Monocular Vision Based Indoor Simultaneous Localisation and Mapping for Quadrotor Platform

by
Saurav Agarwal

Submitted for the degree of M.Sc by Research

Department of Aerospace Sciences
School of Engineering
Cranfield University
Bedfordshire, UK

January, 2012

© January, 2012
Saurav Agarwal
All Rights Reserved

Acknowledgement

I would like to thank my supervisor Dr. Al Savvaris whose constant motivation and guidance has been crucial to my work. His enthusiasm and support has been pivotal to my progress. I am also grateful to my colleague Samer Aldaher for helping me in developing electrical systems for the quadrotor. My sincere thanks to Francis Salama for helping me in conducting various hardware tests and in assembling the quadrotor. I would also like to express my gratitude to Ramey Jamil for his critical review of my written work. I am also thankful to Dr. Samuel Lazarus for verifying my progress at every stage and helping me with understanding key concepts. Many thanks to Mudasser Lone for aiding me in understanding various concepts. Last but not the least, I would like to thank my parents, my brother and my best friend Apoorva Singh for their unwavering support and faith in me.

Abstract

An autonomous robot acting in an unknown dynamic environment requires a detailed understanding of its surroundings. This information is provided by mapping algorithms which are necessary to build a sensory representation of the environment and the vehicle states. This aids the robot to avoid collisions with complex obstacles and to localize in six degrees of freedom i.e. x, y, z, roll, pitch and yaw angle. This process, wherein, a robot builds a sensory representation of the environment while estimating its own position and orientation in relation to those sensory landmarks, is known as Simultaneous Localisation and Mapping (SLAM).

A common method for gauging environments are laser scanners, which enable mobile robots to scan objects in a non-contact way. The use of laser scanners for SLAM has been studied and successfully implemented. In this project, sensor fusion combining laser scanning and real time image processing is investigated. Hence, this project deals with the implementation of a Visual SLAM algorithm followed by design and development of a quadrotor platform which is equipped with a camera, low range laser scanner and an on-board PC for autonomous navigation and mapping of unstructured indoor environments.

This report presents a thorough account of the work done within the scope of this project. It presents a brief summary of related work done in the domain of vision based navigation and mapping before presenting a real time monocular vision based SLAM algorithm. A C++ implementation of the visual slam algorithm based on the Extended Kalman Filter is described. This is followed by the design and development of the quadrotor platform. First, the baseline specifications are described followed by component selection, dynamics modelling, simulation and control. The autonomous navigation algorithm is presented along with the simulation results which show its suitability to real time application in dynamic environments. Finally, the complete system architecture along with flight test results are described.

Contents

List of Figures

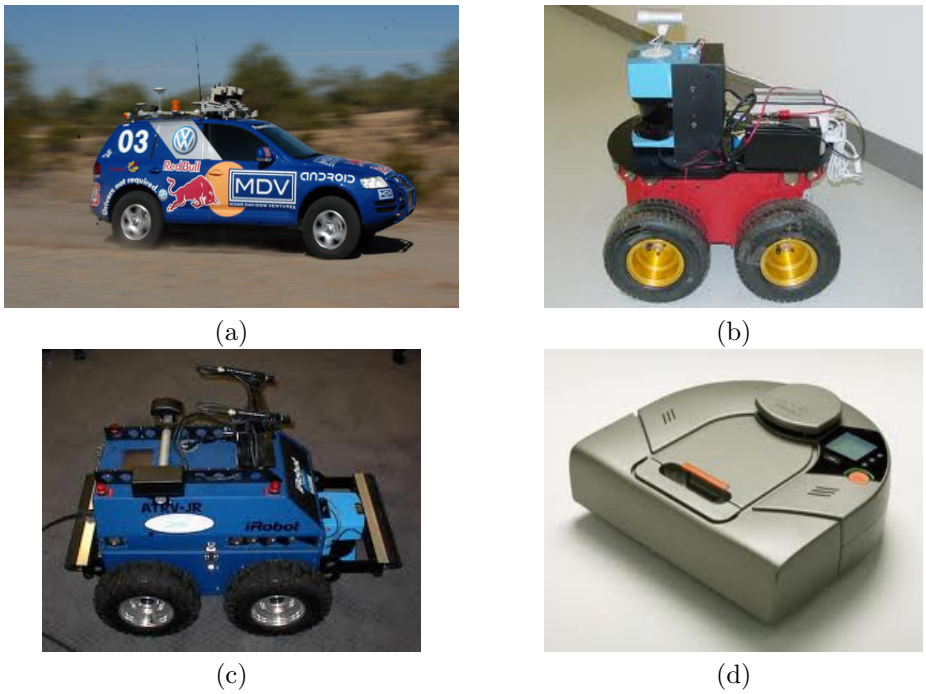
Abbreviations

SLAM	Simultaneous Localisation and Mapping
PDF	Probability Distribution Function
MonoSLAM	Monocular Vision Based Simultaneous Localisation and Mapping
MEMS	Micro Electro-mechanical Sensor
MAV	Micro Aerial Vehicle
EKF	Extended Kalman Filter
GPS	Global Positioning System
RPM	Rounds Per Minute
PWM	Pulse Width Modulation
AGV	Autonomous Ground Vehicle
UGV	Unmanned Ground Vehicle
COTS	Commercial Off the Shelf
C-Space	Configuration Space
V-Space	Velocity Space
TP-Space	Trajectory Parameter Space
W-Space	Work Space

CHAPTER 1

Introduction

If one compares the success and application of robotics within the domain of industrial production to other areas like personal assistance, search and rescue etc., one sees that mobile robots still have a small presence in our society. This paradigm is soon about to undergo a major transformation. Robotic vacuum cleaners [10] capable of mapping a house and cleaning it are now easily available in the market. Experimental robotic cars such as Stanford University's Stanley [43] have traversed hundreds of miles of urban and off-road terrain autonomously. Research robots such as the Pioneer P3-DX have been outfitted with laser scanners and a host of other sensors to successfully map large indoor environments. However, extending the same capabilities to Micro Aerial Vehicles (MAVs) with similar success is a challenge that has not yet been fully tackled. A micro aerial vehicle (MAV) is, by definition, an unmanned aerial vehicle which has significant size and weight restrictions and can either be piloted remotely or fly autonomously.



(a) (b) (c) (d)

Figure 1.1: (a) Stanford's autonomous car Stanley (b) Pioneer P3-DX: A popular research robot in SLAM configuration (c) iRobot's SLAM robot (d) Neato Robotics XV11 autonomous vacuum cleaner

MAVs as shown in fig. 1.2 are being used in several military and civilian applications, in-

(a)

(b)

Figure 1.2: (a) Honeywell RQ-16 T-Hawk (b) Draganfly X6

cluding surveillance, weather observation, disaster relief coordination, and civil engineering inspections. Using Global Positioning System (GPS) aided Micro Electro-Mechanical Sensor (MEMS) based inertial measurement units, researchers have developed MAVs that display a high level of autonomy in outdoor environments without human intervention. Unfortunately, most indoor environments do not have access to global localisation information such as GPS. Autonomous MAVs today are thus very limited in their ability to operate in these areas. Traditionally, unmanned vehicles operating in GPS-denied environments can rely on dead reckoning for localization, but these measurements drift over time. Alternatively, by using on-board environmental sensors, simultaneous localization and mapping (SLAM) algorithms build a map of the environment around the vehicle from sensor data while simultaneously using the data to estimate the vehicle's position. In contrast to ground vehicles, there have been fewer successful attempts to achieve the same results with MAVs. Some of them have been partly successful due to a combination of limited payloads for sensing and computation, coupled with the fast and unstable dynamics of air vehicles.

Figure 1.3: A hotel building damaged by fire

Picture a scenario where a hotel is damaged by an earthquake. Hundreds of guests are trapped inside the damaged structure and there is a distinct possibility of a fire starting due to electrical faults. Such a situation presents an extremely dangerous environment which is difficult for human emergency response teams to penetrate and analyse in a quick and safe manner. A tool would be needed to rapidly provide pictures from in and around the building and map a safe path for rescue workers to search for hostages. An aerial robot capable of flying autonomously in a highly unstructured environment would be the perfect tool for this job. It would easily fly over rough terrain and provide comprehensive coverage of the situation. Such an MAV presents a host of challenges which are detailed in the next section.

1.1 Key Challenges

It is easy to imagine the nature of tasks we could assign to an autonomous MAV but there are numerous challenges to consider. Firstly, there exist two basic issues that any mobile robot must solve in order to achieve a certain degree of autonomy; it should be able to figure out where it is within the world and should be able to drive itself in a safe manner towards a target location. Issues faced greatly depend on low-level issues, such as the kind of employed sensors and high level problems such as computational complexity. The main issues faced by an MAV are:

- **Limited Payload:** For an aerial platform there is the obvious constraint of size and weight. We would require an MAV to have as much autonomy as possible while at the same time having a sufficient range of mobility. In a hovering platform like a quadrotor, most of the energy is consumed to keep the vehicle in the air. Therefore, we need to ensure that the payload demands as little power as possible. Sensors like the ones mounted on the robots shown in Fig. 1.1 are not suitable for an MAV. We are limited to using lightweight laser scanners, miniature cameras and MEMS based IMUs.
- **Limited Computational Resources:** SLAM algorithms are highly computationally demanding even for powerful desktop computers and are therefore not suitable for small embedded computer systems that might be mounted on-board MAVs. It is possible to process the sensory data on a ground control station and have the two systems communicate via a wireless link. However, the network bandwidth becomes a hurdle. Images captured by the onboard camera need to be compressed which adds noise and reduces the amount of information available to analyse. It also adds noise to the signals which is particularly damaging for feature detectors that look for high frequency information such as corners in an image. Additionally, the wireless link would cause a fixed time delay. While this delay can be ignored for slow moving, passively stable ground robots, MAVs have fast and unstable dynamics and a delay in the state estimation would make the platform highly unreliable.
- **Drift prone sensory data:** A ground vehicle has access to wheel odometry whereas an MAV has to rely on highly noisy and drift prone acceleration and angular rate measurements from a gyro. If these measurements are integrated, the position error would grow unbounded. Therefore, an MAV uses relative measurement sensors which allow us to estimate its pose.
- **Rapid Dynamics:** Unlike stable ground vehicles an MAV cannot stand still at a point in space. A ground robot can stand still and take multiple measurements in case its state estimate uncertainty becomes too large. However, an MAV does not have this luxury, it is constantly in a state of motion which means that the motion planning algorithms must be highly adapted to aerial applications. They must ensure that the robot is always viewing known as well as unknown features.
- **Difficult to hold position:** An aerial vehicle's position tends to drift in space while it is flying. Even small perturbations/oscillations can cause the vehicle to drift randomly. Therefore, the controller must have access to accurate and timely state estimates for both position and velocity to enable steady operation.

1.2 Related Work

In recent years, a lot of researchers have attempted to port the capabilities of UGVs to MAVs. Coates et al. [27] have demonstrated terrific autonomous aerobatic manoeuvres on a helicopter by training it using a sub-optimal expert human pilot. Researchers have also shown cooperative swarms of MAVs [42]. While these attempts are all sophisticated pieces of the work, most of them rely on motion tracking systems for localisation indoors. In this work, the aim is to develop a flying robot that can operate autonomously with on-board sensing and computation. This is in contrast to approaches taken by other researchers such as [31] and [32] who have flown indoors using position information from motion capture systems, or external cameras [13]. Blosch et al. [19] have developed a system where they use Parallel Tracking and Mapping developed by Klein et al. [33] to use a downward looking camera mounted on a quadrotor UAV to estimate its pose and trajectory. Sinopoli et al. [40] developed a system for visual navigation by building a 3D collision risk map of the environment and using vision as the obstacle detection sensor.

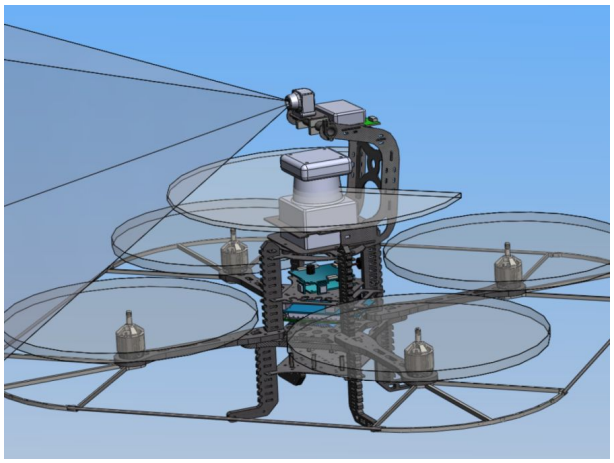


Figure 1.4: Ascending Technologies Pelican Quadrotor [5]

One of the first successful attempts at SLAM using an aerial mapping was by [16]. The MAV shown in fig. 1.4 was developed as a collaboration between researchers at MIT and Ascending Technologies GmbH [5]. Equipped with colour cameras, on-board laser scanner, Atom single board computer and a MEMS based IMU this vehicle was successful in winning the International Aerial Robotics Competition 2009 which required the quadrotor to autonomously enter a hazardous unknown environment through a window, explore the indoor structure without GPS, and search for a visual target. While this system is successful in navigating and localising in a rectilinear environment its capabilities are limited by its 2D sensing.

