are the world coordinate frame and the local frame. The world frame is defined with its origin at a point on the floor of the workspace and its z axis pointing vertically upwards. The local frame is the dynamic coordinate frame associated with the drone. It has its origin at the centre of the drone, with the x axis pointing to the front of the drone, and the z axis pointing upwards.

Coordinate transformation is used in this project to transform a point in the world frame to the local frame so that the controller can be used to directly minimise the distance between the local frame and the desired position which is defined in the world frame (since it is a stationary point in the world frame). This coordinate transformation was achieved using the ROS '*tf*' package [23]. It lets the user keep track of multiple coordinate frames over time, and enables the transformation of points between any two coordinate frames.

The Ground Truth System provides the position and rotation of the local frame with respect to the world frame. It gives the position of the local frame origin as a 3D point vector and the local frame orientation as a rotation vector. If R_L^w is the rotation matrix denoting the rotational part of the coordinate transformation from the world frame to the drone's local frame, and T is the position vector of the local frame origin in the world frame, then a point p_L in the local frame can be transformed to the world frame using the transformation equation:

$$p_w = R_L^w p_L + T$$

where p_w is the position of the point in the world frame. This relation can be used to transform a point in the world frame back to the local frame by inversion.

$$p_L = (R_L^w)^{-1}(p_w - T) = (R_L^w)^T(p_w - T)$$

($R^{-1} = R^T$ since rotation matrices are orthogonal. Note that $(R_L^w)^{-1}$ is in fact R_w^L, the rotation matrix describing the coordinate transformation from the local frame to the world frame.)

Therefore, the position of the destination point defined in the world frame can be obtained in the local frame by using coordinate transformations in each iteration of the control.

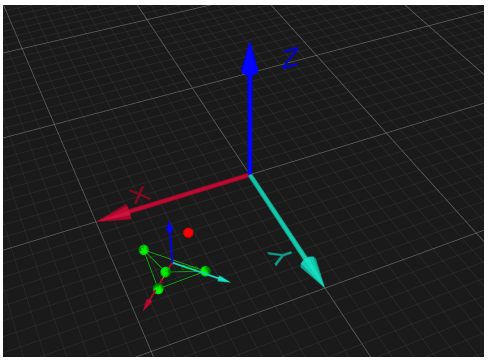


Figure 5.1: The World and Local Coordinate Frames as seen in the QTM Software.
The larger coordinate frame indicates the world coordinate system and the smaller frame shows the local coordinate frame associated with the drone. The Spherical Markers on the Drone are visualised as the red and green dots.

5.2 Control Equation

The ROS driver for the AR.Drone, '*ardrone_autonomy*' provides 8 commands for controlling the motion of the drone through ROS commands – 2 each (positive and negative) for the linear motion velocities along the 3 axes (of the local frame), and 2 for the angular velocities of yaw (rotation about the vertical axis (z) of the drone). Therefore, the AR.Drone has four degrees of freedom – 3 linear and 1 angular – which can be controlled using the 8 ROS commands (two for each DoF). To implement a PID controller for high-level position control of the drone in the workspace, it was these velocities which were passed as ROS commands which needed to be controlled.

The data received from the Ground Truth System is passed to the Control Station, where the PID controller is implemented. The Control Station uses the data to obtain the current position and orientation of the drone in the workspace. The destination pose (position and orientation) can be assigned to be the setpoint for the controller. The destination orientation is the angle between the x axis of the world frame and the x axis of the local frame (about the global z axis). The position part of the destination pose is a 3D point in the world frame. The point has to be transformed to the local coordinate frame of the drone at each iteration of the control. This way, the ROS commands for the drone velocity (which are given in the local frame coordinate system) can be used to directly minimise the error between the current pose of the local frame and the destination pose.

Once the destination point is defined in the local frame, the error $e(t)$ of the control equation can be defined as a 4-dimensional vector containing the position of the destination point in the local frame, and the angle between the orientation of the local x axis and the desired orientation (the drone is assumed to be horizontal at all times), i.e. $e(t) = \begin{bmatrix} p_{x_L} & p_{y_L} & p_{z_L} & (\theta_d - \theta_{curr}) \end{bmatrix}^T$, where p_{x_L}, p_{y_L} and p_{z_L} are the position of the destination point along the x, y, and z coordinates of the local coordinate frame, θ_d is the destination orientation, and θ_{curr} is the current orientation (angle between the local x axis and the global x axis about the global z axis).

The implemented PID controller uses four control equations, one for minimising each of the 4 errors in the error vector. Each of the control equations controlled one of the four DoFs of the quadcopter velocities (3 linear and 1 angular). For each error value, the differential part of the control equation (4.1) was obtained by approximating the differentiation of the error at each iteration as:

$$\frac{de(t)}{dt} \approx \frac{e(n) - e(n-1)}{t_n - t_{n-1}}$$

where n is the iteration number of the control loop, and t_n is the clock time at iteration n.

Similarly, the integral part of the control equation (4.1) was approximated to the Riemann sum as follows:

$$\int e(t)\,dt \approx \sum_n e(t_n)[t_n - t_{n-1}]$$

So the general form of the control equations for each error term in the error vector at the n^{th} iteration of the control loop can be given by:

$$u(n) = K_p e(n) + K_i \sum_n e(t_n)[t_n - t_{n-1}] + K_d \frac{e(n) - e(n-1)}{t_n - t_{n-1}}$$

Now that the control equation for the PID controller is defined, suitable values for the control parameters K_p, K_i, and K_d has to be obtained.

5.3 Tuning the PID Controller

Tuning is the process of adjusting the PID control parameters to get the desired stable control response. There are various sophisticated techniques that use various algorithms and model to self-tune the parameters [24]. These methods are however more complicated than manual tuning and requires more computations. The manual tuning method based on trial-and-error was adopted for this project. It had the advantage of simplicity and required no math.

For small, low torque motors with less gearing (such as the ones in a quadcoptor), manual tuning method to get a good baseline tune is to probe it's response to a disturbance [25]. The steps followed for tuning the PID controller for this project were as follows:

1. Initially all the gains were set to zero.

2. The proportional gain K_p was increased until the response to the disturbance was a steady oscillation.

3. The differential gain K_d was increased until the oscillations went away (i.e. they were critically damped).

4. The integral gain K_i was increased until it brought the steady state error close to zero.

The tuning of four values of K_p, K_d, and K_i are required, one for each of the four DoFs of the AR.Drone. However, the linear motions in all three directions (along the local frame axes) were observed to be performing similarly for similar values. So the PID was tuned only for the motion along one axis (y). However, the response was tested experimentally with the final optimum values for all three linear motions. The control parameters for linear and angular velocities of the drone were tuned separately.

5.3.1 Tuning of Linear Velocities

The control parameters for the linear velocity were tuned by making the drone hover in the air at a point, and then observing the error as it moved 1m along the local y axis.

Proportional Gain K_p: Figure 5.2 shows the change in error over time for different values of K_p. A small steady oscillation was observed for $K_p = 0.05$, which could be corrected by damping. Higher values gave oscillations of higher frequency and amplitude while smaller values took more time to reach zero error. Therefore, 0.05 was chosen as the best value for the proportional gain.

Derivative Gain K_d: The effect of different values of K_d on the change in error over time is shown in Figure 5.3. Smaller values did not give enough damping while larger values resulted in taking more time to reach steady state. A compromise between time required to reach steady state and inadequate damping had to be reached, and 0.2 was chosen as the optimum value for K_d.

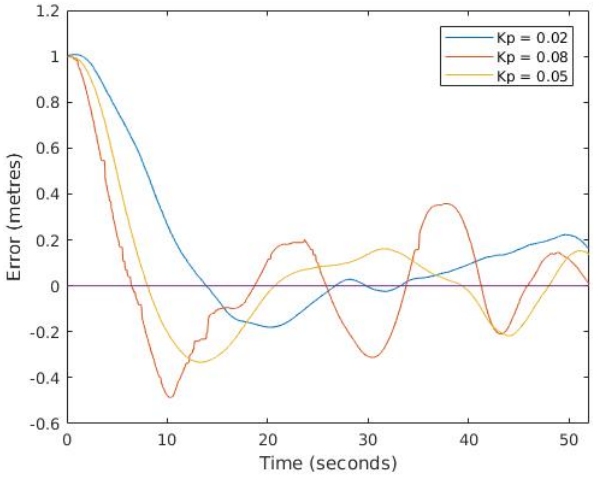


Figure 5.2: Effect of Different Values of K_p on Change in Error

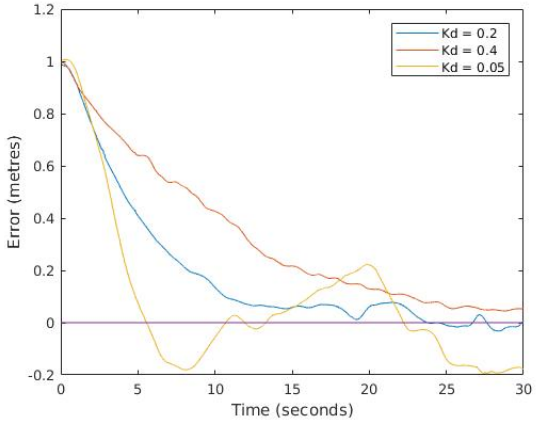


Figure 5.3: Effect of Different Values of K_d on Change in Error

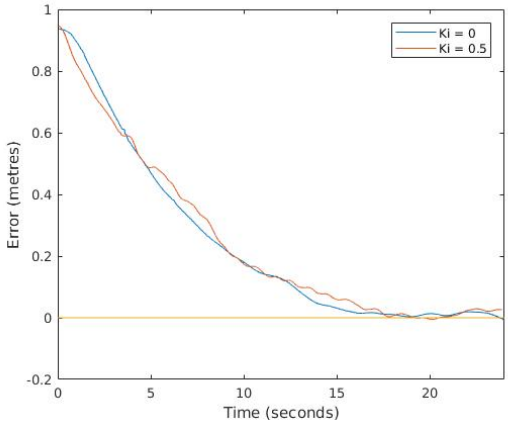
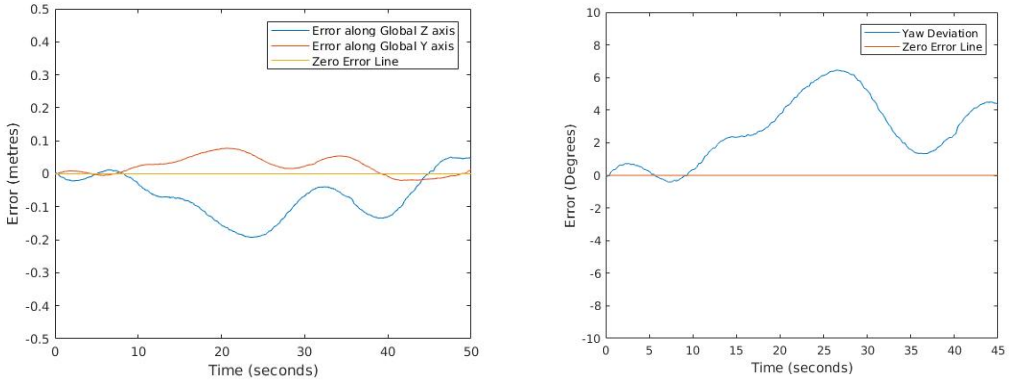


Figure 5.4: Effect of Different Values of K_i on Change in Error

Integral Gain K_i: It was observed that adding the integral term to the controller did not have much effect on its performance (Figure 5.4). In fact, increasing the value of K_i seemed to make the system unstable. This could be because the integral term is used to make the system reach steady state in case of some inherent bias in the system. The ineffectiveness of the integral term in the controller could mean that there is no system bias in the drone's low-level control system. Due to this, and considering the possibility of integral windup, the integral gain was set to zero.

Checking for Drift: Once the control parameters were tuned, the same experiment was conducted using the final values to determine how much the orientation changes due to linear motion, and also to determine the offsets induced along the other axes. This was done by aligning the local and world coordinate frame orientations and making the drone move a distance of 1m along the direction of its y axis. The change in position along the x and z directions of the global frame, and the change in orientation were observed (Figures 5.5a and 5.5b).



(a) Linear Drift

(b) Angular Drift

Figure 5.5: Linear and Angular Drifts Observed when the PID Controller was used to move the Drone a Distance of 1m along the y Direction

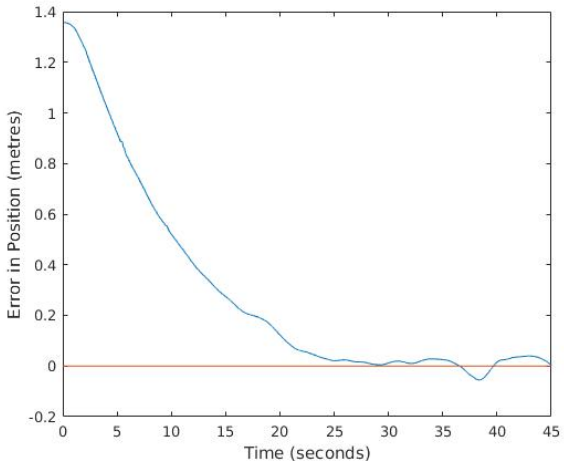


Figure 5.6: Performance of PID Controller for Linear Velocity in 3D Space

The absolute linear errors along both the directions were found to be less than 20 cm, with a mean of 9.23 cm. However, the drift along the z direction seemed to be slightly higher. This was expected as the low level controller uses the bottom camera and ultra sound sensors to estimate the height of the drone, and tries to maintain constant height using their readings. The noise in the sensor readings and other dynamic factors could be the reason for the slightly more pronounced drift along the z axis. This does not happen when the PID controller is used for controlling the position along the z axis also. The drifts in the orientation was observed to be an average of 2.34°, with the maximum error being 6.44°.

Testing Performance for 3D motion: The values obtained for the control parameters were tested for all three linear motions (along x, y and z of the local frame) by making the drone move in the 3D workspace. It was positioned at a point (0.7,0.7,0.7) in world coordinates of the workspace, and made to move to (1.5,1.5,1.5). Figure 5.6 shows the change in the Euclidean error in position with time. The system reached steady state in 24.3 seconds, with a mean steady state error of 4.29 cm.

5.3.2 Tuning for Angular Velocity

The control parameters for the angular velocity control were tuned by making the drone hover in the air, and then rotate 30 degrees in the spot. The change in error over time was observed with the initial

angular error of 30 degrees.

The tuning was done by following the same steps. Initially K_p value was tuned by keeping all others constant. It was observed that neither K_d nor K_i was required to bring the error to zero. This was because the time required to reach the steady state was very low (less than 4 seconds), when the initial error was 30 degrees. Figure 5.7 shows the performance of the controller when $K_p = 5$ and $K_d = K_i = 0$, which were selected as the best values.

Once the control parameters were tuned, the same experiment was conducted using the final values to determine the error in position (linear drift) due to the rotation of the drone. The drone was started with an initial yaw of 30 degrees (with respect to the global frame). It was then made to rotate in the spot to bring the yaw deviation down to zero. The change in the position of the drone due to this rotation was observed using the motion capture system. The drifts observed are shown in Figure 5.8.

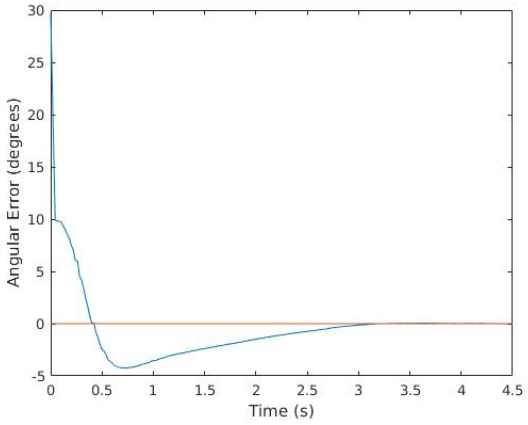


Figure 5.7: Performance of the tuned Controller for Angular Velocities

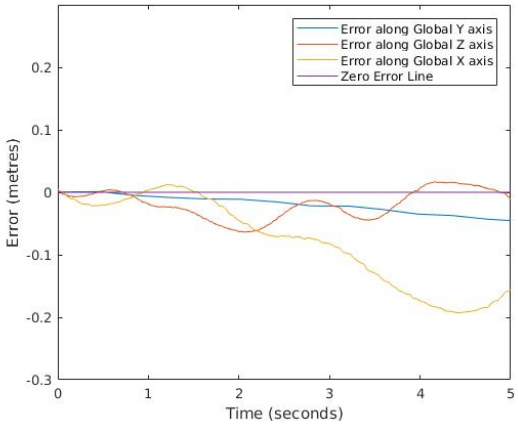


Figure 5.8: Drift in Position along the 3 Axes due to Rotation

It can be seen that the drift in the x direction is more pronounced than the others. This can be attributed to the effect of the rotational torque. The torque produced and the moment of inertia due to the rotation of the drone imparts a displacement along the direction of rotation. Since the drone is rotating away from the y axis and towards the x axis, the effect of the torque is to move the drone in the direction of rotation, i.e. away from the y axis, which results in the displacement along the x direction.

The optimum values for the control variables of the developed PID controller are shown in Table 5.1.

Control Parameter	Optimum Value	
	Linear Velocity Controller	Angular Velocity Controller
K_p	0.05	5
K_d	0.2	0
K_i	0	0

Table 5.1: Optimum Control Parameter Values for the PID Controller

Chapter 6

Trajectory Following

The developed PID controller was used to make the drone follow simple trajectories. Two different navigation methods were developed to test the performance of the controller while following the trajectory. The first method involved the drone correcting both the errors in its position and the error in the orientation simultaneously. This meant that the drone moved and turned at the same time, and the method was named 'move&turn'. The second method, named 'reach&turn', required the drone to first correct its position, and then its orientation once the destination position is reached.

A trajectory in this experiment is defined as a sequence of destination poses that the drone should attain one after the other. If the drone stays for 3 seconds with the euclidean distance between the current position and the desired position (waypoint) less than 10cm, and the orientation error less than 0.15 radians (approx. 8.5°), the destination pose is considered to be attained. The error at each of the trajectory 'waypoints' and the total time taken for successfully navigating the trajectory were noted for the two different navigation methods. The experiments and results of the AR.Drone following an inclined circular trajectory and a figure-8-shaped trajectory using the two methods are presented in this report. These shapes offer a good test for the performance of the controller as one is a very simple shape, while the other is a relatively more complex shape.

6.1 Experiments

6.1.1 Inclined Circle Trajectory

Twenty points on a circle of radius 70 cm inclined 30 degrees to the horizontal were defined as waypoints for the drone to follow. The centre of the circle was at the coordinates $(0.7m, 0.7m, 0.7m)$ in the world frame. The set point orientation at each point was such that it faces towards the next waypoint in the trajectory. The drone was made to hover first, and the trajectory follower algorithm was activated. This was done for both the 'move&turn' and 'reach&turn' methods. The experiment was repeated 3 times for for data reliability.

Figures 6.1 and 6.2 show the best observed trajectories for the two methods, out of the three trials. They show the actual circular path and the path followed by the AR.Drone, for 'move&turn' and 'reach&turn' methods respectively. The waypoints are shown as red crosses.

Figure 6.3a shows the mean Euclidean errors in position at each waypoint for the two methods, while Figure 6.3b shows the mean errors in orientation.

6.1.2 Figure 8 Trajectory

20 points on a horizontal figure-8 path (created using the equation for the lemniscate of Gerono [26]) were defined as waypoints for the drone to follow. The centre of the figure (intersection) was at the coordinates $(0.7m, 0.7m, 0.7m)$ in the world frame. The set point orientation at each waypoint was such that it faces towards the next waypoint in the trajectory.

Figures 6.4 and 6.5 show the actual path and the path followed by the AR.Drone in one of the trials, for the 'move&turn' and 'reach&turn' methods respectively.

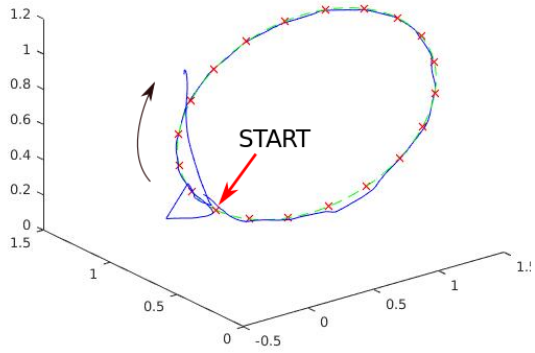


Figure 6.1: Circle Trajectory Followed Using 'move&turn' Method

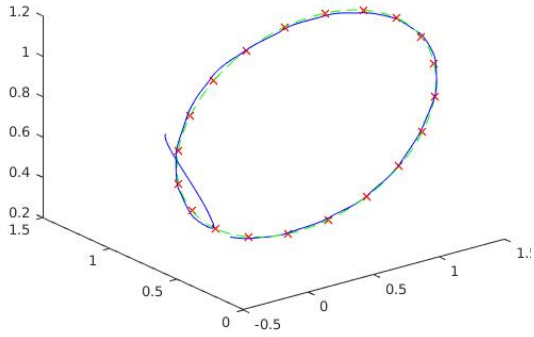
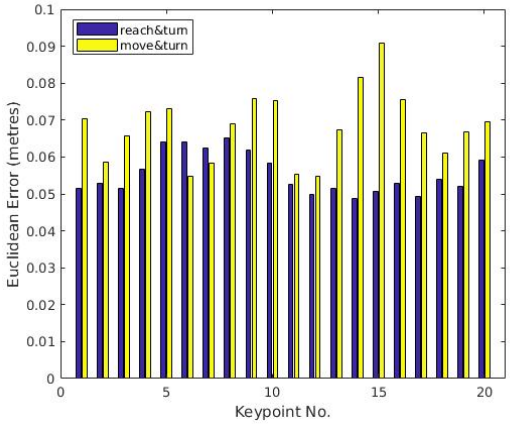
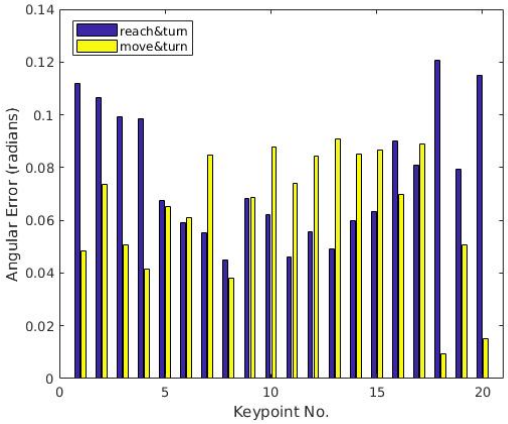


Figure 6.2: Circle Trajectory Followed Using 'reach&turn' Method

(Red crosses indicate defined destination waypoints. Green dotted line is the defined trajectory)



(a) Mean Euclidean Error in Position



(b) Mean Angular Error

Figure 6.3: Mean Error at Each Waypoint for the Circular Trajectory

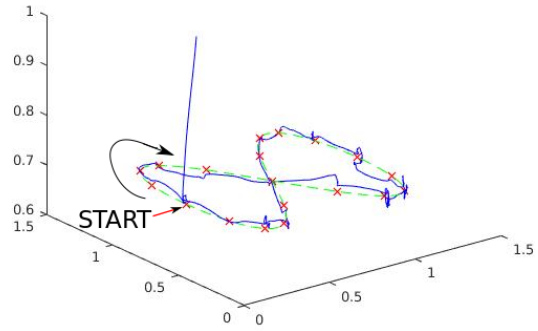


Figure 6.4: Figure-8 Trajectory Followed Using 'move&turn' Method

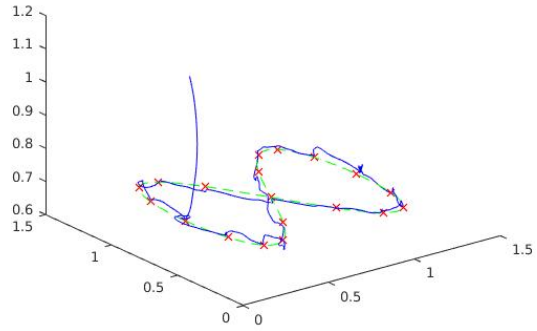


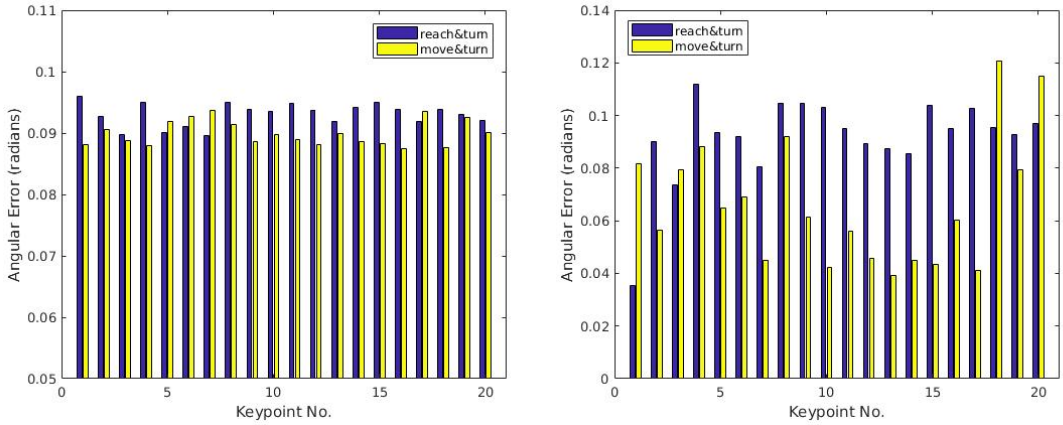
Figure 6.5: Figure-8 Trajectory Followed Using 'reach&turn' Method

(Red crosses indicate defined destination waypoints. Green dotted line is the defined trajectory)

6.2 Observations and Inferences

The observations from the trajectory following experiments and their inferences are as follows:

1. There is less deviation from the actual trajectory when using the 'reach&turn' navigation method.



(a) Mean Euclidean Error in Position

(b) Mean Angular Error

Figure 6.6: Mean Error at Each Waypoint for the Figure-8 Trajectory

This is because this method corrects the position first, and only then the orientation. On the other hand, the 'move&turn' method tries to correct both at the same time, which causes the drone to slightly drift away due to the rotation (section 5.3.2). Another reason for this action is that since the controller tries to minimise the position errors in the local frame, and the local frame is rotating, the linear velocity controller action can produce a motion in a direction that is slightly off. This is because of the delay in the system, which causes the controller to respond to the error a little late. By the time the controller reacts to the error, the orientation of the local frame changes.

The performance difference was not very prominent in the case of the figure-8 trajectory (Figure 6.6a), as the area of the path was too narrow (due to workspace constraints) for the comparatively more complicated path. The trajectory was only approximately 1.2m wide end to end, in both directions. This was a good test for the precision of the controller. The mean error was found to be much higher for both the methods while following the comparatively more complicated trajectory (Tables 6.1 and 6.2).

Table 6.1: Observations for the Inclined Circle Trajectory Experiment

Method	Mean Linear Error (m)	Mean Angular Error (m)	Mean Time (sec)
'move&turn'	0.0681	0.0762	211
'reach&turn'	0.0555	0.0766	264

Table 6.2: Observations for the Figure-8 Trajectory Experiment

Method	Mean Linear Error (m)	Mean Angular Error (m)	Mean Time (sec)
'move&turn'	0.0899	0.0662	339
'reach&turn'	0.0930	0.0917	362

2. The mean angular errors in the case of the circular trajectory were very similar for both the methods. However, in the case of the figure-8 trajectory, the 'reach&turn' method resulted in a higher mean error. This is because the orientation difference between two consecutive waypoints in the trajectory are larger compared to those in the circular trajectory. This does not affect the 'move&turn' method very much as it corrects the orientation on its way to the waypoint, and the error is already low when it reaches a waypoint.

3. In the case of the circular trajectory, it was observed that the 'reach&turn' method took a longer time to complete the trajectory compared to the 'move&turn' method (Tables 6.1 and 6.2). This is because the former makes the drone stop at each point to correct the orientation, while the latter corrects the orientation on the way to the next waypoint.

   However, in the case of the figure-8 trajectory, the time required for the 'reach&turn' method was slightly less than that for the 'move&turn' method. This is because the increased complexity of the trajectory shape resulted in more deviation from the actual path due to drift, and it took the drone more time to reach the waypoints. On the other hand, the 'reach&turn' method finds the shortest path to the next waypoint, and then corrects its orientation, which was more effective for complex trajectories.

In summary, as far as reaching a single destination point is concerned, the 'reach&turn' method is quicker as it would take the shortest path to the destination, and then correct the orientation. The 'move&turn' method, on the other hand, deviates from the shortest path due to its simultaneous orientation correction. When traversing multiple waypoints in sequence however, the performance of the methods are different. The time required by the 'move&turn' method is lesser than the time required by the 'reach&turn' method if the trajectory shape is simple. However, as the complexity of the trajectory increases, the latter can complete the task faster. It was also noticed in general that the time required for traversing the trajectories by both methods increased with the increase in trajectory complexity.

Chapter 7

Conclusion

The PID controller implemented for this project was tuned manually and its effect on linear and angular drift analysed. Two methods of navigation were implemented, one where both the position and orientation were corrected simultaneously ('move&turn'), and the other where the orientation is corrected only after the destination is reached ('reach&turn'). Depending on the complexity of the trajectory, the methods performed differently. The performance of the two methods were tested experimentally and their results presented. Both methods were shown to perform with position errors less than 10 cm and orientation errors less than 1.5 radians.

The developed PID controller can be used for following simple trajectories with position accuracy of upto 5.55 cm (Table 6.1). This is not accurate enough for any precision navigation or aggressive flight patterns. Controllers which can produce complex maneuvers at much higher speeds using data from motion capture system and accurate up to 1cm have been developed [14]. More complicated controls such as flying through narrow, vertical and horizontal openings, and perching on an inverted surface have been demonstrated using feedback controllers [5]. The precision of these controllers can be attributed to modeling the controller from scratch, i.e. without any abstraction from the inner working of the actual control. The controllers are developed from the dynamic model of the drone. In fact, the complex trajectories in [5] are achieved by iteratively refining the controller to account for errors in the dynamic model and noise in the actuators and sensors. Further, the quadcopter dynamics is highly non-linear and can be affected by various aerodynamic factors when it is not hovering. Controllers which account for such aerodynamic effects have been proven to produce better results [27]. This shows that the performance of the controller could be improved by designing it from the lower level using the dynamic model of the quadcopter.