1.3 Aims and Objective

The work presented in this thesis describes a system that integrates sensing, planning, and control to enable a quadrotor to autonomously explore indoor environments using on-board resources without a priori information. The objectives of this project are as follows:

- Implement Extended Kalman Filter based Monocular SLAM Algorithm
- Design and Development of a quadrotor flying platform
- Hardware integration and bench testing of quadrotor
- Implement sensor fusion and path planning algorithms
- Validate system with real-world experiments

1.4 System Description

A three level control hierarchy is implemented. At the lowest level, a high speed Stability Augmentation System stabilises the roll, pitch and yaw of the quadrotor using data from the IMU. At the second level, the Extended Kalman Filter fuses, incoming data from the camera and IMU to generate accurate state estimates of the vehicle and maintains a map of the visual features. At the third level, the state estimates from the data fusion filter are fed to a reactive navigation algorithm which generates velocity and turn rate commands for the quadrotor to safely traverse to a user selected location while avoiding obstacles.

1.5 Report Layout

Chapter 2 starts with a discussion of some of the successful algorithms developed by various researchers and goes on to present an optical flow based methodology that is tested on a Parrot A.R Drone quadrotor. A visual odometry system is also developed and examined. Chapter 2 discusses the motivation for choosing EKF-Based Visual SLAM and the reasons to build a quadrotor. Chapter 3 presents EKF-Based Monocular SLAM which was used by me for this project. Simulation and real time implementation results of monocular SLAM are also discussed. Chapters 4 & 5 present the design, development and dynamical analysis of the quadrotor. The paradigm of autonomous navigation will be presented in Chapter 6. Chapter 7 will discuss the System Architecture for autonomous operation followed by flight test results. Finally the conclusions and discussion will be presented in chapter 8.

CHAPTER 2

Vision for Micro Aerial Vehicles

Considerable research has been conducted on the use of vision for navigation, localisation and mapping. Angeli et al. [14] have demonstrated a visual loop closure system that uses shape information in a scene to match it with a previously visited point. An optical flow balancing based altitude control system for an MAV in indoor flight was demonstrated by Beyeler et al. [17]. They treat altitude control as an obstacle avoidance problem using the ceiling and floor as references. Garratt et al. [29] simulated an MAV navigating in a cluttered urban environment using optical flow field measurements to calculate the Time to Contact to an obstacle. Their work showed that a highly dynamic light weight MAV using only passive visual sensors could safely navigate such an environment. Celik et al. [23] successfully implemented a monocular vision based navigation, localisation and mapping scheme for a helicopter in structured indoor environments using architectural features and employed the use of a particle filter based approach to SLAM. Tournier et al. [45] implemented a 6 DOF pose estimation algorithm based on Moire patterns. Zing et al. [46] have used optical flow to navigate corridor like environments. A visual odometry system that estimates the motion of a camera based on video input using stereo correspondence has been developed by Nister [38]. It takes a purely geometric approach which is solved using the 5 point algorithm and preemptive RANSAC described in Nister [37].

In this chapter, initial work that was carried out to study the feasibility of purely vision based navigation and localisation is presented and discussed.

2.1 Optical Flow Based Navigation

During the initial stages of the project, vision based navigation was explored as a possible solution to autonomous navigation for an MAV. It is relatively easy to implement and has been demonstrated to operate in indoor environments by several researchers ([46], [17]). An algorithm for optical flow based navigation is described here which was also tested on a commercial quadrotor. This work was carried out to study the performance of a purely vision based approach to navigation in an unstructured indoor environment.

Method
This method works on the principal that when the MAV is translating, closer objects give rise to faster motion across the imager than farther objects. Thus in this scheme, the MAV turns away from the side of greater optical flow. In order to implement this method, the field of view of the forward facing camera is divided into two halves i.e. left and right. The optical flow field in the left half $O\vec{F}_L$ and right half $O\vec{F}_R$ are calculated using the Pyramidal Lucas-Kanade feature tracking algorithm described by [21]. The control law is designed as:

$$\Omega = \frac{\sum \|\vec{OF_L}\| - \sum \|\vec{OF_R}\|}{\sum \|\vec{OF_L}\| + \sum \|\vec{OF_R}\|} \tag{2.1.1}$$

Where Ω, further scaled by an appropriate factor, is fed as a yaw rate command to the vehicle,$\|\vec{OF}\|$ represents the magnitude of optical flow. Fig. 2.1 shows the principle of operation.

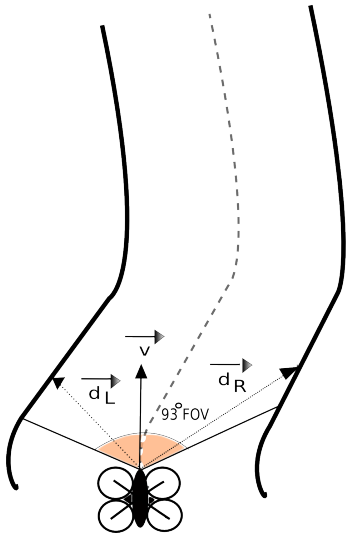


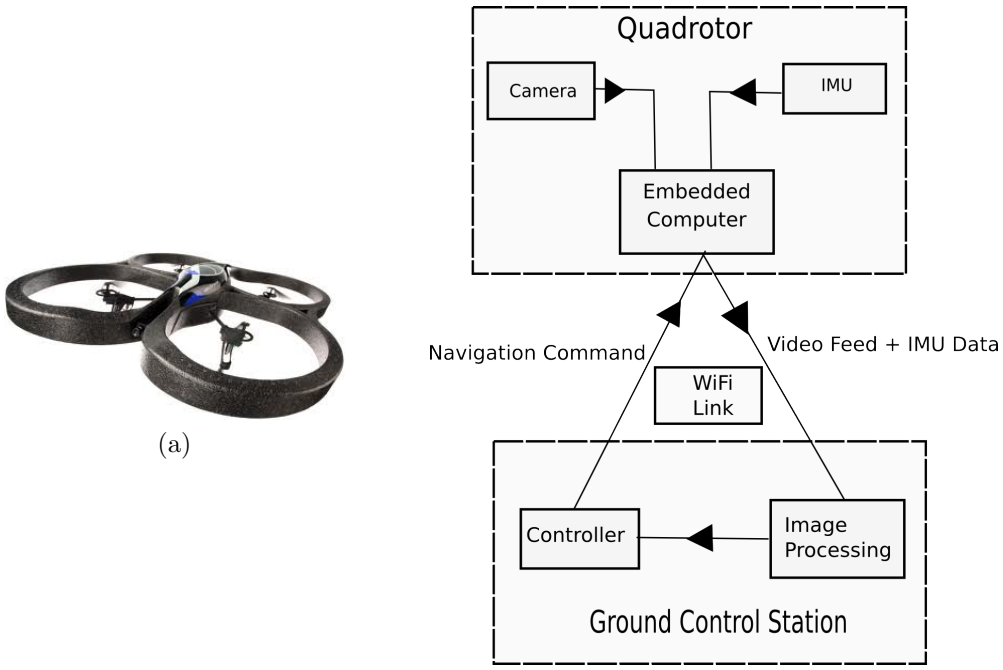
Figure 2.1: Navigating a corridor environment by balancing optical flow. In each visual hemifield up to 200 features are tracked. d_L and d_R indicate the distance vectors from the camera optical centre to obstacles on left and right

Experimental Setup

To test the vision based obstacle avoidance algorithm and verify its suitability for an MAV, a Parror A.R drone (shown in Fig. 2.2) has been used for experiments. It is a lightweight quadrotor equipped with a forward looking 93 degree wide-angle diagonal lens camera, a downward looking camera used for hover stabilisation and an ultrasound altimeter. The on-board IMU is equipped with a 3 axis accelerometer and 3 axis gyro. All image processing algorithms are run off-board on a ground control station which communicates with the MAV via a Wifi link. Fig. 2.2 shows the architecture of the set up. The image processing is done using a Python script written using the OpenCV, open source image processing library. The interface between the image processing script and the Wifi Link to the quadrotor is provided by the official Software Development Kit provided by Parrot for the A.R Drone [4] and an open source library called AutoPylot [3].

Using the experimental setup described in Figure 2.2, a test was conducted to verify the robustness of the optical flow balancing algorithm in an unstructured office environment. Figure 2.3 shows the images from the forward facing camera of the MAV. The blue circles represent the corner features. The left and right green horizontal lines represent the optical flow in the respective visual hemifields. The central green horizontal line depicts the direction and magnitude of the commanded yaw to avoid the obstacle. Image 1 and 2 show the MAV approaching an open door, the central green line in image 2 shows the commanded yaw (i.e. to the left). Image 3 shows the MAV turning away from the obstacle and image 4 shows that the MAV has avoided the obstacle successfully.

It was seen that this optical flow based navigation system is capable of working in scenarios where there is significant motion parallax between the left and right halves of the image. Since, it



(a) (b)

Figure 2.2: Experimental Setup: (a) Parrot A.R Drone (b)Control Flow

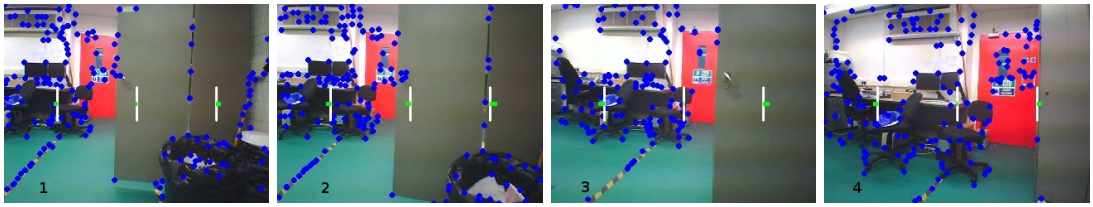


Figure 2.3: MAV flying in cluttered indoor environment. The blue circles represent corner features.

is this motion parallax, which generates the turn rate command. For a corridor like environment, such an algorithm is perfectly suited to guide an aerial vehicle. A corridor provides distinct features in the left and right halves (i.e. corner feature on left and right walls) which generate significant motion parallax. Thus, the vehicle can always be commanded to maintain zero net optical flow, which would guide the vehicle away from the walls. However, in situations where there are large planar obstacles present directly in front of the camera, this algorithm is not able to command a turn since both, left and right visual hemifields exhibit nearly identical optical flow.

2.2 Visual Odometry

An algorithm was implemented based on the 5 point stereo correspondence algorithm developed by Nister et al. [38] that estimates the motion of a camera based on video input. This generates visual odometry, i.e. motion estimates from visual input alone. The 5 point algorithm like any other stereo correspondence algorithm (e.g. 8 Point Algorithm), helps calculate the Fundamental Matrix for feature correspondence. This Fundamental matrix is a 3×3 matrix which relates corresponding points in stereo pairs. If a camera is calibrated, then using its

Intrinsic matrix, the Elementary matrix of the camera can be calculated for the relative poses. The Elementary matrix contains information about the rotation and translation of the camera between two poses (the translation is of-course not scaled). A detailed description of the mathematical solution for this method is beyond the scope of this thesis. The reader is referred to [37] for a thorough understanding.

The key points in this method are:

1. Features are detected in each frame using the Shi-Tomasi corner detector [39]
2. These features are then matched and tracked between pairs of frames using the Pyramidal Lucas-Kanade [21] optical flow tracking algorithm.
3. Using the corresponding features, relative poses between successive frames are calculated using the 5-point algorithm [37]

The code for this method was written using a Python script and OpenCV. An open source implementation of the 5 point method is available from the author's web page [6]. Since I was working with a monocular camera, scale estimation becomes a major drawback. Without the correct scale, any SLAM algorithm cannot present a realistic description of its environments. Therefore, it was decided not to use this algorithm for the SLAM implementation. Also, due to the heavy computational load of this code, it is able to run only at frame rates up to 4 Hz. Thus, it was felt that it would not be suitable for real-time implementation on a dynamic MAV.

2.3 Summary

Two purely vision based systems i.e. optical flow based navigation and stereo correspondence based visual odometry were presented and discussed in the previous sections. Each has its advantages and disadvantages. While the optical flow based navigation algorithm is suitable for a corridor like environment, the visual odometry based system runs at a slow rate due to computational complexity. This motivated the search for a solution, which would enable an MAV to navigate and autonomously map an indoor environment. A careful literature survey revealed that a filter based approach provides the best results for SLAM. Thus, an Extended Kalman Filter based SLAM system, which can maintain an accurate 3D representation of the environmental features as well as the vehicles pose in real time was investigated. Further, it was felt that a commercial quadrotor platform like the A.R Drone did not have the required payload capacity nor the quality of on-board instrumentation (inertial sensors, cameras) needed for this research. Furthermore, commercial aerial platforms that could possibly be used for this project are too expensive and provide limited flexibility in terms of payload. This led to the development of an indigenous quadrotor platform. It was felt that building a quadrotor would provide several advantages, namely; (i) Complete control over the sensory payload, (ii) Lowered development cost, (iii) Software and hardware development expertise gained by various lab members, (iv) Modular platform would be easy to replicate for further research.