The lack of dynamic modelling for the controller was an important factor that affected the performance of the controller in this project. The controller could also be improved by making use of the large range of sensors that are available on the AR.Drone.

Part II

Visual Odometry

Chapter 8

Introduction

Visual Odometry (VO) is the process of determining the motion of a camera by analysing just the images captured by it along the way. The term was coined by D. Nister in his landmark paper [28], chosen for its similarity to wheel odometry, which incrementally estimates the motion of a vehicle by integrating the number of turns of its wheels over time. VO operates similarly by incrementally estimating the pose of the vehicle by analyzing the changes (induced by motion) on the images captured by its cameras.

The advantage of VO with respect to wheel odometry is that VO is not affected by mechanical factors such as wheel slippage on uneven terrains. It has been demonstrated that compared to wheel odometry, VO provides more accurate trajectory estimates on uneven terrains [29]. Therefore, VO can be an interesting supplement to wheel odometry, and also to other navigation systems such as global positioning system (GPS), inertial measurement units (IMUs), and laser odometry. Additionally, when combined with pose optimization techniques like bundle adjustment, VO produces less drift over long-distance travel, which is unachievable even by high-precision IMUs. Visual odometry would be especially important in GPS-denied environments such as underwater, underground, or even inside buildings.

VO is almost always the first step towards more advanced vision-based navigation tasks [30], including visual servo control [2, 3], mapping [31], visual SLAM (Simultaneous Localisation and Mapping) [32] etc. A cooperative vision-based exploration technique using two robots – one aerial and one ground robot – has been shown in [6]. The paper presents a complete system which incorporates a visual-odometry-based pose estimation method to allow a drone to navigate in indoor environments in cooperation with a ground robot, which gives the drone an estimate of its pose.

VO is achieved by tracking features in the sequence of images captured by the on-board camera of the robot, and analyzing the information obtained at each frame to estimate the pose of the camera. At each frame, the pose of the camera is calculated using the 3D-2D correspondences of the tracked features. The pose estimations at each frame can then be combined to obtain the trajectory of camera motion.

The remainder of the report is organized in the following manner: The concepts that were required for understanding and implementing VO are briefly described in Chapter 9. The VO algorithm developed is examined in Chapter 10. The dataset used for its implementation and the tuning of the algorithm parameters for the dataset are explained in Chapter 11. Chapter 12 contains the experiments conducted to test the performance of the developed algorithm and the results obtained. Chapter 13 concludes the report with a review of the project and intended future works.

Chapter 9

Theory Involved

9.1 The Pinhole Camera Model

The most commonly used mathematical model for a camera projection is the pinhole camera model. It is an idealization of the thin lens model as aperture shrinks to zero [33]. It defines the geometric relationship between a 3D point in space and its corresponding 2D projection onto the image plane, by assuming that exactly one ray passes through each point in: the image plane, the pinhole and a given scene point. The mapping of the 3D point to the image plane is called perspective projection [34].

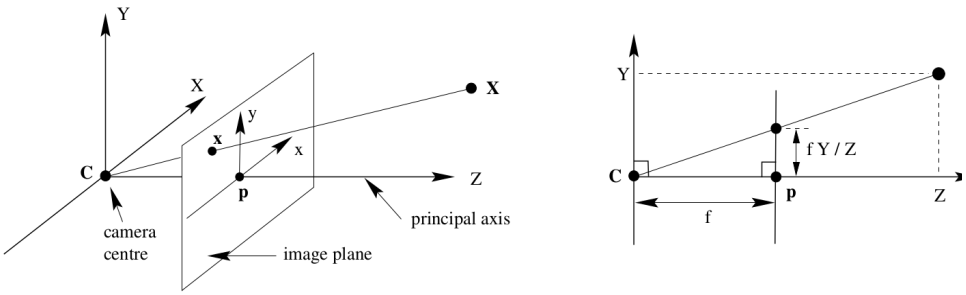


Figure 9.1: The Pinhole Camera Model
Source: BrainTrekking, *'A C++ code to compute OpenGL 4×4 GL_MODELVIEW_MATRIX from 2D-3D points homography'*
Retrieved from: `https://braintrekking.wordpress.com/2013/06/02/a-c-code-to-compute-opengl-4x4-gl\_modelview\_matrix-from-2d-3d-points-homography/`

Figure 9.1 shows the pinhole camera model. Though actually a pinhole camera produces an inverted image behind the pinhole, it is convenient to consider a virtual image associated with a plane lying in front of the pinhole at the same distance from it as the actual image plane. Hence the pinhole camera model usually assumes the virtual plane in front of it as the image plane, as in Figure 9.1.

The camera is at the origin of the world frame C. The point $\mathbf{X}$ represents a point in the real world with coordinates (X,Y,Z) in the world frame, and $\mathbf{x}$ is the projection of $\mathbf{X}$ on the image plane. The distance of the point $\mathbf{x}$ from the camera centre C along the principal axis of the camera is the *focal length* of the camera f. As shown in Figure 9.1 (right), the coordinates of the point $\mathbf{x}$ on the image plane can be found using the properties of similar triangles. Hence, the projection of $\mathbf{X}$ on to the image plane at pixel position (u, v) can be written as:

$$u = \frac{fX}{Z} \qquad v = \frac{fY}{Z}$$

To express this relation in a linear mathematical operation, both the world coordinates and the image coordinates can be converted to homogeneous system, as follows:

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

where $\lambda = Z$ is the homogeneous scaling factor.

This equation projects the point **X** in the world coordinate (X,Y,Z) to **x** in image plane coordinates (u,v). This is the most simple form of the projection equation, and the 3×4 matrix that converts the point from the world coordinates to the image coordinates is called the projection matrix or projectivity, **P**.

It can be seen that this projection assumes that the origin of the image coordinate is at the principal point **p**. However, most of the current imaging systems define the origin of the pixel coordinate system at the top-left pixel of the image. Therefore, to convert the image coordinates to the pixel coordinates, an offset of the origin from the principal point has to be considered in the projection. If o_x and o_y are the offsets of the origin along the x and y directions from **p**, then the projection equation becomes

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & o_x & 0 \\ 0 & f & o_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

To define the above equation, it was implicitly assumed that the dimensions of the image focal plane were in pixels and were square, i.e the number of pixels along the width and height of the image plane are equal. Both these assumptions are invalid in most cases. If a point is projected to a sensor of size $W_s \times H_s$ (in metres), and the pixels are arranged in a rectangular pixel matrix of size $M_x \times M_y$, then two conversion parameters m_x and m_y can be defined as

$$m_x = \frac{M_x}{W_s} \qquad m_y = \frac{M_y}{H_s}$$

Now the projection can be converted from the image focal plane into image pixel plane by multiplying with a conversion matrix as follows

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} m_x & 0 & 0 \\ 0 & m_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f & 0 & o_x & 0 \\ 0 & f & o_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$\Rightarrow \lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_x & 0 & o_x & 0 \\ 0 & \alpha_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

The ratio α_y/α_x is called the aspect ratio of the camera, and defines the ratio of the width of the image to its height. In practice, many a times the pixels in the image may be skewed due to various reasons, such as inaccurate synchronization of pixel-sampling. Hence the skew factor needs to accounted for in the projection equation.

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} \alpha_x & s & o_x & 0 \\ 0 & \alpha_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\mathbf{P}} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

where s is the skew factor, and **P** denotes the projection matrix for the model.

The projection matrix **P** can be split into two separate matrices as follows:

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} \alpha_x & s & o_x \\ 0 & \alpha_y & o_y \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{K}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{[I_3|0]} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{9.1}$$

where **K** is known as the calibration matrix and I_3 is the 3×3 identity matrix.

The second part of the projectivity, $[I_3|0]$ tells us that the camera coordinate frame is aligned with the world coordinate frame, i.e. the camera is located at the origin of the world frame and is looking down its z axis [33]. If the camera frame is different from the world frame, as is often the case, the projection equation and the projection matrix changes.

If R is the 3×3 rotation matrix relating the orientations of the world frame to that of the camera frame, and $C = \begin{bmatrix} c_x & c_y & c_z \end{bmatrix}^T$ defines the origin of the camera frame in the world frame, the world-to-camera coordinate system transformation can be written as

$$\mathbf{X}_{cam} = R\,(\mathbf{X} - C)$$

where $\mathbf{X}_{cam}$ is the position of point $\mathbf{X}$ in the camera coordinates. Using this in the projection equation (9.1),

$$\mathbf{x} = \mathbf{K}\,[I_3|0]\,\mathbf{X}_{cam}$$

$$\Rightarrow \mathbf{x} = \mathbf{K}\,[R|(-RC)]\,\mathbf{X} \qquad \Rightarrow \mathbf{x} = \mathbf{K}\,[R|t]\,\mathbf{X}$$

where t is a 3×1 vector defined as $t = -RC$. It can be noted that the position of the camera in the world coordinates can be obtained as:

$$C = -R^{-1}t \qquad \Rightarrow C = -R^T t \tag{9.2}$$

$R^{-1} = R^T$ since rotation matrices are orthogonal.
The complete projection equation now becomes

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} \alpha_x & s & o_x \\ 0 & \alpha_y & o_y \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{K}} \underbrace{\left(\begin{array}{c|c} R & t \end{array}\right)}_{[R|T]} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$\mathbf{x} = \mathbf{P}\mathbf{X} \tag{9.3}$$

where $\mathbf{x} = \lambda \begin{bmatrix} u & v & 1 \end{bmatrix}^T$ is the image point in homogeneous coordinates.

It can be seen that the calibration matrix $\mathbf{K}$ contains all the intrinsic parameters of the camera, which are required to convert a 3D point in the camera coordinates to the image coordinates, while the matrix $[R|T]$ contains the extrinsic parameters that denote the coordinate system transfrmations from 3D world coordinates to 3D camera coordinates (Figure 9.2). The extrinsic parameters contain the information regarding the position and heading of the camera in world coordinates. The projection matrix $\mathbf{P}$ can now be completely defined as

$$\mathbf{P} = \mathbf{K}[R|t] \tag{9.4}$$

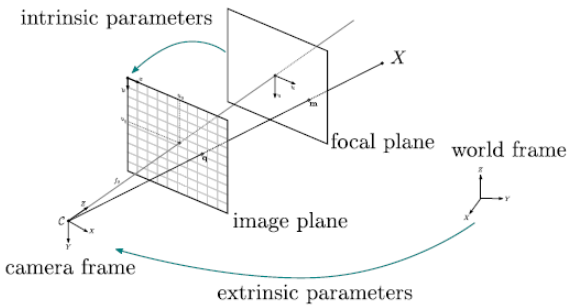


Figure 9.2: Effect of Intrinsic and Extrinsic Parameters on a 3D Point
Source: openMVG, *'cameras'*
Retrieved from: `http://openmvg.readthedocs.io/en/latest/openMVG/cameras/cameras/`

9.2 Reprojection Error

The reprojection error is a measure of how well the projection model (9.1) can estimate the position of a point on the image, given its 3D position in space [35]. It can be defined as the distance between the position of the reprojected point and the actual position of that point in the image.

If $\hat{\mathbf{x}}$ is the image projection of the point $\mathbf{X}$, given by $\hat{\mathbf{x}} = \mathbf{P}\mathbf{X}$, then the reprojection error is the Euclidean distance $d(\mathbf{x}, \hat{\mathbf{x}})$, between $\hat{\mathbf{x}}$ and the actual position of the point in the image, $\mathbf{x}$.

9.3 The Perspective-n-Point Problem

Estimating the pose of a calibrated camera from a set of n 3D points in the world and their corresponding 2D projections in the image is called the Perspective-n-Point (PnP) problem. Many solutions to this problem can be found in literature [36, 37]. The pose of the camera can be found using equation (9.2) if the projection matrix $\mathbf{P}$ and the calibration $\mathbf{K}$ are known. $\mathbf{K}$ is usually constant for a camera and is

obtained using a method called *calibration* (Appendix A.1). However, $\mathbf{P}$ is generally not known, and has to be estimated using the 3D-2D correspondences of reference points.

The ideal formulation for the PnP problem is to find a projection matrix $\mathbf{P}$ which minimizes the reprojection error of the points.

$$\arg\min_{\mathbf{P}} \sum \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2$$

where $\mathbf{x}_i$ is the actual position of the i^{th} feature in the image, and $\hat{\mathbf{x}}_i$ its projection ($\mathbf{PX}_i$).

9.3.1 Direct Linear Transformation

The straightforward solution for the PnP problem, i.e. estimating the projection matrix from a set of 3D-2D correspondences, is the Direct Linear Transformation (DLT) algorithm [38]. It finds the values in $\mathbf{P}$ using a system of linear equations. It can be seen from the projection equations (9.3 & 9.4) that the projection matrix has 12 entries and 11 degrees of freedom (ignoring scale). Expanding the projection equation, each 3D-to-2D correspondence can give two constraint equations of the form [39]:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & -X & -Y & -Z & -1 & Xy & Yy & Zy & y \\ X & Y & Z & 1 & 0 & 0 & 0 & 0 & -Xx & -Yy & -Zy & -y \end{bmatrix} \mathbf{P} = 0$$

$$A_i \mathbf{P} = 0$$

where (X,Y,Z) are the coordinates of the point $\mathbf{X}_i$ in 3D space and (x,y) is the coordinates of its projection $\mathbf{x}_i$ in the image. A_i is the constraint matrix for the i^{th} point correspondence [40].

Hence, to find the 11 unknowns of $\mathbf{P}$, a minimum of $11/2 = 5\frac{1}{2}$ correspondences are necessary. This means 5 complete correspondences and any one of the two 2D image values (x or y) of a sixth correspondence has to be known. Given this minimum number of correspondence, the solution is exact and can be obtained by solving $A\mathbf{P} = 0$, where A is the 11×12 constraint matrix. The data is not always accurate in practical cases due to noise. Therefore, there may not always be a single value for $\mathbf{P}$ which can solve the homogeneous equation $A\mathbf{P} = 0$. In such cases, a total least squares solution can be used by choosing $\mathbf{P}$ as a right singular vector corresponding to the smallest singular value of A [40].

This function does not always produce the best result, since the estimated 'rotation matrix' in the projection matrix $\mathbf{P}$ may not necessarily be a typical rotation matrix having properties such as unit determinant and orthogonality. Therefore, the result obtained by the DLT method is usually optimised using optimisation techniques such as the Lavenberg-Marquardt Optimization (Appendix A.2)

9.3.2 Robust Estimation: Dealing with Outliers

Once the projectivity solution is obtained using the DLT method, RANSAC can be used to verify its accuracy and to find a better solution if available. Algorithms based on RANSAC (Random Sample Consensus) are generally used for making the solution to the PnP problem more robust [40]. RANSAC was introduced by Fischler and Bolles (1981) as an efficient method to handle erroneous correspondences [41]. A smaller set of correspondences is randomly picked and used to estimate a hypothesis (projectivity solution, in this case). Then all the data points are transformed using this solution and the reprojection errors measured. Correspondences with a reprojection error below some predefined threshold are considered inliers to the candidate solution. This procedure is repeated a number of times and the hypothesis (projectivity) that has the highest consensus (most inliers) is selected as the solution. An advantage of this algorithm is that once a good solution is obtained, it can determine the outliers or noise in the dataset.