This leads us to the proceeding chapters which investigate a monocular vision based EKF-SLAM approach followed by the design and development of the quadrotor aerial vehicle. Further on, the flight test results for the complete system are presented.

CHAPTER 3

Real-Time Monocular Vision based Simultaneous Localisation and Mapping

In this chapter, Real Time Monocular SLAM is described, based on the methods developed by Davison [28]. The advantage of this system is that it offers *'Repeatable Localization'*, wherein there is minimal drift from the ground truth. The main aim of this method is to generate a 3D map of scene landmarks which can be referenced over a long time period whilst the camera is in motion, within a probabilistic framework. This map should allow loop closure, i.e. re-recognition of old features that had gone out view, when they are observed again by the camera. Loop closure helps reduce the drift in the map. Researchers have adapted this methodology for use in aerial vehicles. It has been used successfully by [41] for an airship, to aid navigation in a situation where the vehicle loses GPS signals. Their paper demonstrates a heading only visual SLAM based on Davison's MonoSLAM which aids the Inertial Measurement Unit in keeping a track of the vehicle's attitude and position for short periods.

3.1 Methodology

Monocular vision based SLAM is a localisation and mapping method, that works by extracting, storing and re-observing visual features detected by a single camera in motion. The global positions of these visual features are stored as part of the map's state. Since an EKF is used here, these visual landmarks form part of the states of the EKF along with the states of the camera (position, orientation, linear velocity and angular rates). The primary objective of the map is to permit localisation rather than building a dense reconstruction of the environment. Therefore,only a sparse set of high-quality landmarks is tracked. The scene is assumed to be rigid and each landmark is assumed to be a stationary landmark. The visual landmarks are assumed to map to a well defined feature in 3D space. The following sections will explain in detail, how the visual SLAM method works, along with the key equations used in the implementation.

3.1.1 The Filter State

The SLAM map is comprised of the EKF state vector $\hat{x}$ which is basically the vertical stack of state estimates of the camera (13 parameters) and features (6 parameters per feature).

$$\hat{x} = \begin{pmatrix} \hat{x}_v \\ \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{pmatrix} \tag{3.1.1}$$

The camera's state vector x_v comprises:

- 3D position: r^W
- Orientation quaternion: q^{WR}
- Linear velocity: v^W
- Angular velocity: ω^W

relative to 'W' the fixed world frame and "robot" frame 'R' carried by the camera. The camera state is represented mathematically as:

$$x_v = \begin{pmatrix} r^W \\ q^{WR} \\ v^W \\ \omega^W \end{pmatrix} \tag{3.1.2}$$

In Eqn. 3.1.1, y_i is the state of feature 'i', where $i \in [1, n]$, which represents the 3D position vector $P_i = [X_i, Y_i, Z_i]^T$ of the location of the point feature in the world frame using inverse depth parametrisation [24].

A feature is represented in the map by using the inverse depth representation described in [24], as follows:

$$y_i = \begin{pmatrix} x_{ci} \\ y_{ci} \\ z_{ci} \\ \theta_i \\ \phi_i \\ \rho_i \end{pmatrix} \tag{3.1.3}$$

The position vector P_i of the feature in 3D space, is expressed mathematically in terms of the inverse depth parametrisation as:

$$\begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} = \begin{pmatrix} x_{ci} \\ y_{ci} \\ z_{ci} \end{pmatrix} + \frac{1}{\rho_i} m(\theta_i, \phi_i) \tag{3.1.4}$$

Where,

$$m = [cos(\phi_i)sin(\theta_i), -sin(\phi_i), cos(\phi_i)cos(\theta_i)]^T \tag{3.1.5}$$

$[x_{ci}, y_{ci}, z_{ci}]^T$ is the position of the camera centre when the feature is first observed, ρ_i is the inverse depth of the feature along the ray joining the camera centre to the feature and $m(\theta_i, \phi_i)$ represents the unit vector in that direction as a function of the azimuth and elevation.

3.1.2 Feature Initialisation

As explained in the previous sub-section, the features are represented in the map using inverse depth parametrisation. The inverse depth parametrisation only contains information about the feature's position whereas the SLAM system must be able to uniquely identify each feature to be able to recognise them successively from the stream of camera images. Since, this a vision based system, a feature's visual appearance is the sole information that the camera can provide. Therefore, instead of tracking corner features, relatively large image patches (for e.g. 21×21 pixels) are tracked, as they are distinct which makes them stable landmarks. The process for feature initialisation starts by, first, detecting a corner feature in the camera image via the Shi-Tomasi [39] detector after which a 21×21 pixel image patch, centred at the corner feature is extracted. The extracted patch image is stored as a feature's descriptor in the SLAM system's memory. This patch image is stored only once when a feature is initialised after which it is not updated. An approximation is made that a feature is locally planar. If this approximation does not hold true, then as the feature is re-observed, it is not matched correctly during the update step of the Kalman filter. In such a case, the feature is deleted based on a threshold criterion of its cross-correlation score while matching with the original stored image. This threshold is determined heuristically.

Once the feature point is extracted from the image, it is mapped from the 2D image plane coordinates $[u, v]$ to normalised 3D world coordinates $[U, V, 1]$ as:

$$\begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} (u - u_0)/fk_u \\ (v - v_0)/fk_v \end{pmatrix} \tag{3.1.6}$$

where $fk_u, fk_v, u_0, and v_0$ are the standard camera calibration parameters. To orient the vector $[U, V, 1]$ along the world frame, it is rotated by the rotation matrix R_{WR}. Thus, the direction vector from the camera centre to the landmark in world frame is given by:

$$\begin{pmatrix} h^W_{Lx} \\ h^W_{Ly} \\ h^W_{Lz} \end{pmatrix} = R_{WR} \begin{bmatrix} U \\ V \\ 1 \end{bmatrix} \tag{3.1.7}$$

After the feature is extracted, it is inserted into the map i.e. the filter state and covariance need to be modified. The map's state vector simply expanded to accommodate the 6 parameters of a new feature. The feature's covariance is initialised using the image measurement error covariance R_i and state estimate covariance $\hat{P}_{k|k}$. The initial value of the inverse depth ρ_0 and its standard deviation σ_ρ is set based on trial and error. For the experiments carried out, $\rho_0 = 0.25$ and $\sigma_\rho = 0.5$. This gives an inverse depth confidence region of $[-0.75, 1.25]$. This confidence region covers a depth of infinity as well. The filter state covariance (from [24]) is:

$$\hat{P}^{new}_{k|k} = J \begin{pmatrix} \hat{P}_{k|k} & 0 & 0 \\ 0 & R_i & 0 \\ 0 & 0 & \sigma^2_\rho \end{pmatrix} J^T \tag{3.1.8}$$

Where,

$$J = \left(\begin{array}{c|c} I & 0 \\ \hline \frac{\partial y}{\partial r^W}, \frac{\partial y}{\partial q^W}, 0, ..., 0 & \frac{\partial y}{\partial h}, \frac{\partial y}{\partial \rho} \end{array} \right) \tag{3.1.9}$$

3.1.3 Motion Modelling

The state vector is updated in two ways i.e.; 1) Prediction: i.e. when the camera moves in the interval between image capture and 2) Update: i.e. after features are measured. For a camera undergoing motion, the motion model has to account for the random dynamics of the camera. These can be probabilistically modelled. For a camera without odometry, a "constant velocity, constant angular velocity model" is used similar to that used in [28]. It does not restrict the camera motion, but implies that undetermined accelerations occur with a Gaussian profile. Necessarily, it expects smooth dynamics for the camera: big accelerations are assumed to be unlikely.

For each time step, unknown acceleration a^W and angular acceleration α^W are assumed to be processes of zero mean Gaussian white noise, which apply a velocity and angular velocity impulse:

$$n = \begin{pmatrix} V^W \\ \Omega^W \end{pmatrix} = \begin{pmatrix} a^W \Delta t \\ \alpha^W \Delta t \end{pmatrix} \tag{3.1.10}$$

The covariance matrix of the noise vector $\mathbf{n}$ is assumed to be diagonal. Q_v, the process noise covariance is calculated as:

$$Q_v = \frac{\partial f_v}{\partial n} P_n \frac{\partial f_v^T}{\partial n} \tag{3.1.11}$$

where P_n is the covariance of noise vector n. The state update produced is:

$$f_v = \begin{pmatrix} r^W + v^W \Delta t \\ q^{WR} \times q((\omega^W + \Omega^W)\Delta t) \\ v^W + V^W \\ \omega^W + \Omega^W \end{pmatrix} \tag{3.1.12}$$

The rate of growth of uncertainty in the motion model is determined by the size of P_n and the values of these parameters dictate the smoothness of the motion expected. Setting a high standard deviation to the expected angular and linear impulses increases the uncertainty at each update, which must be handled by a large number of good feature matches. On the other hand, a small standard deviation for the impulses would imply smooth motion for the camera, in which case the filter can become unstable if the camera moves suddenly. For this work, the standard deviation for linear impulse was set to $a = 1\ m/s^2$, for angular acceleration it was set to $\alpha = 1\ rad/s^2$. If, there is an Inertial Measurement Unit present, then these parameters can be simply set to the noise characteristics of the sensors as specified by the manufacturer.

3.1.4 Map Initialisation

When the robot is switched on, it doesn't have knowledge about the the world around it. A coordinate frame is defined by the robot wherein it will determine its own motion and build a map, therefore the most logical choice is to place the origin at the robot's starting position. At start-up, a minute amount of information can be given in the form of a known target in front of the camera if the camera does not have access to odometry. This is able to provide several features with known positions and of known appearance. There are two main reasons for this:

- Since there is no direct way to measure feature depths, a known target allows the system to assign a precise scale to the estimated map

- Having features right from the beginning implies that the robot can immediately enter the normal 'predict-measure-update tracking sequence' without any special first step

If there is an IMU present, the map gets scaled automatically when measurements are read from the accelerometer and fused in the EKF.

3.1.5 Feature Measurement and Updating the Map

The location of a landmark relative to the camera is calculated using the estimates, r^W of camera location and P_i of feature location:

$$h_L^R = R^{RW}(P_i^W - r^W) \tag{3.1.13}$$

Where,

h_L^R: Vector from the the camera centre to the feature in the camera's body frame
R^{RW} : Euler rotation matrix of the camera

The standard pin hole model allows the calculation of the position (u, v) at which the feature would be expected to be found in the image:

$$h_i = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u_0 + fk_u \frac{h_{Lx}^R}{h_{Lz}^R} \\ \\ v_0 + fk_v \frac{h_{Ly}^R}{h_{Lz}^R} \end{pmatrix} \tag{3.1.14}$$

Feature matching is performed on raw images rather than undistorting them. The projected coordinates $\mathbf{u} = (u, v)$ are radially distorted to obtain the final predicted image position $\mathbf{u}_d = (u_d, v_d)$. A two parameter radial distortion model from [34] is used. First r_u and r_d are calculated as follows:

$$r_u = r_d(1 + K_1 r_d^2 + K_2 r_d^4) \tag{3.1.15}$$

r_u can be directly calculated as,

$$r_u = \sqrt{(u - u_0)^2 + (v - v_0)^2} \tag{3.1.16}$$

However, r_d is iteratively calculated using the Newton-Raphson Method. (Steps 1 through 3 are repeated 10 times) as:

1. Calculate f:
$$f = r_d + K_1 r_d^3 + K_2 r_d^5 - r_u \tag{3.1.17}$$
2. Calculate f_p :
$$f_p = 1 + 3K_1 r_d^2 + 5K_2 r_d^4 \tag{3.1.18}$$
3. Calculate r_d as:
$$r_d = r_d - \frac{f}{f_p} \tag{3.1.19}$$
4. Repeat from step 1 ten times

The distorted coordinates are then:

$$u_d - u_0 = \frac{u - u_0}{(1 + K_1 r_d^2 + K_2 r_d^4)} \tag{3.1.20}$$

$$v_d - v_0 = \frac{v - v_0}{(1 + K_1 r_d^2 + K_2 r_d^4)} \tag{3.1.21}$$

Where K_1 and K_2 are the radial distortion parameters obtained from camera calibration. The Jacobians of this two-step projection function with respect to camera and feature positions give the uncertainty in the prediction of the feature position, represented by the symmetric 2×2 innovation covariance matrix S_i :

$$S_i = \frac{\partial u_{di}}{\partial x_v} P_{xx} \frac{\partial u_{di}}{\partial x_v}^T + \frac{\partial u_{di}}{\partial x_v} P_{xy_i} \frac{\partial u_{di}}{\partial y_i}^T + \frac{\partial u_{di}}{\partial y_i} P_{y_i x} \frac{\partial u_{di}}{\partial x_v}^T + \frac{\partial u_{di}}{\partial y_i} P_{y_i y_i} \frac{\partial u_{di}}{\partial y_i}^T \tag{3.1.22}$$

The measurement noise covariance R_i is a constant, with the diagonal elements equal to the image noise in pixels. The standard deviation for image noise is assumed to be 0.8 pixels. S_i is of the shape of an ellipse representing a 2D Gaussian *Probability Distribution Function* over image coordinates and choosing the standard deviations (usually 3σ) defines an elliptical search region where there is maximum probability of finding the feature.

S_i provides a measure of how valuable a measurement is; for e.g. feature searches with high S_i (where the result is difficult to predict) will provide more information about estimates of camera and feature positions.