Applying RANSAC to the PnP problem, Fischler and Bolles claimed that the minimum number of correspondence pairs required to obtain finite number of solutions is just three (P3P). They showed that at most four positive solutions are obtained by solving the P3P problem. They also showed that with more number of correspondences, RANSAC can be used to find the best possible solution to fit the dataset. Complete solution to the P3P problem have been obtained using analytical and geometric methods [42].

9.3.3 Obtaining Camera Pose from the Projection Matrix

Given the camera calibration matrix $\mathbf{K}$, equation (9.4) can be used to find the rotation matrix R and the translation vector t as follows:

$$[R|t] = \mathbf{K}^{-1}\mathbf{P}$$

The position of the camera C in the world frame can be calculated from R and t as was shown earlier (9.2).

$$C = -R^T t$$

9.4 Epipolar Geometry and the Fundamental Matrix

Now that the camera projection has been explained, the concept of stereo images and epipolar geometry can be understood. When a 3D scene is viewed from two distinct positions, there are certain geometric relations between the 3D points and their projections onto the two 2D images, which lead to constraints between the image points. The geometry defining these constraints is called epipolar geometry [43].

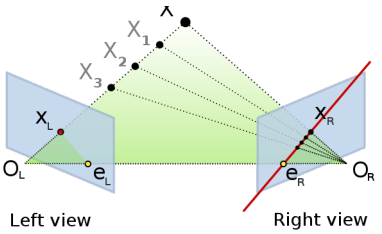


Figure 9.3: Epipolar Geometry
Source: Wikipedia,'*Epipolar Geometry*'
Retrieved from: `https://en.wikipedia.org/wiki/Epipolar\_geometry\#Epipolar\_line`

Figure 9.3 depicts two pinhole cameras (or one camera at two instances of time) looking at point $\mathbf{X}$ in space. Here, as before, the projection problem is simplified by placing a virtual image plane in front of the focal center (instead of the actual image plane behind the camera), to produce an image that is not transformed by the symmetry. O_L and O_R represent the two camera centers. Points $\mathbf{x}_L$ and $\mathbf{x}_R$ are the projections of point $\mathbf{X}$ onto the image planes. The line $O_L\mathbf{X}$ is seen by the left camera as a point because it is directly in line with that camera's optical center. However, the right camera sees this line as a line ($e_R\mathbf{x}_R$) in its image plane, and is called an *epipolar line*. Symmetrically, the line $O_R\mathbf{X}$, seen by the right camera as a point, is seen as epipolar line $e_L\mathbf{x}_L$ by the left camera.

9.4.1 Epipolar Constraints

The epipolar geometry gives two important observations:

(a) If the points $\mathbf{x}_L$ and $\mathbf{x}_R$ correspond to the same 3D point $\mathbf{X}$, then their projection lines must intersect precisely at $\mathbf{X}$. This means that if the image coordinates of $\mathbf{x}_L$ and $\mathbf{x}_R$ are known, $\mathbf{X}$ can be calculated by a process called *triangulation* (section 9.4.2).

(b) If the projection point $\mathbf{x}_L$ is known, then the epipolar line $e_R\mathbf{x}_R$ is known and the point $\mathbf{X}$ should project a point $\mathbf{x}_R$ onto the right image plane, which must lie on this particular epipolar line. This means that for each point observed in one image the same point must be observed in the other image on a known epipolar line. This is an epipolar constraint, and can also be described using a *Fundamental Matrix* (section 9.4.3).

9.4.2 Triangulation

Triangulation or reconstruction of a point in 3D space can be achieved if the image coordinates of the point in two subsequent images, and the projection matrices defining their projections are available. If $\mathbf{P}_L$ and $\mathbf{P}_R$ are the projection matrices to the two images, the two rays in space corresponding to the two image points may easily be computed. Theoretically, intersecting two lines in space is trivial. However, in the presence of noise, these rays can not be guaranteed to cross and the measured points may not lie on corresponding epipolar lines. In this case, the best solution under some assumed noise model has to found.

The most commonly used method is linear triangulation [44, 45]. The projection matrix for the left camera can be written in the form $\mathbf{P}_L = \begin{bmatrix} p_{L_1}^T & p_{L_2}^T & p_{L_3}^T \end{bmatrix}^T$, where $p_{L_i}^T$ denotes the i^{th} row of $\mathbf{P}_L$. The projection equation for this camera can be written as $\mathbf{x}_L = \mathbf{P}_L\mathbf{X}$. From section 9.1, $\mathbf{x}_L = \lambda_L \begin{bmatrix} u_L & v_L & 1 \end{bmatrix}^T$ in homogeneous coordinates. The homogeneous scale factor λ can be eliminated by cross product $\mathbf{x}_L \times \mathbf{P}_L\mathbf{X} = 0$, to give three equations, of which two are linearly independent [46].

$$u_L(p_{L_3}^T\mathbf{X}) - p_{L_1}^T\mathbf{X} = 0$$

$$v_L(p_{L_3}^T\mathbf{X}) - p_{L_2}^T\mathbf{X} = 0$$

$$u_L(p_{L_2}^T\mathbf{X}) - v_L(p_{L_1}^T\mathbf{X}) = 0$$

These equations are linear in the components of $\mathbf{X}$. Using the same steps for the right camera image, an equation of the form $A\mathbf{X} = 0$ can be formed, where

$$A = \begin{bmatrix} u_L p_{L_3}^T - p_{L_1}^T \\ v_L p_{L_3}^T - p_{L_2}^T \\ u_R p_{R_3}^T - p_{R_1}^T \\ v_R p_{R_3}^T - p_{R_2}^T \end{bmatrix}$$

This gives a total of four equations in four unknowns. However, this is a redundant set of equations, since the solution is determined only up to scale. $\mathbf{X}$ can be determined by solving $A\mathbf{X} = 0$, using the singular value decomposition (SVD). However, SVD is known to be the most time-consuming of all the matrix factorizations [46]. Therefore, to improve efficiency, a transformation of $A\mathbf{X} = 0$ can be used. Since, the last element of the homogeneous coordinate $\mathbf{X}$ is 1, the function can be rewritten as follows:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \\ A_{41} & A_{42} & A_{43} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} -A_{14} \\ -A_{24} \\ -A_{34} \\ -A_{44} \end{bmatrix}$$

Now the 3D coordinates (X, Y, Z) can be solved using the pseudoinverse, which is much more efficient than SVD. The only drawback of using the pseudoinverse is that the method assumes that the 3D point $\mathbf{X}$ is never at infinity. However, since in the real-world, there are few points at infinity, this method will not affect the precision of triangulation.

9.4.3 The Fundamental Matrix

The fundamental matrix $\mathbf{F}$ is a 3×3 matrix which can relate corresponding points in stereo images. Specifically, it relates the two points $\mathbf{x}_L$ and $\mathbf{x}_R$ such that the product $\mathbf{F}\mathbf{x}_L$ describes the epipolar line $\mathrm{e}_R\mathbf{x}_R$ on which the corresponding point on the other image, $\mathbf{x}_R$, must lie (Figure 9.3). That means, for all pairs of corresponding points, the following equation holds

$$\mathbf{x}_R^T \mathbf{F} \mathbf{x}_L = 0$$

Therefore, given the projection of a scene point into one of the images, the corresponding point in the other image is constrained to a line described by the above equation, helping the search, and allowing for the detection of wrong correspondences [47].

Chapter 10

Algorithm

Formal Algorithm

Visual odometry can be achieved by continuously estimating the camera pose at each frame. Using the methods explained in the previous sections, a code for implementing Visual Odometry (VO) was written in python. Various OpenCV functions were adopted for efficiency and better results. Algorithm 1 shows a basic structure of the algorithm implemented for visual odometry in this project.

Algorithm 1 Visual Odometry

Input: n reference points with known 3D coordinates, sequence of m gray scale images
Output: Estimated trajectory followed by the camera

1: **procedure** ESTIMATE CAMERA POSE AND TRAJECTORY
2: Run Template Matching on the current frame using all n features (Algorithm 2) (*Section* 10.1)
3: **if** No of features tracked > minimum required **then**
4: Estimate projection matrix from 3D-2D correspondences of those features by solving PnP problem for the first frame (*Section* 9.3)
5: Compute camera pose for the first frame (*Section* 9.3.3)
6: **for** frame = $2 : m$ **do**
7: Track 2D features from previous frame using optical flow method
8: **if** No. of features tracked < minimum required **then**
9: Find new features and update correspondences (Algorithm 3) (*Refer Section* 10.3)
10: **end if**
11: Estimate projection matrix from 3D-2D correspondences by solving PnP problem for frame
12: Compute camera pose for frame
13: **end for**
14: **Plot** trajectory of camera, **return** Final Mean Reprojection Error
15: **end if**
16: **end procedure**

The algorithm is given a set of patches with known 3D world coordinates, and a sequence of image frames. The output of the algorithm is a plot of the estimated trajectory of the camera motion.

It starts by trying to find the initial patches (whose world coordinates are known) in the first frame using Algorithm 2 (Section 10.1). It keeps trying until enough features are matched in a frame. This is done to ensure that the visual odometry algorithm starts with a good estimation of the initial position of the camera in the world frame. Once the algorithm is able to find the required number of matches, it tries to track them in the subsequent frames using optical flow (section 10.2). New features are tracked if there are not enough of the previous features in the current frame, using Algorithm 3 (section 10.3). For each frame, the projection matrix is estimated by solving the PnP problem (section 9.3), and the camera pose is obtained. By estimating the pose of the camera at every frame, the trajectory traced by the camera can be estimated. The total mean reprojection error in the algorithm is computed to gauge the its performance.

10.1 Matching Initial Patches

The first step in the VO algorithm is finding the matches for the 'reference' features in the first image frame obtained from the camera. A minimum number of features (hardcoded as a threshold in the algorithm) has to be matched for the VO algorithm to proceed.

The method used for finding the desired features in an image is called Template Matching. It is a technique for finding parts of an image which match a given template image. There are various methods available for template matching [48], however they can be broadly classified into two: feature-based and template-based approaches. If the template image has strong features, a feature-based approach may be considered. This is mainly used when there is a possibility of the matched feature being transformed in the considered image.

When the bulk of the template image is present in the matching image (as in this case), the template-based approach is more effective. This method involves 'sliding' the template over the entire matching image, and finding the region in it which matches the most. The template is moved over the matching image one pixel at a time, and a metric which represents the degree of matching is calculated at each location. The metric chosen for matching in this project was the Normalized Cross-Correlation coefficient [49, 50, 51] due to its efficiency in computation and robustness in terms of image lighting. By normalizing the image and template vectors to unit length in this way, the metric can make up for the difference in the image energy that occur due to variations in lighting conditions.

If I is the image and the sum is over x, y under the window containing the feature f positioned at u, v, the coefficient at each position is computed as

$$R(x,y) = \frac{\sum_{x,y}\left(I(x,y) - \bar{I}_{u,v}\right)\left(f(x-u,y-v) - \bar{f}\right)}{\sqrt{\sum_{x',y'}\left(I(x,y) - \bar{I}_{u,v}\right)^2\left(f(x-u,y-v) - \bar{f}\right)^2}}$$

where $\bar{f}$ is the mean of the template and $\bar{I}_{u,v}$ is the mean of $I(x,y)$ in the window under the template.

Algorithm 2 Template Matching

Input: Gray scale template image f of size $w \times h$; gray scale matching image I of size $W \times H$
Output: Top left corner position of the most matching region in I. (Window size: $w \times h$)

```
procedure MATCH TEMPLATE
    for row x = 0:(H − h) do
        for column y = 0:(W − w) do
            R_{x,y} ← NCC at (x, y) between the window at I and f
        end for
    end for
return x, y of max(R), max(R)
end procedure
```

This process gives a matrix R (of size $(W-w+1) \times (H-h+1)$) for each template, which contains the normalized cross-correlation coefficient at each $(w \times h)$ window in the matching image. The highest value denotes the best matching region in the image. If the highest value ($\max(R)$) is above a threshold, it is defined as a good match, and the feature is selected for tracking in the next image. Now that the positions of the features are known in both the world frame and the image frame, the position and orientation of the camera can be estimated as described in Section 9.3.

10.2 Tracking Features across Images

Now that the camera pose is estimated for the first image, the algorithm takes in the second image frame from the sequence. This is equivalent to the camera moving (with respect to the tracked points in the world frame) and capturing another image. The previously located features are obviously not going to be at the same image coordinates. The process of estimating the translation of a point given two subsequent images is called feature tracking across images.

This is usually achieved using the concept of *optical flow*. When there is a relative motion between the scene and the observer (the camera), there is an apparent motion of the objects in the image. This pattern of apparent motion that is observed on the image due to the relative motion between the camera (observer) and the scene is called optical flow.

Since these discreet image displacements occur over a sequence of ordered images, and the images can be assumed to be captured by the same camera at discreet intervals of time, optical flow can be said to be an approximation to image motion defined as the projection of velocities of the 3D points onto the image plane [52]. Mathematically, it is 2D vector field where each vector is a displacement vector showing the movement of points from first frame to second [53].

Each pixel has a different intensity value (I) depending on its position in the image (x,y) and the time frame (t), which denotes the time the image was taken relative to the other images in the sequence. Optical flow methods try to calculate the motion between the image frames taken at time t and $t + \Delta t$ for each pixel position. If after time Δt, a point in an image moved Δx and Δy in the x and y direction respectively from its initial position (x,y), a brightness consistency constraint can be obtained that can relate the pixels in the two image frames as below:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{10.1}$$

By assuming the motion of the point between the two frames to be small, it is possible to use the Taylor series approximation on the right-hand side of the equation as follows:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t$$

Using this in equation 10.1, the constraint now becomes

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0$$

By taking the time derivative, the equation can be written as:

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t}\frac{\Delta t}{\Delta t} = 0$$

$$\Rightarrow \frac{\partial I}{\partial x}\dot{x} + \frac{\partial I}{\partial y}\dot{y} + \frac{\partial I}{\partial t} = 0$$

Now the optical flow (or the velocity components) of the pixel $I(x, y, t)$ are $\dot{x}$ and $\dot{y}$ in the x and y directions respectively. $\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial t}$ are the derivatives of the image at (x, y, t) in the corresponding directions. Rewriting these as I_x, I_y and I_t respectively, the equation becomes:

$$I_x\dot{x} + I_y\dot{y} = -I_t \qquad \Rightarrow \qquad \nabla I^T v = -I_t \tag{10.2}$$

where v is the velocity vector. Therefore a single pixel information gives an equation of two unknown variables (I and v), which cannot be solved without another set of equations.