At each measurement step of the Kalman Filter, the measurement hypothesis are tested by projecting them as an elliptical search region onto the image. After the 3D point is projected into the image using the standard measurement model, the innovation covariance provides the elliptical search area where the feature is most likely to be found. Within the elliptical search area, the SLAM filter tries to match the initial patch image stored in the map to the one extracted from the current image. Normalized cross-correlation search is used to match the template of the stored patch to the possible patches within the search region. If the correlation score exceeds a threshold of 0.8, the feature is said to have been measured successfully and its confidence level is increased by 1. In case the correlation score falls below 0.6, the measurement is considered bad and the confidence level of the feature is decreased by 1. A low correlation score can be attributed to an occlusion or the feature not being distinct enough, which makes it unsuitable for the SLAM map. If the correlation score falls between these two thresholds then the feature's confidence level is unaffected. The confidence level of a feature is basically a numerical value which is affected by the measurement of that feature. The confidence value helps in deciding which features are to be kept in the map and which are to be deleted. If a feature's confidence value falls below zero, it is deleted from the SLAM filter. If the number of matched features falls below a threshold (usually 4-6 features) then new features are searched for in the image, in regions where the camera is not predicted to observe any past features.

3.2 Algorithm Implementation

A real time C++ code has been developed implementing Monocular SLAM using an Extended Kalman Filter based on the methodology described in the previous section. The main steps of the algorithm are as follows:

1. Initialise the camera state vector and covariance based on an initial estimate of the position and uncertainity. In our case the starting position is assumed to be $(X = 0, Y = 0, Z = 0)$ and the camera coordinate frame is assumed to be oriented along the world frame
2. Acquire image from camera and convert it to grayscale for fast processing
3. Extract feature patches (of size 21×21) using Kanade-Lucas-Tomasi (KLT) detector. The number of features tracked is key in deciding the accuracy of the system. More features

result in higher accuracy but slower performance. A compromise must be reached between performance and accuracy

4. Predict new feature position based on current motion estimates of camera
5. Active search of features within search region defined probability distribution
6. Match features by patch cross correlation (min. correlation score 0.8 for successful match). A track is kept of the number of times a features is matched which affects the confidence level of each feature as described in section 3.1.5
7. Update Kalman filter using the matching observations
8. Render 2D and 3D graphics for visualisation (optional)
9. If the number of matched features falls below a certain predefined threshold (usually 6 features), new features are added in regions of the image which previously do not contain features.
10. Start over from step 2

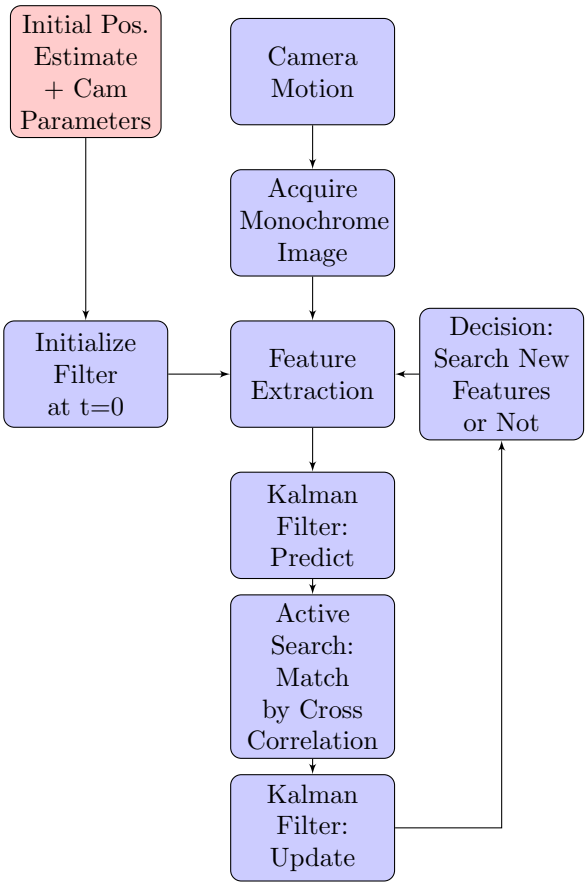


Figure 3.1: Flowchart depicting the main steps of the algorithm

3.3 Simulation Performance

The table below gives a break down of the processing time for the code with a minimum of 12 features in view. It shows that the current implementation can easily run at 25 Hz which is sufficient for real time SLAM. An increase in the performance can be attained via code optimisation and skipping the graphical rendering step.

Table 3.1: Average Processing Time for Different Stages of the Algorithm: Real Time

Step	Time (ms)
Image Acquisition	2
Feature Correlation Search	2
Kalman Filter	29
Feature Initialisation Search	3.5
Graphical Rendering	1.75
Total	38.25

3.3.1 Accuracy

The accuracy of the system was tested against the results generated by the MATLAB code provided by Civera et al. [25]. Their implementation is currently the state of the art in EKF based monoslam. The position and orientation were compared for a sequence of images for camera motion of 130 seconds. Table 4.2 presents the position estimates from the two at every 10 seconds.

Table 3.2: Localisation accuracy

Ground Truth			SLAM (m)		
x	y	z	x	y	z
-0.1268	0.0416	0.0372	-0.1336	0.0509441	0.0531847
-0.2821	0.0644	0.0277	-0.278938	0.0759559	0.0536762
-0.3839	0.0651	0.0181	-0.418046	0.075778	0.0415132
-0.3628	0.0437	0.0267	-0.395185	0.0480599	0.0825013
-0.2388	0.0295	0.0445	-0.245275	0.0310467	0.0626606
-0.0941	0.0100	0.0341	-0.103298	0.00984878	0.0561518
-0.0179	-0.0250	0.0102	-0.0204742	-0.0260352	0.0387348
-0.0330	-0.0564	0.0047	-0.0327291	-0.0743485	0.067596
-0.1090	-0.0473	0.0157	-0.142783	-0.0713295	0.0745398
-0.2212	-0.0187	0.0146	-0.310985	-0.0292691	0.0839611
-0.3279	-0.0043	-0.0292	-0.39637	-0.0111663	0.0803065
-0.3482	-0.0176	-0.0705	-0.378851	-0.0231401	0.0244458
-0.3000	-0.0262	-0.0742	-0.312162	-0.0206843	-0.0192843

Table 4.3 presents the position mean and standard deviation of the error between the values from [25] and my code. It can be seen that the positioning accuracy is very reliable and the order of error in centimetres. It is clear that once IMU data is fused with the Kalman filter, the positioning accuracy will become even more accurate and we can hope to limit the error

for long trajectories.

Table 3.3: Mean and Standard Deviation of Error

	x(m)	y(m)	z(m)
Mean	0.0249	0.0018	-0.0493
Standard Deviation	0.0279	0.0109	0.0302

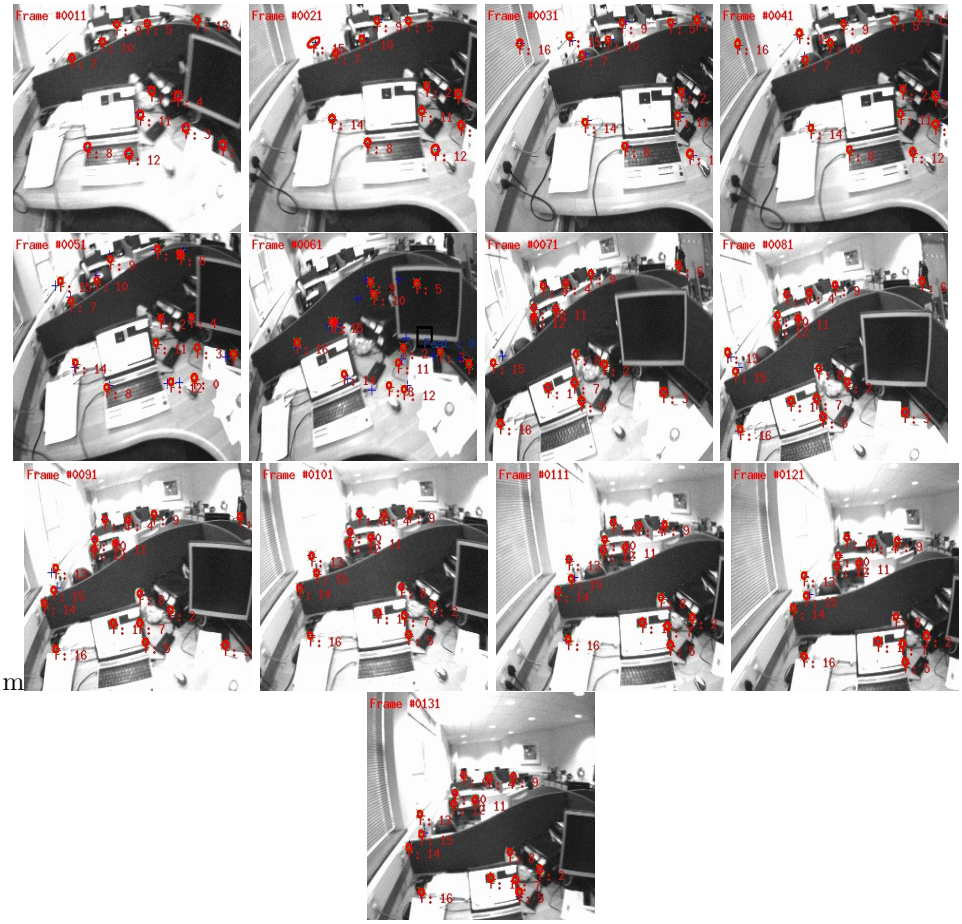


Figure 3.2: Sequence of images from simulation

Fig 4.2 shows the key frames at which the localisation values were compare and Fig 4.3 depicts the 3 dimensional path of the camera.

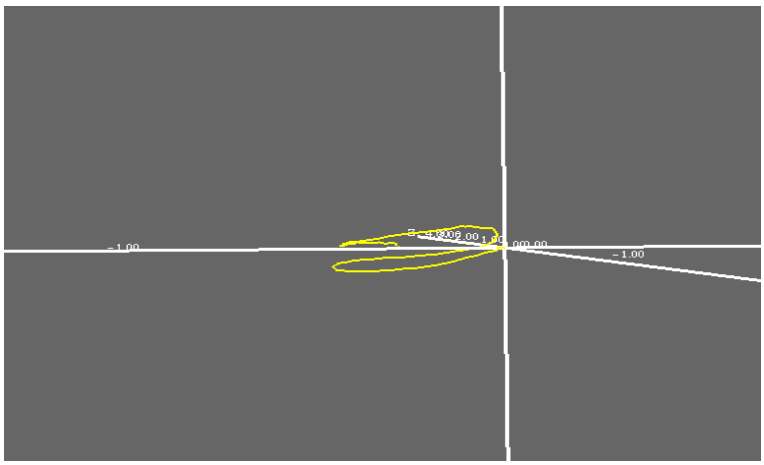


Figure 3.3: 3D path reconstruction

3.4 Improving IMU performance with Visual SLAM

Here are presented the results of the work published in a paper based on the work done as part of this project. A software framework was developed to test the proposed SLAM system. The virtual world is populated with 3D landmarks. The Kalman filter is initialised with a high uncertainty. Figure 3.4, 3.5 and 3.6 show the true position and the estimates from the IMU and the vision aided filter. Figure 3.7, 3.8 and 3.9 show the orientation estimates error from the IMU and the vision aided filter. It is clear that visual measurements enhance the accuracy of the odometry. As time progresses the Kalman filter is able to accurately determine the state of the vehicle. The IMU readings exhibit continuous divergence from the true value whereas the filtered estimates have considerably less error. Compared to the position error the heading error is slightly higher but still within reasonable bounds.

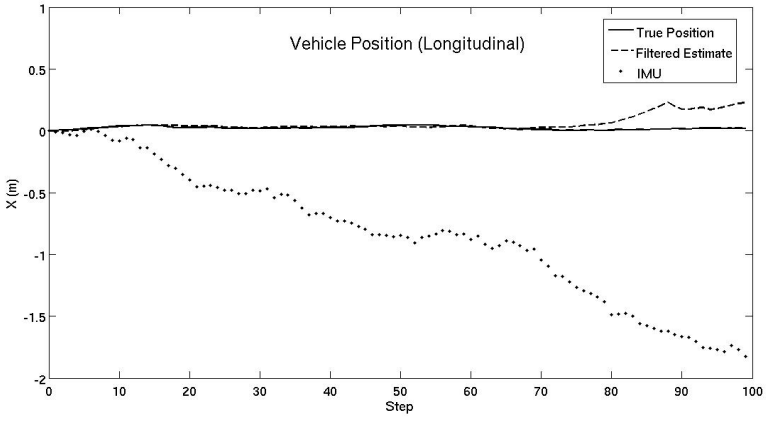


Figure 3.4: Vehicle position along X axis

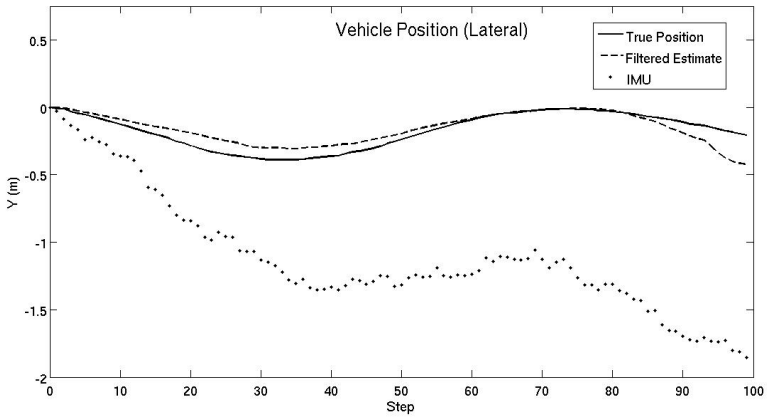


Figure 3.5: Vehicle position along Y axis

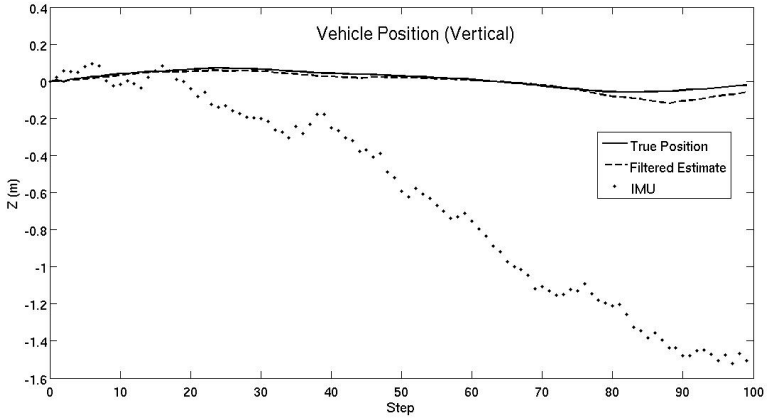


Figure 3.6: Vehicle position along Z axis

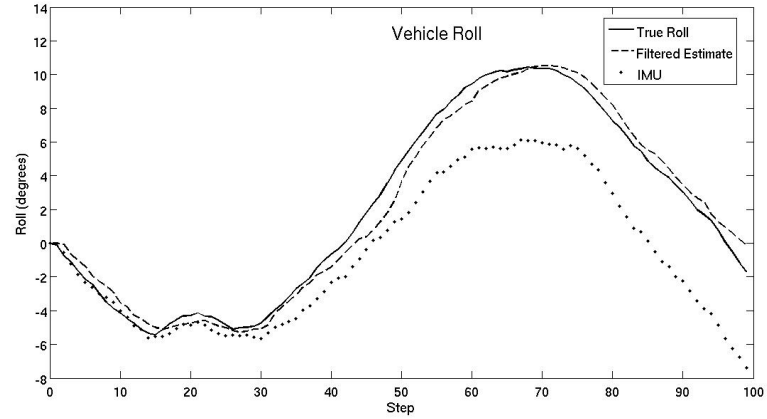


Figure 3.7: Vehicle Orientation in Roll (ϕ)

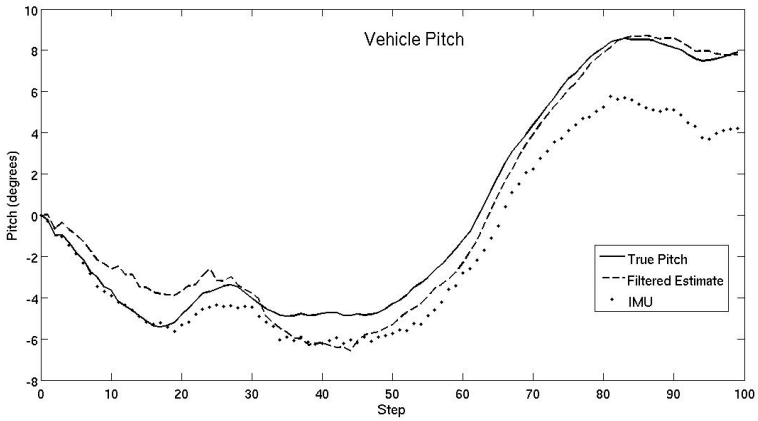


Figure 3.8: Vehicle Orientation in Pitch (θ)

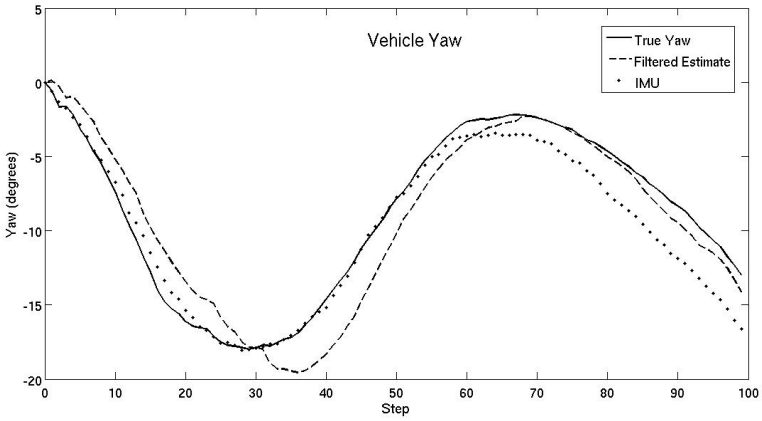


Figure 3.9: Vehicle Orientation in Yaw (ψ)

3.5 Real-Time SLAM

Finally, the performance of the algorithm was tested in real-time using a standard webcam. The camera was moved along a rectangle of side 16 cm $\times$ 10 cm on a flat surface in a 67 second run (the principal axis of the camera is not perfectly aligned with the physical rectangle, which accounts from the alignment error of the rectangle in the 3D path reconstruction from the X-Y plane i.e. the surface). The error between the starting position and end position in the localisation is 2.63 cm. Fig. 3.10 shows the path reconstructed as the camera was moved along a rectangle.

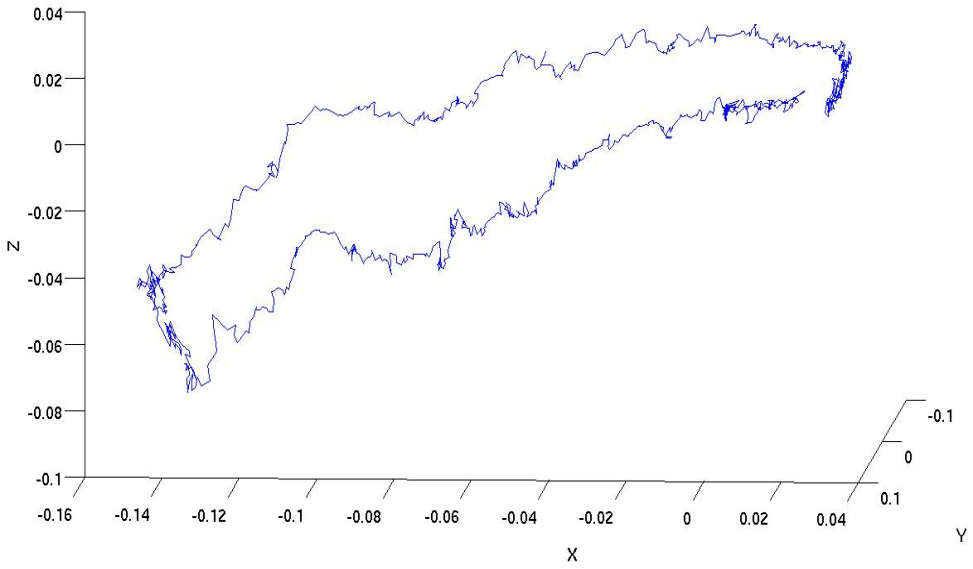


Figure 3.10: 3D Path

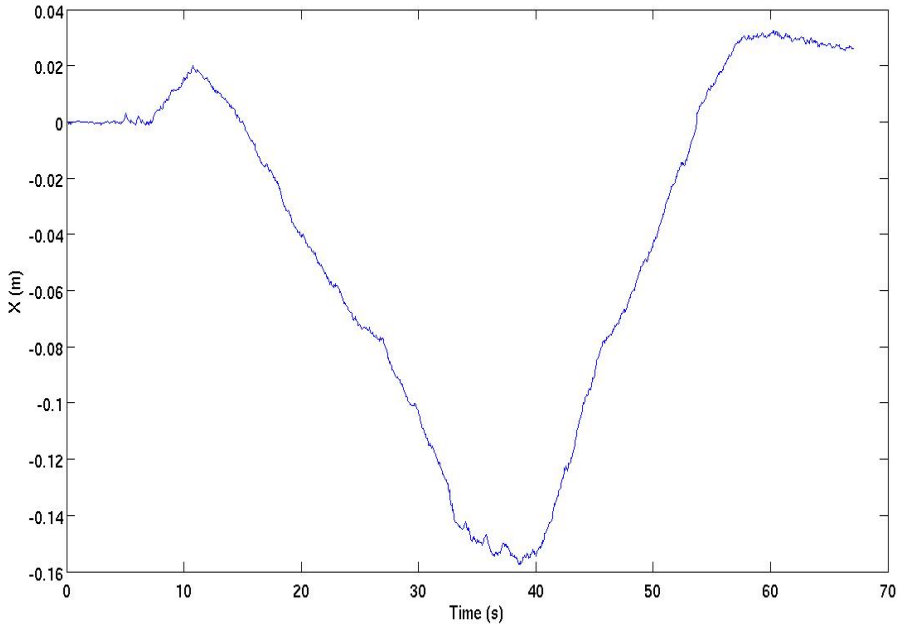


Figure 3.11: X vs. Time

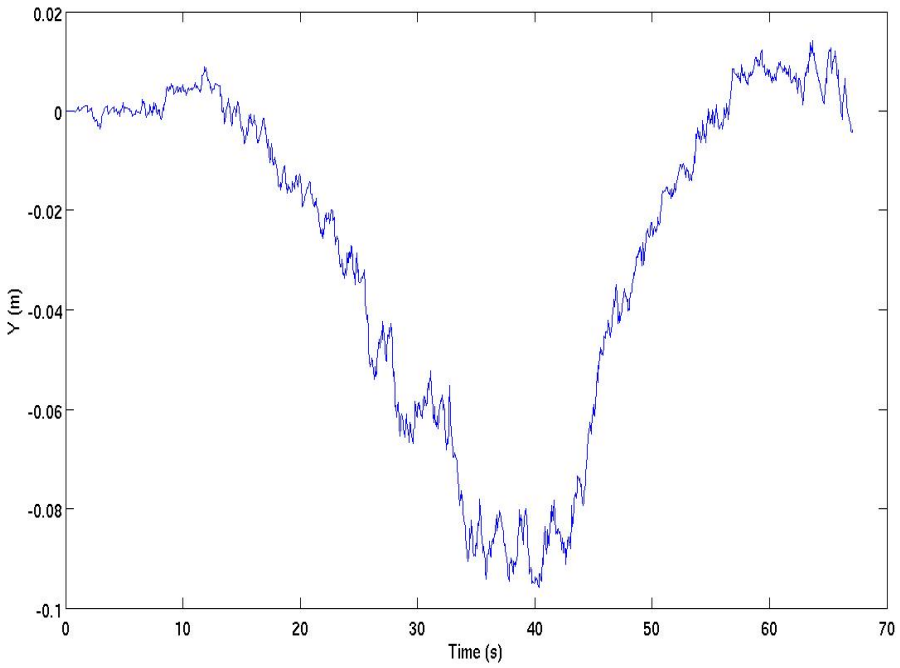


Figure 3.12: Y vs. Time

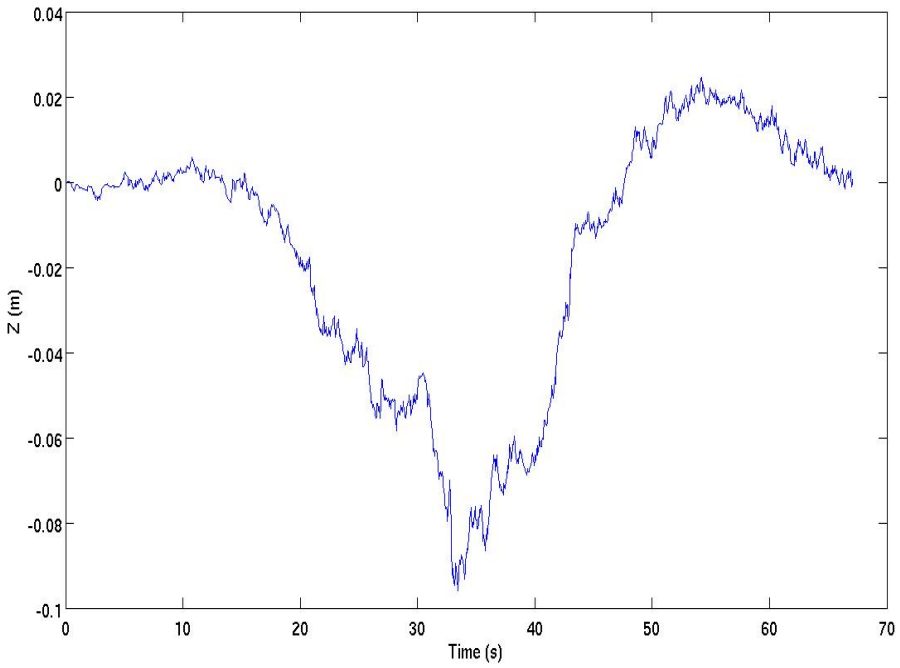


Figure 3.13: Z vs. Time

3.6 Summary

The key points of this approach are that it is able to model the motion probabilistically, while enabling an active measurement of the landmarks. It is highly efficient and robust to randomness of the camera dynamics and at the same time is capable of loop closure. Compared to other SLAM algorithms that use Rao-Blackwellised Particle Filter based approaches like [36] and [44], Davison's method uses an Extended Kalman Filter. While an EKF based approach cannot be used for a big map, it is certainly better at tracking since its means the non-diagonal covariance relationships are maintained in the SLAM filter. These relationships are highly useful as each features observation affects the rest of the features. Its limitation is that as it relies on an EKF to maintain the state estimates, high number of features tend to increase the complexity of the calculations which makes this system only suitable for small environments. However, it is possible to extend the algorithm for large scale environments by introducing some form of intelligent map management. It has been shown in other work on submaps that a system of small scale maps can be successfully joined by a set of estimated transformations as long as the submaps can be matched accurately. Clemente et al. [26] have published a paper on mapping large outdoor loops using a single camera SLAM system running an EKF by building and storing small individual local maps in real time and then associating them in the background. This chapter has described the algorithm and its performance characteristics. It is proven that that the speed of the algorithm is suitable for accurate real time SLAM. The accuracy is then verified by the simulations and real-time test. The algorithm successfully implemented loop closure which is crucial to enhance localisation estimates as landmarks are re-observed. This key feature enables long term accurate tracking. Using inverse depth parametrisation for feature initialisation, feature depth uncertainty is successfully reduced and estimated within seconds.