10.2.1 Lucas-Kanade Method

Assuming that the displacement of a point across two subsequent images is small, an equation in two unknowns was obtained (10.2). Lucas-Kanade method additionally assumes that the displacement of the pixels in the *neigbourhood* of that pixel are constant [54]. Therefore, the optical flow equation can be assumed to hold for all the pixels within the window centered at the pixel considered p. Lucas-Kanade method usually considers a 3×3 window. Now, the local optical flow (or velocity vector) will be constant for all the pixels in the window, i.e. the local optical flow vector $(\dot{x}, \dot{y})$ must satisfy:

$$I_x(q_1)\dot{x} + I_y(q_1)\dot{y} = -I_t(q_1)$$

$$I_x(q_2)\dot{x} + I_y(q_2)\dot{y} = -I_t(q_2)$$

$$I_x(q_2)\dot{x} + I_y(q_2)\dot{y} = -I_t(q_2)$$

$$\vdots$$

$$I_x(q_n)\dot{x} + I_y(q_n)\dot{y} = -I_t(q_n)$$

where q_i is the i^{th} pixel in the window, and $I_x(q_i), I_y(q_i), I_t(q_i)$ are the partial derivatives of I with respect to position x, y and time t, evaluated at the point q_i. This equation can be written in vector form as

$$Av = b$$

where A is a 9×2 matrix containing the I_x and I_y for each of the nine pixels in the window, v is the velocity vector $[\dot{x} \quad \dot{y}]^T$, and b is a 9×1 vector containing the negative of I_t for each pixel. The system of equation now has more equations than unknowns and is over-determined. Lucas-Kanade method uses the least squares principle to solve a 2×2 form of this system obtained by multiplying with A^T.

$$A^T A v = A^T b$$

$$\Rightarrow v = (A^T A)^{-1} A^T b$$

$$\Rightarrow \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \sum_i (I_x(q_i))^2 & \sum_i I_x(q_i) I_y(q_i) \\ \sum_i I_y(q_i) I_x(q_i) & \sum_i (I_y(q_i))^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i) I_t(q_i) \\ -\sum_i I_y(q_i) I_t(q_i) \end{bmatrix}$$

The sums are running from $i = 1$ to 9.

Therefore, given the positions of a point to track, the Lucas-Kanade method gives the optical flow vectors for the point, using which its position in the next frame can be determined. The method then looks for the feature at the position in the second image. Since the position of the feature will not be exactly at the determined position, a threshold (which depends on the task and dataset) is set to determine the neighbourhood to be searched.

10.3 Making up for Lost Features

It is inevitable that over time the algorithm will lose track of the features it had been tracking in the previous images. This could be because the features are not in the field of view of the camera anymore, or they become occluded by other objects in the scene etc. As was explained in section 9.3, the number of feature correspondences play an important role in obtaining good pose estimation of the camera. Furthermore, if the number of features available is too low, a projection matrix can not be estimated at all. Hence, finding new 'good' feature correspondences is a very important part of a visual odometry algorithm.

The steps followed to achieve this is shown formally in Algorithm 3. More details regarding the implementation of each step follows.

Algorithm 3 Finding New 2D-3D Correspondences

Input: Current gray image frame I_t and previous gray image frame I_{t-1}
Output: Image coordinates (x_i, y_i) in I_t and world coordinates (X_i, Y_i, Z_i) of the new features

1: **procedure** UPDATE FEATURES
2: **if** baseline > minimum required baseline **then**
3: Find new feature points in I_t and I_{t-1} using SIFT
4: Find matching features using Ratio Test
5: Refine matches using Mutual-Match Test
6: Filter out outliers and optimise using RANSAC
7: Obtain the image coordinates (x_i, y_i) of the final features in I_t
8: Estimate 3D world coordinates (X_i, Y_i, Z_i) of the new features by triangulation, using their pixel positions in I_t and I_{t-1}
9: **end if**
10: **return** (x_i, y_i), (X_i, Y_i, Z_i)
11: **end procedure**

10.3.1 Feature Detection using SIFT

In 2004, D. Lowe introduced a method to detect and describe features that are invariant to scale, translation, rotation and other imaging parameters, called the Scale-Invariant Feature Transform (SIFT) [55]. He called such features *keypoints*. The SIFT algorithm finds good keypoints and gives a unique description vector for each keypoint called a *descriptor*. The main steps involved in finding keypoints and their descriptors are explained below.

(a) *Scale-Space Extrema Detection*

The first stage of keypoint detection is to identify locations and scales of features which can be repeatably assigned under different views of the same object. It is obvious that the same window

cannot be used to detect keypoints with different scale. Therefore, a *scale-space* filtering method is adopted for finding different features at different scales. Scale-invariant features are detected by searching for stable features across all the possible scales, using a continuous function.

The Gaussian function is claimed to be only possible scale-space kernel [55]. The Laplacian of Gaussian (LoG) function can be used as a blob-detector that can detect blobs of various sizes in the image due to change in the standard deviation of its function, σ. SIFT approximates the LoG function using the Difference of Gaussian (DoG) function, which is computationally much less expensive. DoG is obtained simply by subtracting the Gaussian blurrings of an image with two different standard deviations (σ and $k\sigma$).

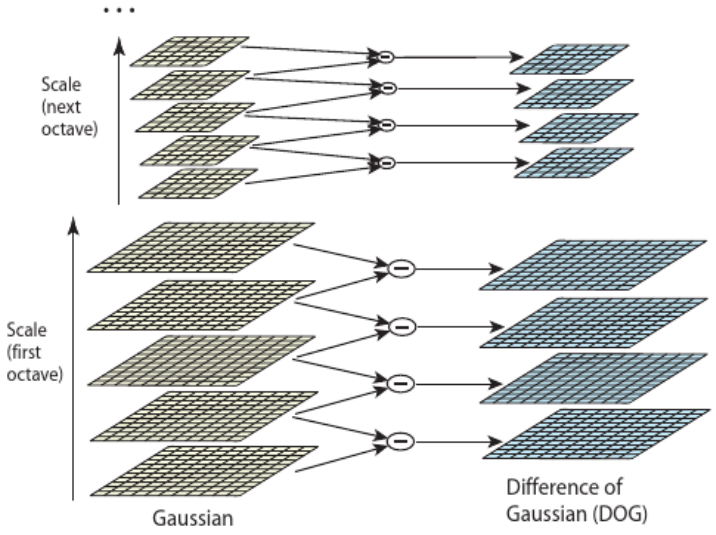


Figure 10.1: SIFT Image Pyramid
Source: [55]

SIFT algorithm is applied over an *image pyramid*, which has several subgroups called *octaves* (Figure 10.1 [55]). In each octave, the initial image is repeatedly convolved with Gaussians to produce a set of scale space images. Adjacent Gaussians are then subtracted to produce the DOG. After each octave, the Gaussian image is down-sampled by a factor of 2 to produce an image $\frac{1}{4}$ times the previous size, and the process is repeated again.

Once the DoG are found for the image pyramid, the local maxima and minima of the DoG is found. This is done by comparing each sample point pixel to its eight neighbours in the current image and nine neighbours in the scale above and below, i.e. the point is compared to all the points in its $3 \times 3 \times 3$ neighbourhood. The sample point is selected as a keypoint if it is either larger than all of its neighbours or smaller.

(b) *Keypoint Localization*

Once a keypoint candidate is found, the next step is to fit it to the nearest data for location and scale. Lowe used Taylor series expansion of the scale-space to get more accurate locations of extrema. If the intensity there is less than a threshold (0.03 as per [55]), it is discarded.

The DoG function has higher response for edges, which are not very good keypoints due to their possible ambiguity along the edge. A 2×2 Hessian matrix is used in the SIFT algorithm to remove such points. It is known that for edges, one eigenvalue is greater than the other. So any point which has the ratio of the two eigenvalues greater than a threshold (10 as per [55]), the point is rejected.

(c) *Orientation Assignment*

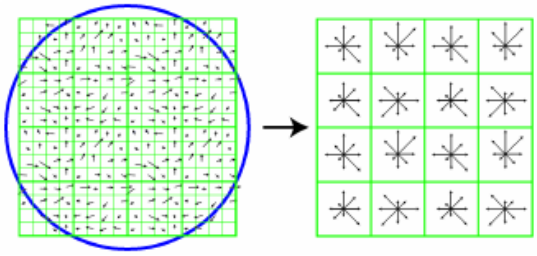
To achieve invariance to image rotation, an orientation is assigned to each keypoint. An orientation histogram is formed from the gradient orientations of sample points within a region around the keypoint. The scale of the keypoint is used to select the Gaussian smoothed image in the scale-space with the closest scale, so that all computations are performed in a scale-invariant manner.

The orientation histogram is made of 36 bins, dividing the total 360 degrees into bins of 10 degrees each. Depending on the orientation of the gradient of a sample point in the region, a value is added

to the corresponding bin, weighted by its gradient magnitude and by a Gaussian-weighted circular window with a σ that is 1.5 times that of the scale of the keypoint. Dominant directions of the local gradients can be identified as the bins with high peaks in the orientation histogram. The highest peak in the histogram is detected and assigned as the orientation of that keypoint. Any other local peak that is within 80% of the highest peak is used to create another keypoint with that orientation. Therefore, for locations with multiple peaks of similar magnitude, there will be multiple keypoints created at the same location and scale but different orientations. Lowe claims that although only about 15% of points are assigned multiple orientations, it contribute significantly to the stability of matching.

(d) *Keypoint Descriptor*

Now that each keypoint has been assigned with a location, scale and orientation, the next step is to define a descriptor for it, which is highly distinctive yet is as invariant as possible to the remaining descriptors in terms of variations such as change in illumination or 3D viewpoint. It should then be possible to use this descriptor to identify the same feature in a different image.



Left: Gradient of Image. Right: Descriptor of the Keypoint

Figure 10.2: SIFT Descriptor Vector Formulation

Source: [56]

Figure 10.2 shows the formulation of a SIFT descriptor for a keypoint. For creating a descriptor for each keypoint, a 16×16 neighbourhood around the keypoint is taken (Figure 10.2-left), and divided into 16 sub-blocks of 4×4 size (Figure 10.2-right). For each sub-block, an orientation histogram of 8 bins each is created (dividing the total 360 degrees into bins of 45 degrees each). The bins are filled with values depending on the orientation of the sample point in the window, as was done for orientation determination (section 10.3.1(c)). However, this time a Gaussian weighting function with σ equal to one half the width of the descriptor window is used to assign a weight to the magnitude of each sample point (denoted by the circle in the left image). The purpose of this weighting is to avoid sudden changes in the descriptor with small changes in the position of the window, and to give less importance to gradients that are far away from the centre of the window.

Now a total of $4 \times 4 \times 8 = 128$ bins are available. The descriptor is defined as a vector of size 128, containing the values of the 128 bins for the keypoint. The vector is normalized to unit length to make the descriptor robust to contrast changes in the image. This makes the descriptor invariant to changes in illumination.

By detecting local maxima in scale space and gathering 128-bin values as a descriptor, SIFT has the following advantages:

- Contrast-invariance and illumination-invariance due to normalization of the scale of the derivative.
- Scale-invariance, due to non-maximum suppression and processing of image patches in scale space.
- Rotation-invariant, since the the orientation is assigned as the highest peak in the histogram .

10.3.2 Keypoint Matching

The SIFT algorithm is used to find SIFT features and their descriptors in images I_t and I_{t-1}. Matching keypoints are found by comparing the SIFT descriptors from the two images. Lowe used a 'ratio test' for finding the best matches. This report describes two additional steps that were implemented to get better results.

(a) *Lowe's Ratio Test*

To find the best match for a descriptor from I_{t-1} to I_t, the nearest neighbor is defined as the keypoint in I_t with minimum Euclidean distance for the invariant descriptor vector from I_{t-1}. It is to be noted that there may be some features in I_{t-1} for which there are no good matches in I_t. So it will be necessary to discard features which do not have a good match.

However, since a global threshold on the distance to the closest feature does not perform well, Lowe compared the distance of the closest neighbor to that of the second-closest neighbor. He claims that this measure performs well because "*correct matches need to have the closest neighbor significantly closer than the closest incorrect match to achieve reliable matching*".

Hence, good matches are defined as those which have the ratio (given by the distance to closest neighbour to that to the second-closest) less than a threshold value (0.8 in Lowe's paper). This ratio is claimed to have rejected 90% of the false matches while discarding less than 5% of the correct matches.

(b) *Mutual Consistency Test*

The quality of feature matching can be greatly improved by only keeping matches that are mutual. The descriptors for feature pair from both images are matched and is defined as a good match only if both descriptors in a matched pair agree that the closest matching feature is the other in that pair. If the best match for descriptor i in image I_{t-1} is j in image I_t, and the best match for descriptor j in image I_t is i in image I_{t-1}, then the pair i and j is selected as a match.

(c) *Removing Outliers with RANSAC*

The matches obtained are further refined by removing outliers. This is done again using RANSAC, which generally works by estimating a mathematical model that fits the maximum number of data provided. Here, RANSAC is used to estimate a fundamental matrix $\mathbf{F}$ (section 9.4.3) that would best describe the feature correspondences in the two images I_t and I_{t-1}.

This is done by selcting a few correspondences from the image pair at a time, and estimating a fundamental matrix for the selection. The fundamental matrix $\mathbf{F}$ which can relate most of the corresponding matches is selected as the best model. Once the best suited fundamental matrix is obtained, the correspondences which cannot be described using it (outliers) are discarded. As explained in section 9.3.2, RANSAC is an outlier rejection algorithm, not an inlier selection algorithm. It is necessary to have a sufficient number of inliers to make it work, which is why the ratio test and mutual consistency tests are done first. This reduces the percentage of false positives among the matched points.

The above steps gives the image coordinates of good keypoints which can be used for tracking in the next image frames.

Now that the image coordinates of the new keypoints in images I_t and I_{t-1} are obtained, their positions in the world coordinates can be obtained using the method of triangulation, provided there is enough baseline (distance between the camera centres of the two images) for proper triangulation (Section 9.4.2).

Chapter 11

Algorithm Tuning

ISMAR 2015 - Tracking Competition

For experimentation and optimization of the algorithm, it was necessary to use a consistent dataset, on which the developed algorithm can be tested. For this project, the dataset from the off-site tracking competition conducted as a part of the International Symposium on Mixed and Augmented Reality (ISMAR) 2015 was used (available at `http://ypcex.naist.jp/trakmark/tracking-competition/?page\_id=972`).

Figure 11.1: World Coordinate System for the Dataset
Source: ISMAR 2015 Tracking Competition, *'Code and Data'*
Retrieved from: `http://ypcex.naist.jp/trakmark/tracking-competition/?page\_id=972`

The competition consisted of three levels. The first was simply to find matches of the template features in an image, and to obtain their 2D image coordinates. The aim of the second level was to track these features across a sequence of images, given their 3D positions in the world frame, and to obtain the projection matrix at each frame. All the features in the first image were present in all the subsequent images for this level. In the third level however, features will be lost and new features have to obtained. Projection matrices at each frame were to be estimated. The dataset for the third level was used for testing the developed VO algorithm, as this provided a reasonable challenge for a VO task. Once a good estimation of the projection matrix is made at each frame, the camera trajectory can be obtained easily.

11.1 The Dataset

The dataset used for testing the algorithm consists of:

(a) A sequence of 481 VGA images $(I_0, I_1, ..., I_{480})$. The input images were prepared by generating synthesized images using the POV-Ray, which is a tool for rendering high-quality computer graphics. The resolution of the image is 640 × 480 pixels. The target scene is composed of rigid objects, and illuminated by multiple static or dynamic light sources.

The target scene is a block house, made by LEGO bricks Mountain Hut (31025), placed on the flat surface (Figure 11.1). The 3D world coordinate system has the positive x-axis pointing to the right, the positive y-axis pointing down, and the positive z-axis pointing into the screen. The origin of the 3D world coordinates is the left front corner of the base plate.

(b) 40 reference points each of which has a square 65×65 pixels image patch which can be located in I_0. Additionally, the 3D world coordinates corresponding to the centre of each pixel is given in a *csv* file.

(c) The camera calibrations and the distortion coefficients of the camera are provided.

11.2 Algorithm Parameters for the Dataset

There are numerous parameters that have to be tuned for the proper implementation of a VO algorithm. The best values for these parameters are highly dependent on the type and quality of images, type of objects in the images, the frame rate, noise in the data, distortions and intrinsics of the camera etc. Hence the values of the parameters have to be specifically tuned for each task.