CHAPTER 4

Design and Development of Quadrotor

4.1 Introduction

A thorough survey of commercially available MAVs for the purpose of this research revealed that the only suitable system was the Pelican quadrotor as shown in fig. 1.4. However, the cost of the entire system exceeds 15,000 GBP which was beyond our budget. Hence, it was decided to build a quadrotor in-house using COTS components from various suppliers. This way, the cost of the entire system has been brought down to less than half of the commercially available system. The second major advantage of building such a complex system in-house is that it helps build the capability for systems integration within the group and future projects will benefit from the plug and play capabilities developed in this project.

4.2 Baseline Specifications

For the purpose of building this quadrotor, certain basic requirements were specified, namely:

1. Flight time of approximately 10 minutes
2. Ability to carry approximately 500 grams of payload (LIDAR , Single Board Computer, camera)
3. Easily repairable
4. Modular construction which can be easily replicated

These specifications were loosely based on the performance of the Pelican quadrotor as advertised by Ascending Technologies [5] and tailored to meet our expectations from the MAV. An initial estimate of the expected mass breakup is shown in Table 4.2.

Table 4.1: Expected Weights

Component	Mass (g)
Frame	350
Motors	220
Main Battery	500
Secondary Battery	200
IMU	50
Arduino	40
Xbee Module	30
Wiring	100
Single Board Computer	300
R/C Receiver	20
LIDAR	165
Electronic Speed Controllers	120
Miscellaneous Structural Components	200
Sonar	5
Camera	25
Total	2325

4.3 Hardware Description

To meet the baseline specifications, a survey of available products revealed the most suitable and easily available components as described below :

4.3.1 Frame

A standard quadrotor frame and landing gear designed for heavy payloads was purchased from Hi-Systems GmbH. Carbon fibre and polyvinyl carbonate sheets were used to mount the IMU and stability controller, laser scanner and on-board computer. To mount all the sheets onto the centre plate, double angle strips available from Meccano were used.

The quadrotor includes the following major structural components (excluding nuts/bolts):

1. 4 Aluminium arms with dimensions 29 cm × 1 cm × 1 cm and a thickness of 1 mm weighing 25 gm each.
2. 2 Polycarbonate Discs with a diameter of 18 cm and thickness of 1 mm for use as the centre plates weighing 10 gm each
3. Plastic landing gear weighing 85 gm.
4. 1 carbon fibre sheet with dimensions 15 cm × 18 cm × 0.2 cm weighing 35 gm
5. 1 carbon fibre sheet strengthened with a polyvinyl carbonate sheet with dimensions 13 cm × 7.5 cm × 0.2 cm weighing 40 gm
6. 4 double angle strips 5 × 2 holes weighing 8 gm each and 6 double angle strips 7 × 1 hole weighing 10 gm each from Meccano
7. 1 carbon fibre sheet with dimensions 11 cm × 9 cm × 0.1 cm weighing 10 gm

(a) (b)

Figure 4.1: Components purchased from Hi-Systems GmbH (a) Quadro-XL kit (b) MK FlexLanderXL

4.3.2 Propulsion

Propeller

An initial design revealed that the weight of the system would be close to 2.35 kg. This meant that each motor must provide a minimum of 5.77 N of thrust to hover. However, a margin of safety must be available to accommodate the changes in the design weight. We assumed a 30 % thrust margin. therefore approximately 30 N of maximum available thrust was needed. After studying various manufacturer websites and online resources such as Aeroquad [1] and DIY Drones [7], it was realised that APC Slow Fly propellers are most commonly used for slow flying heavy lift duties such as the one required on this system.

The equation below states the relationship between propellers sizes and their maximum RPMs as suggested by the manufacturer [2]. The Slow fly APC props should ideally not be used close to their maximum RPM as their efficiency drops drastically due to blade tip stall and it is also not safe to operate them beyond the manufacturer specified limit.

$$MaxRPM = \frac{65000}{PropDia(inches)} \tag{4.3.1}$$

An analysis of various propeller sizes revealed that a 12 x 3.8 APC slow fly prop would be ideally suited to this application as it could generate the required thrust at low power consumption.

Table 4.2: Propeller performance

Propeller	Max RPM	RPM @ required max. thrust	Power @ required max. thrust (W)
10 × 3.8	6500	6330	71.4
11 × 3.8	5909	5720	69.9
12 × 3.8	5416	4762	61.2

Motor and Speed Controllers

DC brushless motors are ideal for model aircraft applications as they are lightweight, durable and highly efficient. Dualsky XM2830CA-10 brushless motors were chosen. It is a lightweight (55 gm) motor, capable of running large propellers and a maximum power output of 200 W. The motors were paired with Dualsky XC1812BA Electronic Speed Controllers (ESC) which read PWM signals from the stability computer and in-turn controls the motor RPM.

(a)

(b)

(c)

Figure 4.2: (a) APC 12 × 3.8 Propeller (b) Dualsky XM2830-CA 10 Brushless DC Motor (c) XC1812BA ESC

4.3.3 Power

Lithium Polymer batteries are most commonly used to power small scale models. A Kong-Power 3 Cell Li-Po battery was chosen with a rating of 5100 mAh as it was the best available model in the market that met the power requirements and weight considerations (454 gm).

Figure 4.3: Li-Po battery packs

4.3.4 Sensors

1. **Laser**: We use a short range Hokuyo URG-04LX as it is most suitable for our application. With a range of 4 m and weight of 165 gm, it is one of the lightest laser scanner available in the market and commonly used for aerial and ground robots. Since we do not aim to localise using scan matching, the range of this scanner is sufficient for our purpose.

2. **Camera**: A Unibrain Fire-i Monochrome Digital Board Camera was chosen for the platform. It is one of the most popular cameras used SLAM research as it can be outfitted with various kinds of lenses (e.g. wide-angle) and it provides a high data rate using FireWire at 400Mbps. It progressively scans each frame which leads to reduced motion blur during fast dynamic motion compared to normal CMOS sensor based cameras. This reduced motion blur helps in matching feature appearance during dynamic motion of the

Figure 4.4: Hokuyo URG-04LX 2D Laser Scanner [8]

camera. Its dimensions are a mere 59 × 53.5 × 19.5 mm and it weighs 35 grams. It is used in conjunction with a wide-angle lens with focal length 2.1 mm and horizontal Field of View of 81 degrees and vertical field of view of 60 degrees.

Figure 4.5: Unibrain Fire-i board camera [12]

3. **IMU**: Nowadays there are a number of MEMS based Inertial Measurement Units readily available in the market. For aerospace applications however, there are only a few companies whose products match the performance required for building a SLAM application. Since, the MAV will be flying indoors only GPS integration was not one of our criterion. Based on price comparison, and overall performance the SBG Systems IG-500A IMU was chosen. It runs an on-board Kalman Filter with a max. update rate of 180 Hz and fuses magnetometer, accelerometer and gyroscope readings to calculate the vehicle orientation. It is able to run in polling and continuous data transmission mode and comes with a software development kit that can be used to develop low-level communication protocols.

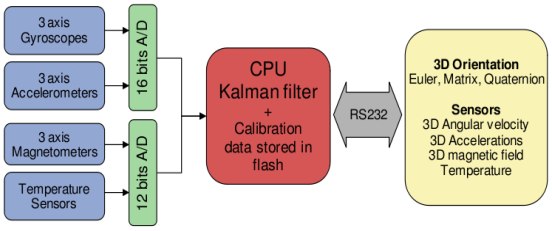


Figure 4.6: Block Diagram of IG-500A functioning [11]

4. **Sonar**: A MaxSonar LV-EZ0 ultrasonic range finder was selected as the altitude sensor. It has a maximum range up to 20 feet and works on a a negligible amount of power (0.05 W). It can output data as analogue voltages, pulse width signals and serial data.

4.3.5 Radio Controller

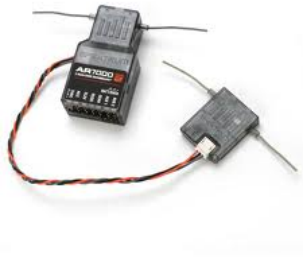
A Spektrum AR7000 2.4 Ghz 7 channel digital receiver was chosen to be used with a MacGregor/JR DSX-12 2.4 Ghz digital transmitter. The receiver provides all the requisite channels

Figure 4.7: SBG Systems IG-500A MEMS IMU [11]

Figure 4.8: MaxSonar LV-EZ0 ultrasonic range finder

for Throttle, Roll, Pitch, Yaw control and 3 mode settings viz. Altitude Hold on/off, Autopilot on/off, Take-off / Landing.

(a)



(b)

Figure 4.9: (a) 12 Channel Transmitter (b) 7 Channel Receiver

4.3.6 Wireless Communication

Wireless communication forms an integral part of the quadrotor. It is used to transmit mission critical information (vehicle states etc.) to the Ground Control Station and receive commands from it. The most common and easy to use devices are Xbee modules which run on the ZigBee communication protocol. They are easy to set up and work as plug and play devices. 2 Xbee pro 2.4 Ghz transmitter/receiver devices along with an Arduino shield and PC connector dongle were acquired. One device is integrated with the Arduino board using the Xbee shield specifically designed for the Arduino Mega. The second module is connected to the PC via the USB Xbee explorer. The maximum available data rate is 57600 bps which is sufficient for wireless telemetry. Fig. 4.10 shows the various components of the wireless communication system.

4.3.7 Stablity Controller

A stability controller is the most crucial system in keeping the vehicle in the air. It must be able to run the real-time PID control loops to maintain the commanded yaw, pitch and roll.

(a) (b) (c)

Figure 4.10: (a) Xbee Arduino Shield (b) Xbee Pro TX/RX module (c) Xbee explorer

An Arduino Mega 2560 based on an ATMEL ATMEGA 2560 microcontroller was selected for this purpose. It is an open source platform which has numerous in-built functionalities such as PWM generation, Serial Communication etc. which can be taken advantage of for rapid development of the control system. It is sufficiently fast to run the stability loop at 150 Hz.

Figure 4.11: Arduino Mega 2560

4.3.8 Computing

The high level on-board computer runs the crucial SLAM algorithms and data fusion to calculate accurate state estimates. Since image processing is a highly computationally intensive job, a powerful, compact computer was needed for this purpose. After testing the SLAM algorithms in simulation on a PC, it was realised that to maintain real-time performance at 25 Hz, a minimum configuration equivalent to the PC would be required. After an exhaustive search, a Single Board Computer NANOGM45-A2 by Impulse Corporation was finalised. It carries an Intel Core2Duo 2.45 GHz processor, with 2 GB of RAM. With some modifications to the operating system i.e. suspending background tasks, adequate real-time performance can be achieved. The drawback of this device is it's high power consumption with a max. rating of 3.06 A at 12 V (36 W). The entire module weighs approximately 0.250 kg without a cooling fan and 0.3 kg with cooling.

Figure 4.12: Single Board Computer

4.4 Weights Analysis of Purchased Components

Table 4.4 show the measure weights of the purchased components. It can be see that the final weight of the vehicle is very close the expected weight distribution given in Table 4.2.