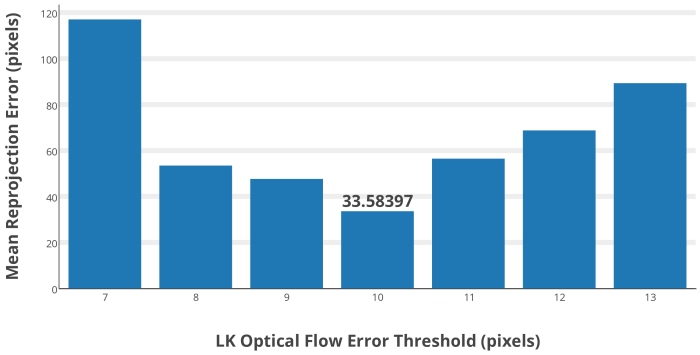


Figure 11.2: Effect of the Error Threshold of Lucas-Kanade Optical Flow Method on the Mean Reprojection Error at the end of the Algorithm

The mean reprojection error at the end of the algorithm was chosen as the metric for tuning the parameters in this project. The VO algorithm was run on the ISMAR dataset. For tuning each parameter, all other parameters were kept constant, and the mean reprojection error for different values of the candidate parameter was observed. Figure 11.2 shows the effect of the parameter 'Maximum Distance Error Threshold in Lucas-Kanade Optical Flow Computation' on the mean reprojection error, when all the other values were kept constant (the effect of other parameters are not shown due to the similarity of the experiment results and space constraint of the report).

The final values that gave the lowest reprojection error (33.58 pixel units) are shown in Table 11.1.

Parameter	Optimum Value
Minimum Probability Value for Choosing a Feature as a Template Match in the Frame (section 10.1)	0.95
Minimum Number of Template Matches to be Obtained in the First Frame (out of 40)	40
Minimum Baseline for Triangulation (section 9.4.2)	100 wcu[1]
Minimum Number of Features to be Tracked per Frame (If the number of features is less than this value, baseline is checked and new features are found. High values make code slower)	200
Inlier Threshold for RANSAC Procedure for Solving PnP (section 9.3)	3 pixels
Inlier Threshold Used by RANSAC for Removing Outliers from Mutual Matches (maximum distance from a point to an epipolar line, beyond which the point is considered an outlier)(section 10.3.2)	1 pixel
Ratio for Lowe's Ratio Test (section 10.3.2)	0.5
Maximum Distance Error Threshold in Lucas-Kanade Optical Flow Computation (section 10.2.1)	10 pixels

Table 11.1: Optimum VO Parameter Values for the ISMAR Dataset

[1]World Coordinate Units (wcu): Used here to denote length measurements in same units as the world coordinate system in the ISMAR dataset.

Chapter 12

The 'Reprojection Filter' Technique

A technique to reduce the mean reprojection error of the algorithm was implemented to improve the performance of the code. The technique removes any correspondence feature that has a reprojection error above a threshold value for each frame. After computing the opical flow and obtaining the matching feature in the second image, the reprojection errors for each of the matched features is computed. Any feature with a reprojection error above the threshold is discarded. Therefore, any feature whose 3D world position was estimated wrongly will not be considered.

The mean reprojection errors for the algorithm with and without this '*reprojection filter*' trick were compared. The estimated camera trajectories, computational time, and the number of features tracked in each frame, are also considered for comparing.

12.1 Experiments and Results

Since the dataset used is synthetic and the data is not affected by any external noise, the algorithm would perform exactly the same way each time, for a given set of parameter values. The optimum parameter values obtained (Table 11.1) were used for the experiments. **The 'reprojection filter' threshold was set as 25 pixels**. Therefore, when using the 'reprojection filter' technique, any feature in any frame which has a higher reprojection error than 25 pixels will be discarded as a bad match.

Reprojection Error: The mean reprojection error for all the features in all the frames was calculated. The mean reprojection error without the 'reprojection filter' was found to be **33.584 pixels** and the mean error using the technique was **5.769 pixels**. Therefore, it was possible to reduce the mean reprojection error by more than 82% using the 'reprojection filter' technique. The reprojection error at each frame for the algorithm with and without the technique are plotted in Figure 12.1. The left figure shows the full plot, and the right figure shows the plot with the length of the y axis limited to 50 for better visualization. It can be seen that the reprojection error with the 'reprojection filter' technique remains less than 10 pixels at all frames, whereas in the other case, the error is much higher and at times, even goes over 400, due to errors caused by tracking 'bad features' and highly erroneous triangulation. The maximum error without the 'reprojection filter' was found to be 768.892 whereas the maximum using the technique was just 9.525 pixels.

Number of Tracked Features: The number of features that were tracked as good correspondences for obtaining the camera pose during each frame of the algorithm, were observed for both methods. Figure 12.2 show the number of features tracked at each frame with and without using the 'reprojection filter' technique.

The number of features in the beginning is less as the baseline is smaller than the threshold (100 wcu, refer Table 11.1). The number of tracked features are obviously less for the 'reprojection filter' method, since more features are filtered out.

The sudden increase in the number of features indicate the points were the algorithm updates the number of features. The algorithm adds new features if enough features are not being tracked and enough baseline is available between the two frames (current and previous). The number of times the features are updated are more when the 'reprojection filter' technique is not used. This can be explained by considering the wrong camera pose estimations. Due to wrong and noisy estimations, the method estimates poses that are far apart in subsequent frames. Therefore the baseline will appear to be more than the threshold even when it actually is not. This leads to the algorithm finding new features every time the number of tracked features falls below the threshold (set as 200, refer Table 11.1). On the other

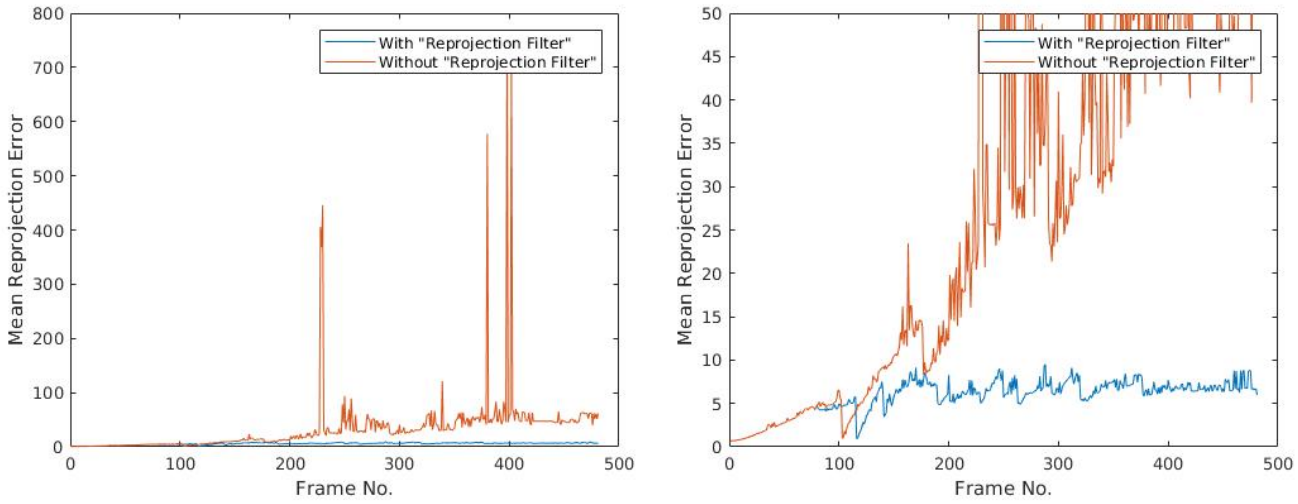


Figure 12.1: Mean Reprojection Error per Frame for the 2 Methods
Left: Full Plot. Right: Magnified view (y axis range limited for better visualization)

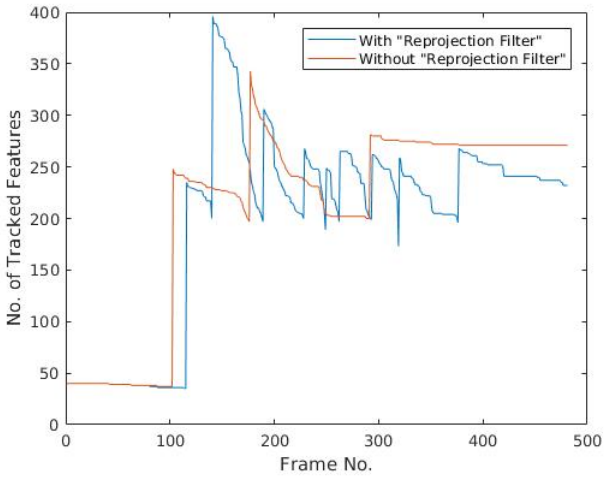


Figure 12.2: Number of Features Tracked per Frame for the Two Methods

hand, using the 'reprojection filter' technique gives a better camera pose estimation, and it does not update features when the baseline is not large enough.

Computation time: The computation time required for running the algorithm on the dataset using the two methods were tested. It was observed that using the 'reprojection filter' technique was computationally slightly more expensive (24.35 seconds) than running the algorithm without it (23.81s). This is expected, as the 'reprojection filter' method had an extra step in each frame, which checked the reprojection error of all the tracked features in that frame, and removed the 'bad' features. The reason that the time difference is not larger is because the 'reprojection filter' method tracks more number of features, and also updates the features more often, as was shown in the previous experiment Since, the difference in time (0.54s) is small, it can be neglected if time is not an important constraint for the task.

Estimated Trajectory: The trajectories estimated after running the algorithm with and without the 'reprojection filter' technique are shown in Figures 12.3 and 12.4. It is clear that the trajectory estimate obtained using the 'reprojection filter' technique is better defined and more consistent to the semicircular path of the camera that was observed by visualizing the image sequence (the true trajectory of the camera is not available). This is because features having high reprojection errors were tracked in many frames in case of the latter method, which gave noisy information and resulted in bad pose estimation at those frames. Another observation made in the estimated trajectory is that there is there is increased uncertainty in the pose estimation as the algorithm progresses. This is due to drift which is common in

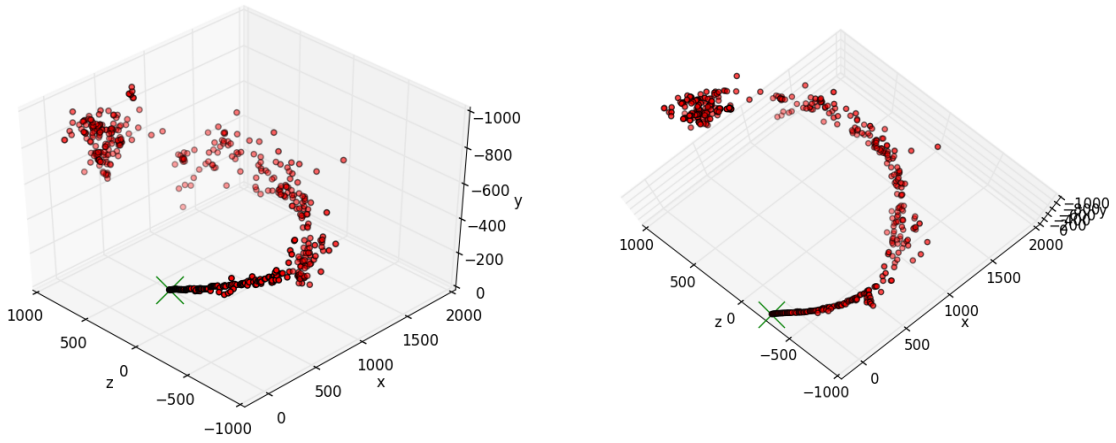


Figure 12.3: Two Views of the Trajectory Estimated Using the 'Reprojection Filter' Technique (The green 'X' marks the starting point of the trajectory)

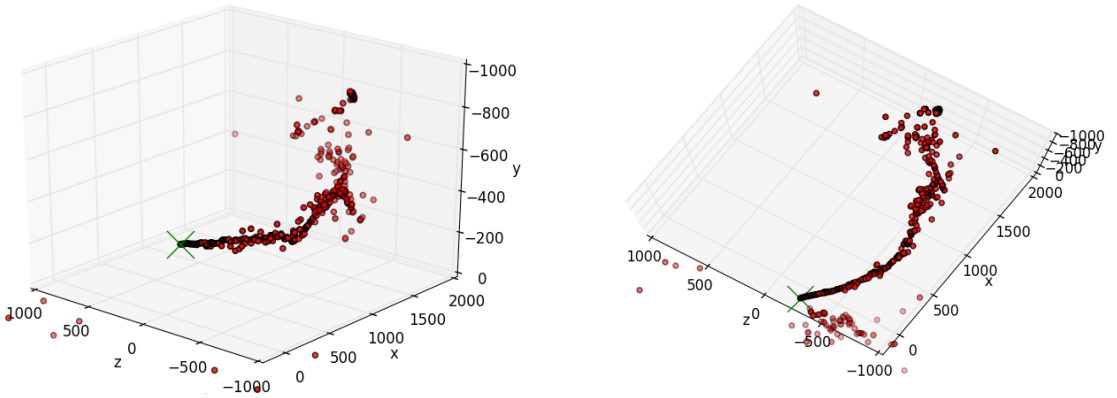


Figure 12.4: Two Views of the Trajectory Estimated Without Using the 'Reprojection Filter' Technique (The green 'X' marks the starting point of the trajectory)

visual odometry algorithms, and occur due to progressive accumulation of pose estimation errors. The camera pose estimation at each frame is obviously not the exact position of the camera (due to various noise factors and errors and approximations in the calculations). The small errors at each frame accumulate over time and causes larger errors in the pose estimation at subsequent frames. This phenomenon is called drift, and can be corrected by giving additional information in the form of additional sensors [6], providing markers as absolute references [57], combining with SLAM algorithm to improve localisation [1], or using trajectory optimization methods such as bundle adjustment.

12.2 Comparison With the Winning Code of ISMAR 2015

The code performance was compared with the winner code of the ISMAR 2015 Offline Tracking Competition. The algorithm seemed to perform better than the winning code in terms of reprojection error, the accuracy of the features being detected as matches in subsequent frames, and speed. A video comparing the performance of the two codes can be seen here: `https://youtu.be/TxMpHycdhc0` (Figure 12.5).

It is clear from the video that the features tracked (depicted by coloured markers) in the winning code are not always consistent with their positions in the world. The features float away from their initial position in subsequent frames. The VO code developed for the project however, removes these points if they drift too much across frames. The 'reprojection filter' makes sure that only features which can

be reprojected back into the image reasonably well are considered. This would mean that the number of features tracked will be fewer. However, it can be seen in the video that the positions of the tracked features are much more stationary in the world frame, and the drift is much less. It can also be seen that more features are discarded compared to the winning code.

The video also proves that the VO code developed for this project is faster than the winning code (by about 13 seconds). This is considering the fact that the former was written in python and the latter in C++, which is supposed to be much faster. One of the factors for this result could be the fact the number of features tracked is much more in the winning code (721 features in the last frame) compared to the developed VO code (301 in the last frame).

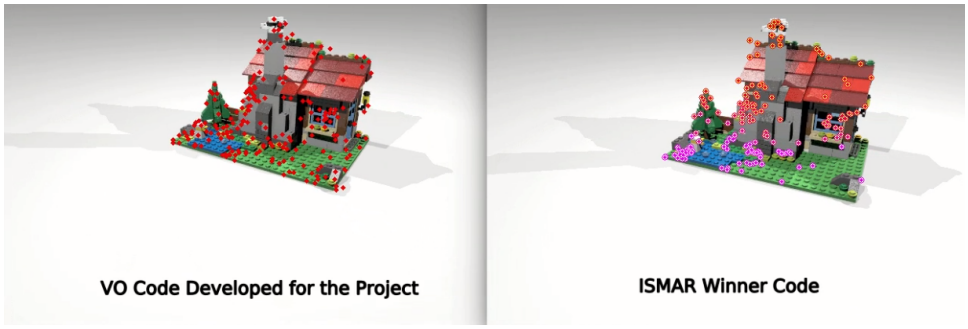


Figure 12.5: A Frame from the Video Comparing the performance of the Developed VO Code with that of the Winning Code of ISMAR 2015 Tracking Competition
(Tracked features are shown as the coloured dots)

The winning code does an additional step of optimisation at the end, called 'bundle adjustment', over the last 24 frames. Bundle adjustment is a technique for optimising the feature correspondences in a selected window of image frames to minimise the reprojection errors [58]. For each frame in the window , it iteratively estimates the projection matrix $\mathbf{P}$ and the 3D points ($\mathbf{X}_0$, $\mathbf{X}_1$,..., $\mathbf{X}_n$), so that the reprojection error for the frame is minimised. This is usually done as a last step over a window of the last few frames since the method is computationally expensive and is difficult to implement in real-time. The mean reprojection error for the winning code is claimed to be 161.188 before and 117.599 after applying bundle adjustment. These values are much higher than the final mean reprojection error obtained while using the VO code of this project (5.769 pixels).

Chapter 13

Conclusion and Future Work

A visual odometry code was developeds to obtain the trajectory of a camera using only the sequence of images captured by it, and a few keypoints whose 3D world coordinates are known. If enough keypoints are matched in the first image, the algorithm can track these features in the subsequent images and use their 3D-2D correspondences to estimate the camera pose at each frame. If it loses track of features due to any reason, the algorithm can find new feature correspondences which can be used for pose estimation. Once all the image frames in the sequence are analysed, the algorithm can plot a trajectory using the estimated poses at each frame.

Various techniques from computer vision were used for designing the visual odometry algorithm. The performance of the algorithm was analaysed using the mean reprojection error at the end of the visual odometry. It was found that it depends on various parameters of the algorithm. The best values of the parameters were chosen by comparing the reprojection errors for different values.

A method to improve the performance of the visual odometry algorithm was introduced in the project. This method, called the 'reprojection filter' technique, checks for feature correspondences which gave large reprojection filter and discards them, thereby improving the final mean reprojection error of the visual odometry. The visual odometry code was tested on the dataset provided by ISMAR for their Offline Tracking Competition of 2015. The performance of the algorithm with and without the 'reprojection filter' was experimentally compared using this dataset and the results presented. It was shown that the 'reprojection filter' technique was able to bring down the total mean reprojection error by more than 82%, at the cost of tracking comparatively fewer features and being slightly slower. However, if the scene is relatively textured, new good features can be easily obtained and discarding 'bad' features can only improve the performance of the algorithm, and the difference in computational speed for this task was negligibly small (0.54s) to be a significant demerit.

Finally, the algorithm performance was compared with that of the winner of the ISMAR tracking competition. It was found that the algorithm developed for the project using the 'reprojection filter' performed much better than the competition-winning code in terms of reprojection error, speed and accuracy of feature tracking across frames.

The development of the algorithm began with the intention of applying it on the AR.Drone for image based visual servoing (IBVS). Therefore, the algorithm was tried on the AR.Drone using the camera on its underside. The camera was calibrated using the ROS 'camera_calibration' node [59]. A set of 30 feature patches and their position in the world coordinates (obtained using the motion capture system) was provided. The high-level PID controller developed in Part I was used for moving the drone to a destination point. The objective of the task was to use the VO algorithm to estimate the trajectory followed and compare the result with the ground truth provided by the Ground Truth System. However, it was observed that the results were not good enough to be presented. The pose estimates were too noisy and the algorithm often collapsed as it lost track of all the features in the frame. Due to constraints in time, the algorithm could not be tuned further for proper application on the AR.Drone.

Various observations were however made while trying to implement visual odometry on the drone. It was noticed that the controller for the drone had to made more stable, since the vibrations of the drone affected the quality of the images captured by the camera, and resulted in the algorithm losing all the tracked features. The low resolution of the camera (320×240 pixels) could be a factor for the poor results. Thus, future work on the topic would focus on using the higher-resolution front camera (720p) and a more textured environment. It was also noticed that the VO algorithm seemed to be too slow and noisy for real-time implementation on the drone. Improvements in terms of speed and better handling

of noise and unstable camera motions should be made before the algorithm can be applied on a dynamic real-world robot.

The performance of the algorithm can be improved by using better feature detection methods. The algorithm currently uses SIFT detection technique for finding new features, which is slower than many other available methods. Detection algorithms such as SURF (Speeded-Up Robust Features) [60], BRIEF (Binary Robust Independent Elementary Features) [61], and ORB (Oriented FAST and Rotated BRIEF) [62], have been proved to be faster and more efficient than SIFT [63]. Localisation of pose for visual odomery can be improved by providing some additional information for localisation, such as placing markers in the environment [64, 17], using a previously-recorded map [65], or using an additional depth sensor [66, 67]. Probabilistic approach for pose and trajectory estimation can help get more accurate results. A visual odometry algorithm using steps similar to those in this project was applied with probabilistic estimations in [68]. The algorithm was implemented on a camera mounted on a moving car in an urban environment. The results presented showed that the estimations were much more robust and accurate due to the probabilistic modeling of estimates and noise. Another possible improvement to be considered is windowed bundle adjustment, as was implemented in the ISMAR-winning code. This would refine the camera trajectory to produce more accurate estimates [46].

Future work is directed towards implementing real-time visual servoing on the AR.Drone as a first step towards visual SLAM. The main area to focus on is improving the performance of the visual odometry algorithm for proper real-world applications by using probabilistic methods to improve estimations and more efficient computer vision techniques for robustness and speed.

References

[1] Brian Williams and Ian Reid. "On combining visual SLAM and visual odometry". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 3494–3500.

[2] O. Bourquardez et al. "Image-Based Visual Servo Control of the Translation Kinematics of a Quadrotor Aerial Vehicle". In: *IEEE Transactions on Robotics* 25.3 (2009), pp. 743–749. ISSN: 1552-3098. DOI: 10.1109/TRO.2008.2011419.

[3] D. Zheng et al. "Image-Based Visual Servoing of a Quadrotor Using Virtual Camera Approach". In: *IEEE/ASME Transactions on Mechatronics* 22.2 (2017), pp. 972–982. ISSN: 1083-4435. DOI: 10.1109/TMECH.2016.2639531.

[4] Leandro Lima Gomes et al. "Unmanned Quadcopter Control Using a Motion Capture System". In: *IEEE Latin America Transactions* 14.8 (2016), pp. 3606–3613.

[5] Daniel Mellinger, Nathan Michael, and Vijay Kumar. "Trajectory generation and control for precise aggressive maneuvers with quadrotors". In: *The International Journal of Robotics Research* 31.5 (2012), pp. 664–674.

[6] Piotr Rudol et al. "Micro unmanned aerial vehicle visual servoing for cooperative indoor exploration". In: *Aerospace Conference, 2008 IEEE*. IEEE. 2008, pp. 1–10.

[7] R Mahony, V Kumar, and P Corke. "Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor. RoboticsAutomationMagazine, 19, 20-32". In: *Mayhew et al. C. Mayhew, R. Sanfelice, and A. Teel. On path-lifting mechanisms and unwinding in quaternion-based attitude control. Automatic Control, IEEE Transactions on. PP (99)* (2012), pp. 1–1.

[8] Jacobo Jiménez Lugo and Andreas Zell. "Framework for autonomous on-board navigation with the AR. Drone". In: *Journal of Intelligent & Robotic Systems* 73.1-4 (2014), pp. 401–412.

[9] Martin Saska et al. "Low cost mav platform ar-drone in experimental verifications of methods for vision based autonomous navigation". In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 2012, pp. 4808–4809.

[10] ardrone autonomy. *ROS.org*. 2015. URL: http://wiki.ros.org/ardrone_autonomy.

[11] Oqus. *Qualysis*. 2017. URL: http://www.qualisys.com/cameras/oqus/.

[12] Byung-Yoon Lee, Hae-In Lee, and Min-Jea Tahk. "Analysis of adaptive control using on-line neural networks for a quadrotor uav". In: *Control, Automation and Systems (ICCAS), 2013 13th International Conference on*. IEEE. 2013, pp. 1840–1844.

[13] Samir Bouabdallah, Andre Noth, and Roland Siegwart. "PID vs LQ control techniques applied to an indoor micro quadrotor". In: *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. Vol. 3. IEEE. 2004, pp. 2451–2456.

[14] Daniel Mellinger and Vijay Kumar. "Minimum snap trajectory generation and control for quadrotors". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2520–2525.

[15] Jeremy H Gillula et al. "Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 1649–1654.

[16] Simon Zingg et al. "MAV navigation through indoor corridors using optical flow". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 3361–3368.

[17] M. Blsch et al. "Vision based MAV navigation in unknown and unstructured environments". In: *2010 IEEE International Conference on Robotics and Automation*. 2010, pp. 21–28. DOI: 10.1109/ROBOT.2010.5509920.

[18] Ascending Technologies. 2012. URL: http://www.asctec.de/.

[19] Kiam Heong Ang, Gregory Chong, and Yun Li. "PID control system analysis, design, and technology". In: *IEEE transactions on control systems technology* 13.4 (2005), pp. 559–576.

[20] Andrew Zulu and Samuel John. "A review of control algorithms for autonomous quadrotors". In: *arXiv preprint arXiv:1602.02622* (2016).

[21] Yun Li, Kiam Heong Ang, and Gregory CY Chong. "PID control system analysis and design". In: *IEEE Control Systems* 26.1 (2006), pp. 32–41.

[22] Introduction: PID Controller Design. *Control Tutorials for MATLAB and Simulink*. URL: http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID.

[23] tf. *ROS.org*. 2017. URL: http://wiki.ros.org/tf.

[24] P Gawthrop. "Self-tuning PID controllers: Algorithms and implementation". In: *IEEE Transactions on Automatic Control* 31.3 (1986), pp. 201–209.

[25] Tim Wescott. "PID without a PhD". In: *Embedded Systems Programming* 13.11 (2000), pp. 1–7.

[26] Lemniscate of Gerono. *Wikipedia*. 2016.

[27] Gabriel Hoffmann et al. "Quadrotor helicopter flight dynamics and control: Theory and experiment". In: *AIAA Guidance, Navigation and Control Conference and Exhibit*. 2007, p. 6461.

[28] D. Nister, O. Naroditsky, and J. Bergen. "Visual odometry". In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.* Vol. 1. 2004, I–652–I–659 Vol.1. DOI: 10.1109/CVPR.2004.1315094.

[29] Liz Murphy et al. "Experimental comparison of odometry approaches". In: *The International Symposium on Experimental Robotics (ISER)*. Ed. by Oussama Khatib. Quebec City, Canada: Springer, 2013, pp. 877–890. DOI: 10.1007/978-3-319-00065-7_58. URL: http://eprints.qut.edu.au/52842/.

[30] Guilherme N DeSouza and Avinash C Kak. "Vision for mobile robot navigation: A survey". In: *IEEE transactions on pattern analysis and machine intelligence* 24.2 (2002), pp. 237–267.

[31] Richard Steffen and Wolfgang Förstner. "On visual real time mapping for unmanned aerial vehicles". In: *21st Congress of the International Society for Photogrammetry and Remote Sensing (ISPRS)*. 2008, pp. 57–62.

[32] Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. "Monocular-SLAM–based navigation for autonomous micro helicopters in GPS-denied environments". In: *Journal of Field Robotics* 28.6 (2011), pp. 854–874.

[33] R. I. Hartley and A. Zisserman. "Camera Models". In: *Multiple View Geometry in Computer Vision*. 2nd ed. New York: Cambridge University Press, ISBN: 0521540518, 2004, pp. 266–290.

[34] Olivier Faugeras. *Three-dimensional computer vision: a geometric viewpoint*. MIT press, 1993.

[35] R. I. Hartley and A. Zisserman. "Computation of the Trifocal Tensor T". In: *Multiple View Geometry in Computer Vision*. 2nd ed. New York: Cambridge University Press, ISBN: 0521540518, 2004, pp. 400–403.

[36] Yihong Wu and Zhanyi Hu. "PnP Problem Revisited". In: *J. Math. Imaging Vis.* 24.1 (Jan. 2006), pp. 131–141. ISSN: 0924-9907. DOI: 10.1007/s10851-005-3617-z. URL: http://dx.doi.org/10.1007/s10851-005-3617-z.

[37] Thomas Petersen. "A comparison of 2d-3d pose estimation methods". MA thesis. Aalborg University, Aalborgb, 2008.

[38] R. I. Hartley and A. Zisserman. "Computation of the Camera Matrix P". In: *Multiple View Geometry in Computer Vision*. 2nd ed. New York: Cambridge University Press, ISBN: 0521540518, 2004, pp. 178–193.

[39] Davide Scaramuzza and Friedrich Fraundorfer. "Visual Odometry: Part I The First 30 Years and Fundamentals". In: ().

[40] Elan Dubrofsky. "Homography estimation". PhD thesis. University of British Colombia (Vancouver), 2009.

[41] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: http://doi.acm.org/10.1145/358669.358692.

[42] Xiao-Shan Gao et al. "Complete Solution Classification for the Perspective-Three-Point Problem". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 25.8 (Aug. 2003), pp. 930–943. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2003.1217599. URL: http://dx.doi.org/10.1109/TPAMI.2003.1217599.

[43] R. I. Hartley and A. Zisserman. "Epipolar Geometry and the Fundamental Matrix". In: *Multiple View Geometry in Computer Vision*. 2nd ed. New York: Cambridge University Press, ISBN: 0521540518, 2004, pp. 239–261.

[44] Richard Hartley, Rajiv Gupta, and Tom Chang. "Stereo from uncalibrated cameras". In: *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*. IEEE. 1992, pp. 761–764.

[45] R. I. Hartley and A. Zisserman. "Structure Computation". In: *Multiple View Geometry in Computer Vision*. 2nd ed. New York: Cambridge University Press, ISBN: 0521540518, 2004, pp. 310–324.

[46] Zhinan Xu. "Stereo Visual Odometry with Windowed Bundle Adjustment". MA thesis. 2015.

[47] R. I. Hartley and A. Zisserman. "Computation of the Fundamental Matrix F". In: *Multiple View Geometry in Computer Vision*. 2nd ed. New York: Cambridge University Press, ISBN: 0521540518, 2004, pp. 279–309.

[48] Roberto Brunelli. *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley Publishing, 2009. ISBN: 0470517069, 9780470517062.

[49] John P Lewis. "Fast template matching". In: *Vision interface*. Vol. 95. 120123. 1995, pp. 15–19.

[50] John P Lewis. *Fast Normalized Cross-Correlation*. 1995.