Table 4.3: Weights

Component	Mass (g)
Frame*	318
Motors	248
Main Battery	428
Secondary Battery	192
IMU Assembly**	88
Arduino	37
Xbee Module	32
Wiring	102
Single Board Computer	264
R/C Receiver	17
LIDAR	165
Electronic Speed Controllers	108
SBC base plate	40
LIDAR base plate	50
Brackets	80
Sonar	3
Propeller	17 (per prop)
Total	2240

*Frame: includes centre plates, landing gear, battery holder, arms and nuts/bolts
**IMU Assembly: includes MEMS IMU, logic level converter, carbon fibre mounting plate
***Camera not included

4.5 Computer Aided Design

A CAD model was made using CATIA v5 to model the inertial parameters accurately and form a graphical representation of the intended product. This step is crucial to the development as it allows us to model the structural parameters of the quadrotor precisely and any modifications to the hardware can be updated in the CAD model which allows us to quickly compute the updated inertial parameters.

Fig. 4.13 shows the computer generated model of the quadrotor.

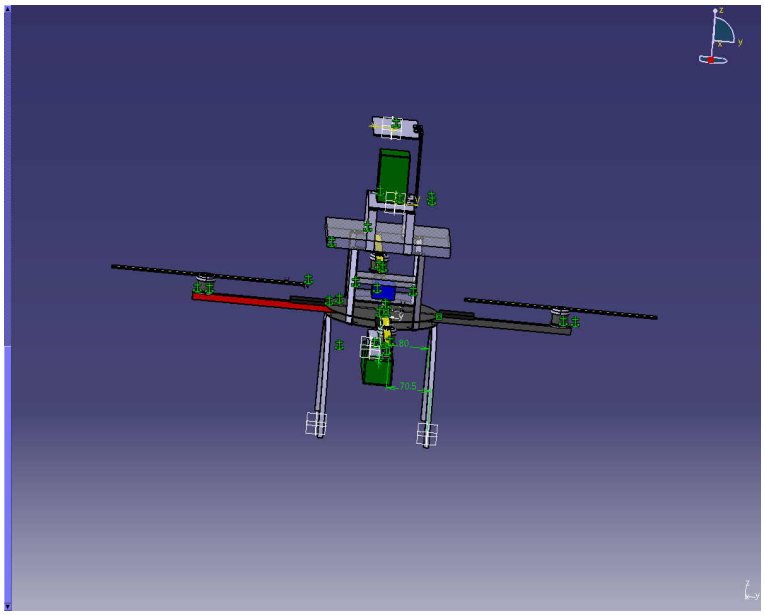


Figure 4.13: CATIA model of quadrotor

Table 4.4: Inertial Parameters

Parameter	Estimated Value ($kg \cdot m^2$)
I_{xx}	0.034
I_{yy}	0.034
I_{zz}	0.035

CHAPTER 5

Quadrotor Modeling & Control

5.1 Quadrotor Basics

The quadrotor design has four fixed-pitch propellers in cross configuration. Driving the two pairs of propellers in opposite directions removes the need for a tail rotor. Consequently, vertical rotation is achieved by creating an angular speed difference between the two pairs of rotors. Increasing or decreasing the speed of the four propellers simultaneously permits climbing and descending. Rotation about the longitudinal and the lateral axis and consequently horizontal motions are achieved by tilting the vehicle. This is possible by conversely changing the propeller speed of one pair of rotors. In spite of the four actuators, the quadrotor remains an under-actuated and dynamically unstable system. In fact, MAV class quadrotors require a very small rotor diameter which is very penalizing in terms of efficiency. However, the inherent simple mechanics of quadrotors and the increased payload are their main advantage in the MAV class.

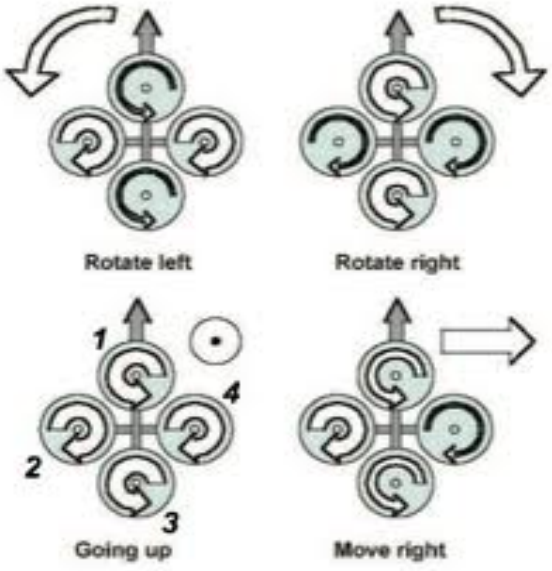


Figure 5.1: Quadrotor Principle

The method followed here was to first identify the components to be used by working with the baseline specifications and product data sheets. Once that was achieved, the individual components were purchased and assembled. The inertial parameters of the vehicle were determined for dynamic modelling using the CAD model and only the dynamics of the actuators which are important in the case of a quadrotor were identified. This approach makes it easy to build dynamic models of unstable systems, since we do not have to perform closed loop identification in flight. Newton-Euler formalism was used to model the dynamics of the vehicle (Appendix A). Blade Element Theory and Momentum Theories were used to model the BumbleBee quadrotor. A detailed description of the dynamic modelling and derivation of the equations of motion is provided in Appendix B.

5.2 System Identification

Before, the quadrotor can be simulated in a virtual environment, certain characteristics of its dynamics need to be identified. These include, the Thrust and Drag factor, motor dynamics and body inertial matrix. The more accurately these parameters are identified, the closer the model behaves to the real system.

5.2.1 Inertial Parameters

The hardware integration was conducted in CATIA to build a close-to-reality representation of the final product. It enabled calculation of the inertial parameters which are required for dynamic modelling and simulation. The total mass of the quadrotor is 2.297 kg which is very close to the initial design weight.

Table 5.1: Moments of Inertia

Parameter	Estimated Value ($kg \cdot m^2$)
I_{xx}	0.034
I_{yy}	0.034
I_{zz}	0.035

5.2.2 Aerodynamics

Here, the methods used to identify the aerodynamic parameters (b-Thrust factor & d- Drag Factor) for the BumbleBee quadrotor are described. The propeller's rotation generates lift and drag which contribute to the dynamics of the vehicle. The thrust generated is used to control the roll and pitch motion whereas the drag generated contributes to the yaw.

A test rig was set up to measure the thrust generated by the propellers in a static condition. The motor was bolted to a heavy block of wood as shown in Fig. 5.2. Then, this block was placed on a weighing machine and the propeller RPM was varied over a wide range. The change in the reading of the weighing scale gives a direct measure of the propeller thrust.

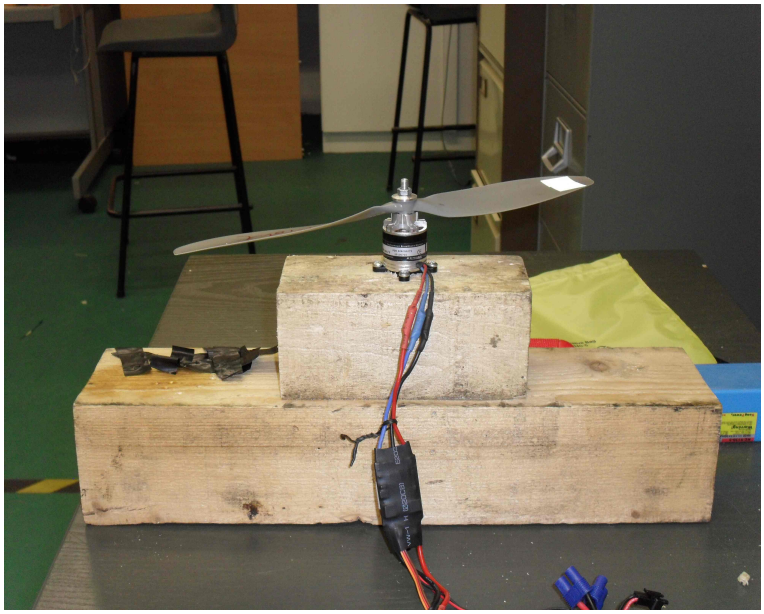


Figure 5.2: Motor bolted to wooden block as part of thrust testing rig

From Blade Element Theory as described in Appendix A, the lift generated by a propeller is described by Eqn. A.2.9 stated here again for clarity. It shows clearly that the thrust is directly proportional to the square of the angular velocity of the prop.

$$L = N_B \rho_A a c \omega_p^2 R_p^3 \left(\frac{\theta_{I_0}}{6} - \frac{\theta_{tw}}{8} - \frac{\lambda}{4}\right) \tag{5.2.1}$$

Fig. 5.3 shows the measured values. It is seen that the thrust is related linearly to the PWM signal value. From this, it can be inferred that the square of the angular velocity should have a linear relationship with the Pulse Width Modulation (PWM) signal value (duty cycle [μs]) as well. Fig. 5.4 reveals that indeed, that is true.

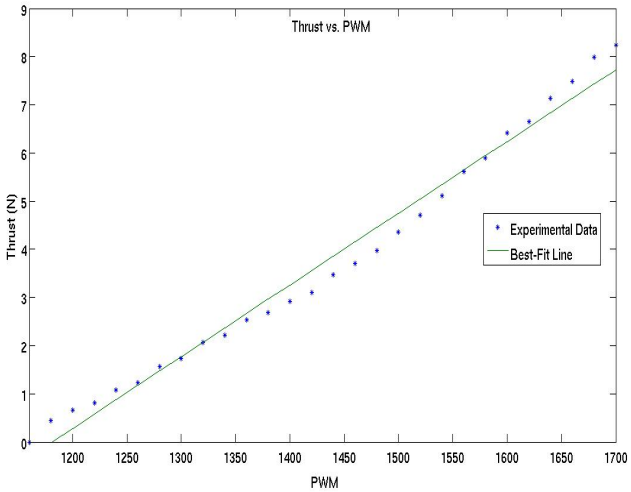


Figure 5.3: Thrust vs. PWM

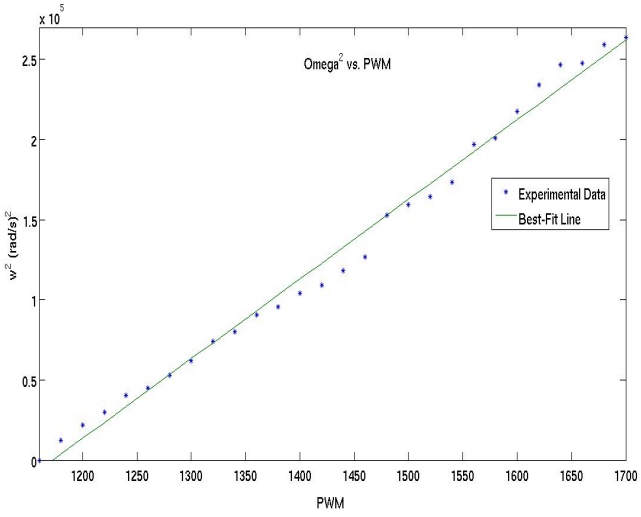


Figure 5.4: ω^2 vs. PWM

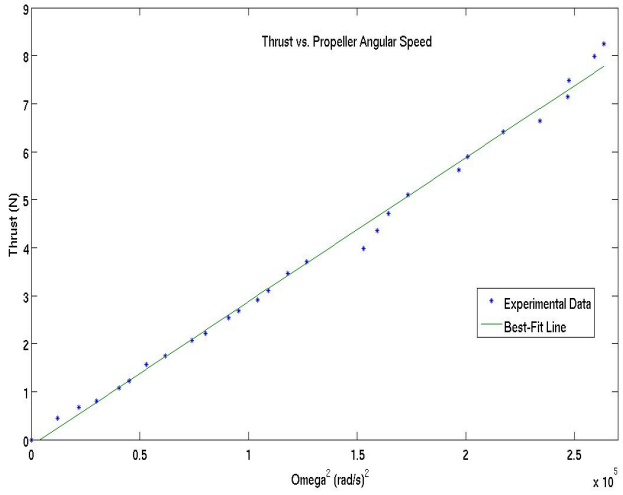


Figure 5.5: Thrust vs. ω^2

The thrust factor b, can be written as :

$$b = \frac{L}{\omega_p^2} = N_B \rho_A a c R_p^3 (\frac{\theta_{I_0}}{6} - \frac{\theta_{tw}}{8} - \frac{\lambda}{4}) = 2.998 \times 10^{-5} [N \cdot s^2] \tag{5.2.2}$$

To estimate the drag factor d, it is necessary the describe the torque acting on the shaft as a function of the angular speed. The relationship is derived from fundamental aerodynamic calculations in Eqn. A.2.11 is stated here again.

$$Q = N_B \rho_A a c \omega_p^2 R_p^4 (\frac{C_D}{8} + a\lambda(\frac{\theta_{I_0}}{6} - \frac{\theta_{tw}}{8} - \frac{\lambda}{4})) \tag{5.2.3}$$

The drag coefficient C_D was estimated from [20] as 0.05. λ is the inflow ratio calculated from momentum theory as described in Eqn. A.1.8.

$$\begin{cases} \lambda = \frac{v_i}{\omega_H R_P} \\ W_P = 2\rho_a A v_I^2 \end{cases} \tag{5.2.4}$$

Where, ω_H is the angular speed of the blade in hovering condition and R_P is its radius. W_P is the weight carried by one propeller, ρ_a is the air density, A is the disc area of the propeller. λ is calculated from the the experimental data to be 0.0215. Thus, the drag factor d is estimated to be :

$$d = \frac{Q}{\omega_p^2} = N_B \rho_A a c R_p^4 (\frac{C_D}{8} + a\lambda(\frac{\theta_{I_0}}{6} - \frac{\theta_{tw}}{8} - \frac{\lambda}{4})) = 1.742 \times 10^{-6} [N \cdot m \cdot s^2] \tag{5.2.5}$$

5.2.3 Motor Dynamics

The motor dynamics can be modelled as a first-order transfer function between the propeller's set point and its true speed. This assumption was validated with an experimental setup using a laser tachometer as shown Fig. 5.6. The setup consists of a handled tachometer used to focus a beam of laser onto a highly sensitive photo-transistor whose output is connected to a highly sensitive oscilloscope. The photo-transistor circuit's output voltage goes high when the blade of the propeller interrupts the laser beam. Thus in this way, the dynamic response can be recorded and post processed using the Matlab System Identification toolbox.