[51] Bo Yi et al. "A fast matching algorithm with feature points based on NCC". In:

[52] Constance S Royden and Kathleen D Moore. "Use of speed cues in the detection of moving objects by moving observers". In: *Vision research* 59 (2012), pp. 17–24.

[53] Steven S. Beauchemin and John L. Barron. "The computation of optical flow". In: *ACM computing surveys (CSUR)* 27.3 (1995), pp. 433–466.

[54] Bruce D Lucas, Takeo Kanade, et al. "An iterative image registration technique with an application to stereo vision". In: (1981).

[55] David G Lowe. "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[56] Nguyen-Khang Pham, Annie Morin, and Patrick Gros. "Boosting of factorial correspondence analysis for image retrieval". In: *Content-Based Multimedia Indexing, 2008. CBMI 2008. International Workshop on*. IEEE. 2008, pp. 224–229.

[57] Fabrizio Lamberti et al. "Mixed marker-based/marker-less visual odometry system for mobile robots". In: *International Journal of Advanced Robotic Systems* 10.5 (2013), p. 260.

[58] Bill Triggs et al. "Bundle adjustment - a modern synthesis". In: *International workshop on vision algorithms*. Springer. 1999, pp. 298–372.

[59] camera calibration. *ROS.org*. 2015. URL: http://wiki.ros.org/camera_calibration.

[60] Herbert Bay et al. "Speeded-up robust features (SURF)". In: *Computer vision and image understanding* 110.3 (2008), pp. 346–359.

[61] Michael Calonder et al. "Brief: Binary robust independent elementary features". In: *Computer Vision–ECCV 2010* (2010), pp. 778–792.

[62] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2564–2571.

[63] Luo Juan and Oubong Gwun. "A comparison of sift, pca-sift and surf". In: *International Journal of Image Processing (IJIP)* 3.4 (2009), pp. 143–152.

[64] Daniel Eberli et al. "Vision based position control for MAVs using one single circular landmark". In: *Journal of Intelligent & Robotic Systems* 61.1-4 (2011), pp. 495–512.

[65] Tomáš Krajník et al. “AR-drone as a platform for robotic research and education”. In: *International Conference on Research and Education in Robotics*. Springer. 2011, pp. 172–186.

[66] Albert S. Huang et al. “Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera”. In: *Robotics Research : The 15th International Symposium ISRR*. Ed. by Henrik I. Christensen and Oussama Khatib. Cham: Springer International Publishing, 2017, pp. 235–252. ISBN: 978-3-319-29363-9. DOI: 10.1007/978-3-319-29363-9_14. URL: http://dx.doi.org/10.1007/978-3-319-29363-9_14.

[67] Paolo Stegagno et al. “Vision-based autonomous control of a quadrotor uav using an onboard rgb-d camera and its application to haptic teleoperation”. In: *IFAC Proceedings Volumes* 46.30 (2013), pp. 87–92.

[68] Ignacio Parra et al. “Robust visual odometry for vehicle localization in urban environments”. In: *Robotica* 28.03 (2010), pp. 441–452.

[69] Bert M Haralick et al. “Review and analysis of solutions of the three point perspective pose estimation problem”. In: *International journal of computer vision* 13.3 (1994), pp. 331–356.

[70] Duane C Brown. “Decentering distortion of lenses”. In: *Photogrammetric Engineering and Remote Sensing* (1966).

[71] Seven Beauchemin and Ruzena Bajcsy. “Modelling and removing radial and tangential distortions in spherical lenses”. In: *Multi-Image Analysis* (2001), pp. 1–21.

[72] Camera Calibration and 3D Reconstruction. *OpenCV*. 2017. URL: http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html.

[73] Kenneth Levenberg. “A method for the solution of certain non-linear problems in least squares”. In: *Quarterly of applied mathematics* 2.2 (1944), pp. 164–168.

The references have been prepared using the Modern Language Association of America (MLA) format, *MLA Handbook, Eighth Edition.*

Appendix A

Additional Notes

A.1 Camera Calibration

The camera calibration matrix $\mathbf{K}$, as explained in section 9.1, contains five intrinsic parameters of the camera. These parameters are unique for the camera and should be estimated correctly for obtaining the correct projection of the object onto the image plane. Additionally, real camera lenses typically suffer from non-linear lens distortion such as radial distortion, which causes straight lines to be mapped as curves. Hence, like skew, radial distortion is a deformity in image projection. However, unlike skew, these parameters are not contained in the calibration matrix.

The process of finding these intrinsic parameters of a given camera is called calibration. Calibration may sometimes be used to refer to the process of estimating the entire projection matrix (i.e. both the intrinsic and extrinsic parameters of the camera). But since the extrinsic parameters are not fixed and would depend on factors such as the coordinate frame considered, position and orientation of the camera with respect to it etc, in this report, the term calibration is used in terms of estimating the intrinsic parameters alone, i.e. the values in the calibration matrix $\mathbf{K}$, and other distortion coefficients.

Various methods have been developed and proposed for estimating the projection matrix [38]. However, the most common method of approach for estimating the calibration parameters of a camera is using a 'calibration object' or 'pattern' with known geometry [69]. If at least three pairs of correspondences are known, the relation between the image coordinates and the 3D object points can be used to solve the mapping of the 3D scene to 2D image, i.e. to estimate the projection matrix $\mathbf{P}$ [41]. The problem of estimating the projection matrix when n 3D object points and their corresponding 2D image projections are known, is called the Perspective-n-Points problem, and is explained in section 9.3.

If the extrinsic parameters are known, the calibration matrix can be computed directly using equation (9.4),

$$\mathbf{K} = P_{ext}^{-1}\mathbf{P}$$

where $P_{ext} = [R|t]$ is the extrinsic parameter matrix.

There will be some amount of distortion in even the best camera available. Moreover, some lenses such as fish-eye lenses, or wide-angle cameras have radial distortion due to their wider field of view. These distortions manifest in the form of 'barrel' and 'pincushion' effects. Radial distortion can be corrected using Brown's distortion model [70], also known as the plumb-bob model. Tangential distortion is caused by imperfect parallel alignment between the lenses and the image sensor. Due to improvement in technology, most camera systems do not suffer too much from tangential distortions. However, for unbiased results, the distortion model considers both tangential and radial distortions. The modelled distortions can be accounted for using the following [71]:

$$x_{\mathrm{d}} = x_{\mathrm{u}}(1 + K_1 r^2 + K_2 r^4 + \cdots) + (P_2(r^2 + 2x_{\mathrm{u}}^2) + 2P_1 x_{\mathrm{u}} y_{\mathrm{u}})(1 + P_3 r^2 + P_4 r^4 + \cdots)$$

$$y_{\mathrm{d}} = y_{\mathrm{u}}(1 + K_1 r^2 + K_2 r^4 + \cdots) + (P_1(r^2 + 2y_{\mathrm{u}}^2) + 2P_2 x_{\mathrm{u}} y_{\mathrm{u}})(1 + P_3 r^2 + P_4 r^4 + \cdots)$$

where: $(x_{\mathrm{d}},\ y_{\mathrm{d}})$ = distorted image point as projected on image plane using specified lens,
$(x_{\mathrm{u}},\ y_{\mathrm{u}})$ = undistorted image point as projected by an ideal pin-hole camera,
$(x_{\mathrm{c}},\ y_{\mathrm{c}})$ = distortion center (assumed to be the principal point),
$K_n = n^{\mathrm{th}}$ radial distortion coefficient,

$P_n = n^{\text{th}}$ tangential distortion coefficient
$r = \sqrt{(x_{\text{u}} - x_{\text{c}})^2 + (y_{\text{u}} - y_{\text{c}})^2}$

The camera calibration on the AR.Drone for this project was done using the ROS package *'camera_calibration'*, which is based on the OpenCV camera calibration method [72, 59]. A checkerboard of known dimensions was used for calibrating the cameras on the AR.Drone. The position of the squares on the board is known. To compute the calibration parameters accurately, several pictures of the board shown at different positions and orientations were taken, ensuring that the field of view of the camera is filled as much as possible. The input data are the 2-D positions of the corners of the squares on the board and their corresponding pixel coordinates in each image. Since the camera is left stationary, the distortion and intrinsic parameters are generated by calculating a homography transformation between the chessboard and the image planes.

A.2 Levenberg-Marquardt Optimization

The DLT method for estimating the projection matrix **P** (section 9.3.1) does not always produce the best result. The rotation matrix R in the projection equation (the first three columns of **P**) should have all the properties of any other rotation matrix such as unit determinant, orthogonality etc. Since DLT finds a solution for the nine numbers in the R part of the **P** (a rotation matrix has only three degrees of freedom), the resultant is not necessarily a rotation matrix. Furthermore, DLT does not minimize the required cost function (reprojection error). However, there are principled ways to iteratively change the values of R and t so that the reprojection error decreases. The Levenberg–Marquardt (LM) is one such method.

The LM algorithm is a minimization method that can be used to solve non-linear least square problems. It combines the advantages of gradient-descent and Gauss-Newton methods for solving non-linear least squares problems [73]. The method adopted at each 'step' (an iteration) depends on whether the algorithm is converging to a local minima or not. It starts with an initial guess of the solution, which is adjusted only for downhill steps. The LM algorithm has a 'damping factor' λ which is adjusted at each iteration. If the reduction is rapid, a smaller value of λ is used, which would bring the algorithm closer to the Gauss-Newton method. On the other hand, if the reduction in the residual is insufficient, λ can be increased, simulating the gradient descent function by guiding it a step closer in the direction of the gradient-descent. This way the algorithm always descends in the parameter space, ensuring convergence as long as the initial guess is good.

Appendix B

Code Repositories

- *PID Controller*: The code developed for the PID control of the AR.Drone is available at: `https://github.com/justagist/pid_control_ardrone.git`

  It depends on the ROS drvier for the AR.Drone, ardrone_autonomy (`http://wiki.ros.org/ardrone_autonomy`).

  The controller requires a motion capture system and another computer sending the data wirelessly.

  It can be used without a motion capture system, using a simulator for the AR.Drone: 'tum_simulator' available at `http://wiki.ros.org/tum_simulator`

- *The VO Code Developed*: The VO code developed is available at `https://github.com/justagist/VisualOdometry_miniproject.git`. It includes the ISMAR dataset that was used for this project.

- *The ISMAR 2015 Winner Code*: The winner code was obtained with permission from: `https://github.com/eaa3/STAM.git`. It can be used with the same dataset, or from: `https://github.com/eaa3/STAM_DATA.git`

Appendix C

Statement of Information Search Strategy

Most of the reference materials were obtained from IEEE Explore, Google Scholar or using the online and offline services of the University Library. Relevant articles and books were also obtained from citations provided in online sources such as Wikipedia. The book '*Multiple View Geometry in Computer Vision*', by R.I. Hartley and A. Zisserman was used extensively for understanding the fundamentals of computer vision.

Other references include thesis reports and websites such as OpenCV, ROS, and product websites (motion control system and quadcopter). Certain images were obtained from online sources to explain the concepts and theory required for the project. Their sources have been clearly referenced in the captions.

Appendix D

Mini-Project Declaration

The University of Birmingham

School of Computer Science

Second Semester Mini-Project: Declaration

This form is to be used to declare your choice of mini-project. Please complete all three sections and upload an electronic copy of the form to Canvas: https://canvas.bham.ac.uk/courses/21891

Deadline: 12 noon, 24 January 2017

1. Project Details

Name: Saif Sidhik

Student number: 1734904

Mini-project title: Autopilot System in Quadcoptors

Mini-project supervisor: Professor Jeremy L Wyatt

---

2. Project Description

The following questions should be answered in conjunction with a reading of your programme handbook.

Aim of mini-project	• To design and implement a high-level position controller on an AR.Drone 2.0 quadcoptor for autonomous flight along a defined trajectory.

Objectives to be achieved	• Develop a high-level control system that would:
	  ◦ Enable autonomous flight of the AR.Drone 2.0 to desired waypoints.
	  ◦ Achieve accurate final pose of the quadcoptor in terms of position and orientation.
	• Use the 'ground truth' trajectory obtained from motion capture system to quantitatively evaluate the estimated trajectory produced by the developed system.

Project management skills
 	
Briefly explain how you will devise a management plan to allow your supervisor to evaluate your progress
	• Understanding low-level control systems used in remote-controlled micro aerial vehicles.
	• Learning the basics of remotely controlling the AR.Drone 2.0. This would include learning the functions of the various commands used for sending messages via ROS to the quadcopter system, and being able to control the drone wirelessly from a Ground Station (laptop).
	• Understanding and learning to use the Motion Capture System. The 'ground truth' obtained from the motion capture system will be used for improving and evaluating the performance of the controller system.
	• Develop a high-level controller which can correct errors in the trajectory using the incoming data from the on-board cameras and sensors.
	• Conduct indoor lab experiments equipped with a motion capture system in order to:
	  ◦ Quantitatively evaluate the performance of the proposed controller.
	  ◦ Qualitatively demonstrate the effectiveness of the controller in following various trajectories as a final demo.

Systematic literature skills **Briefly explain how you will find previous relevant work**	The project will be mainly based on the paper by Jacobo Jiménez Lugo and Andreas Zell - "Framework for Autonomous On-board Navigation with the AR.Drone". Few related papers and literature were collected from several online sources such as IEEE and Google Scholar. Further materials will be collected from online sources and the college library.

Communication skills **What communication skills will you practise during this mini-project?**	Formal presentation of scientific research, experiments and observations through a scientific report.

3. Project Ethics Self-Assessment Form

Please answer YES/NO to the following questions:

- **Does the research involve contact with NHS staff or patients?**

  NO
- **Does the research involve animals?**

  NO

- **Will any of the research be conducted overseas?**

  NO

- **Will any of the data cross international boarders?**

  NO

- **Are the results of the research project likely to expose any person to physical or psychological harm?**

  NO

- **Will you have access to personal information that allows you to identify individuals, or to corporate or company confidential information (that is not covered by confidentiality terms within an agreement or by a separate confidentiality agreement)?**

  NO

- **Does the research project present a significant risk to the environment or society?**

  NO

- **Are there any ethical issues raised by this research project that in the opinion of the PI or student require further ethical review? If you are unsure, consider whether the project has the potential to cause stress or anxiety in the people you are involving.**

  NO

- **Human subjects can be involved as users, providers of system requirements, testers, for evaluation, or similar such activities. Does the experiment involve the use of human subjects in any other capacity? If you are unsure, answer YES.**

  NO

- **Answer YES if ANY of the following are true**

  *** the project has the potential to cause stress or anxiety in the people you are involving, e.g. it addresses potentially sensitive issues of health, death, religion, self-worth, financial security or other such issues**

  *** the project involves people under 18**

*** the project involves a lack of consent or uninformed consent**

*** the project involves misleading the subjects in any way**
If the project's involvement of people relates only to straightforward information gathering, requirements specification, or simple usability testing, then you can indicate NO.

NO

- **If any of the above questions is answered YES, or you are unsure if further review is needed (the first point is usually a good indicator - may cause stress or anxiety) then you should refer it for review.**
- **Further review will involve the School Ethics Officer meeting with the supervisor and ideally the student, reviewing the project, and suggesting any procedures necessary to ensure ethical compliance.**

DECLARATION

By submitting this form, I declare that the questions above have been answered truthfully and to the best of my knowledge and belief, and that I take full responsibility for these responses. I undertake to observe ethical principles throughout the research project and to report any changes that affect the ethics of the project to the University Ethical Review Committee for review.