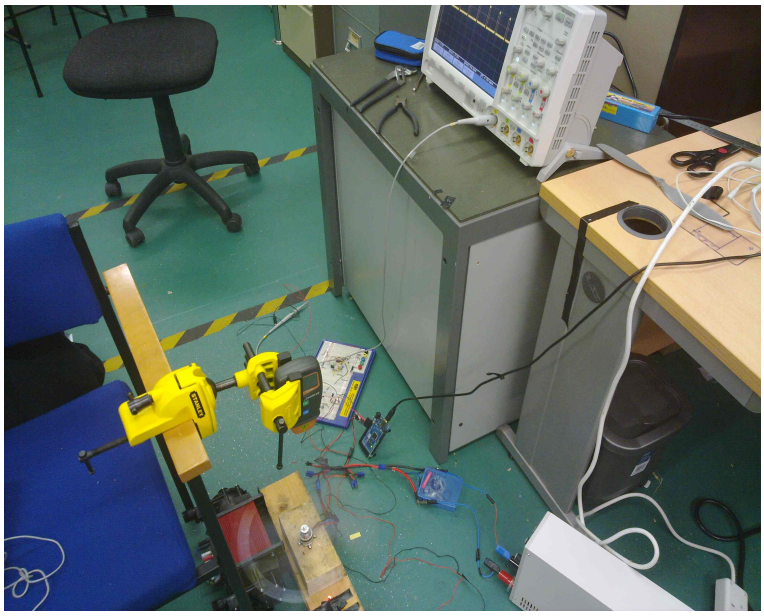


Figure 5.6: Motor dynamic response testing rig

From the data, the transfer function was identified as,

$$G(s) = \frac{0.9939}{0.31744s + 1} \tag{5.2.6}$$

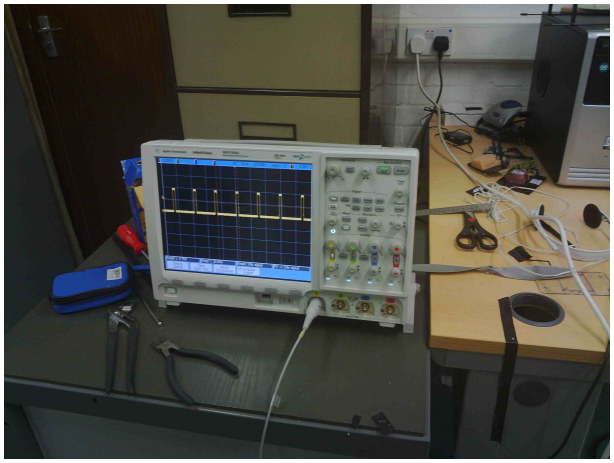


Figure 5.7: Propeller angular velocity as seen on oscilloscope

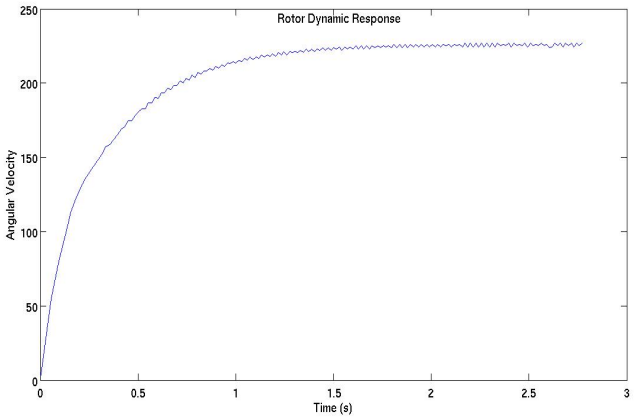


Figure 5.8: Propeller dynamic response

5.3 Controller Development

The dynamic model presented in Eqn. B.3.18 contains gyroscopic effects and rotational coupling. These effects can be neglected in a near hover situation as the effect of the thrust and drag action is more significant. In order to design multiple PID controllers for this system, once the gyroscopic effect and cross-coupling are removed, the attitude and altitude equations described in Eqn. B.3.18 can be re-written as,

$$\begin{cases} \ddot{\phi} = \frac{U_2}{I_{xx}} \\ \ddot{\theta} = \frac{U_3}{I_{yy}} \\ \ddot{\psi} = \frac{U_4}{I_{zz}} \\ \ddot{Z} = -g + cos(\theta)cos(\phi)\frac{U_1}{m} \end{cases} \tag{5.3.1}$$

The quadrotor attitude control systems uses a cascading PID controller for roll and pitch stabilisation. Cascading PID has been shown to have a faster response to control inputs. It can be mathematically proven that the working frequency of the controller is increased and the time constant of the object is reduced by using cascaded PID controller.

Fig. 5.9 shows the cascading PID controller used for roll (ϕ). Fig. 5.10 shows a PID controller used for Yaw. The vehicle dynamics block also contains the actuator dynamics.

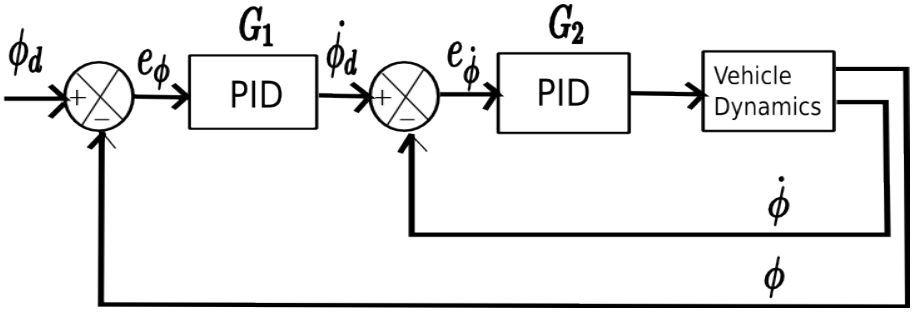


Figure 5.9: Cascading PID controller used for roll and pitch stabilisation

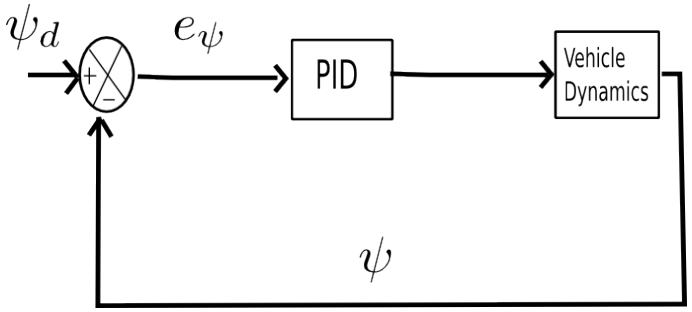


Figure 5.10: Simple PID controller used for yaw stabilisation

5.3.1 Simulation Results

1. **Roll:** A cascading PID controller was simulated with the model described in Eq. 5.3.1 using the system parameters as identified in the various tests in section 6.2. The controller was commanded to drive the vehicle to a roll angle of 10 degrees. Fig. 5.11 shows the simulation results

   The paramaters for the first controller G_1 are $K_P = 2.3,\ K_I = 0,\ K_D = 0$ and for G_2 are $K_P = 0.9,\ K_I = 0,\ K_D = 0.09$. Since the quadrotor is symmetrical about the X and Y axes, the control gains are the same for roll and pitch.

2. **Yaw:** A PID controller was simulated for yaw rate command tracking. Fig. 5.12 shows the simulation result for tracking a yaw rate of 10 degrees per second.

   The paramaters for the yaw rate controller are $K_P = 2.0,\ K_I = 0,\ K_D = 0.01$.

3. **Altitude:** A PID controller was developed for Altitude control. The quadrotor is commanded to hold an altitude of 50 cm. Fig. 5.13 shows the results. The parameters for the altitude controller are $K_P = 0.18,\ K_I = 0.08,\ K_D = 0.6$.

4. **Speed:** A PID controller for forward speed control was developed (since inertia matrix is symmetric, same gains can be used for lateral speed control). The quadrotor is commanded to attain a speed of of 0.50 m/s (max. operating speed, determined for similar

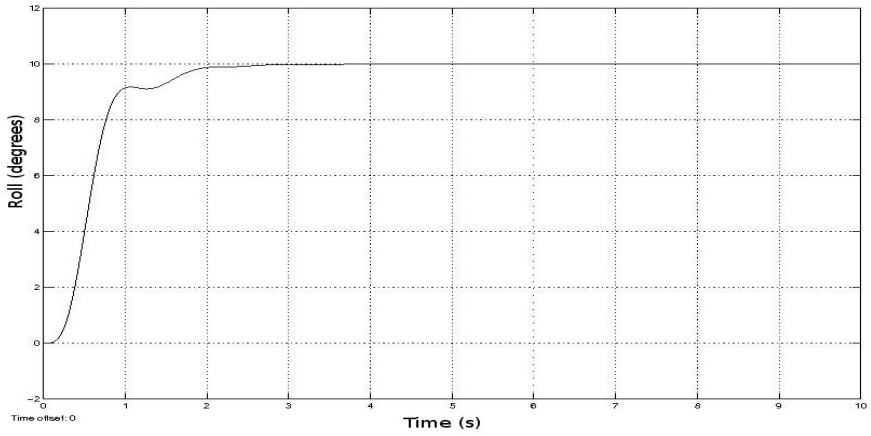


Figure 5.11: Simulation results for roll stabilisation

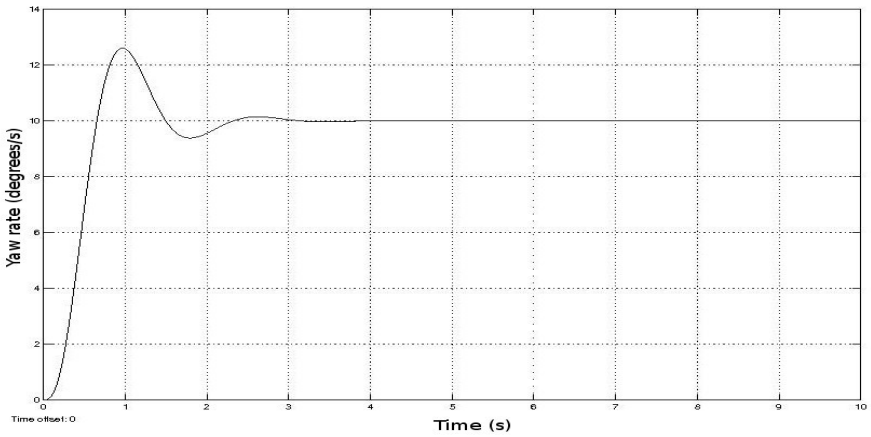


Figure 5.12: Simulation results for yaw rate tracking

use in [16]. Fig. 5.14 shows the results. The paramaters for the altitude controller are $K_P = 4.0,\ K_I = 0,\ K_D = 0.05$.

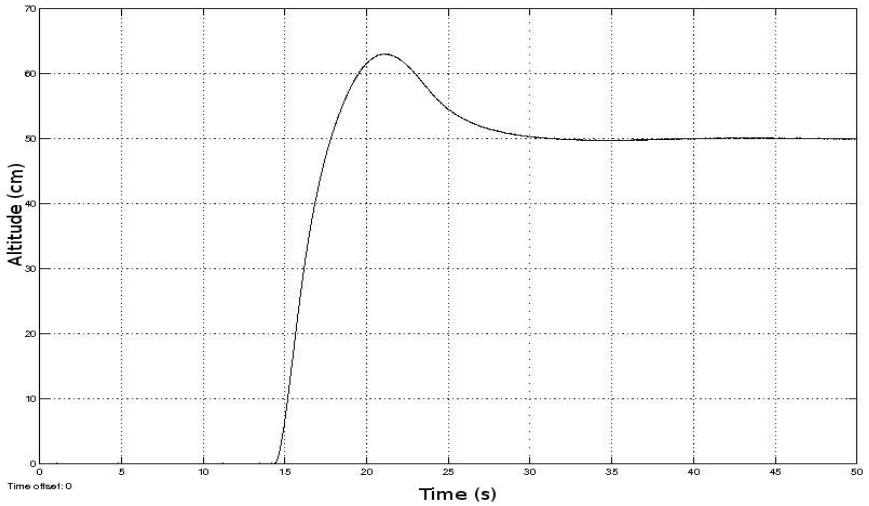


Figure 5.13: Simulation results for altitude hold

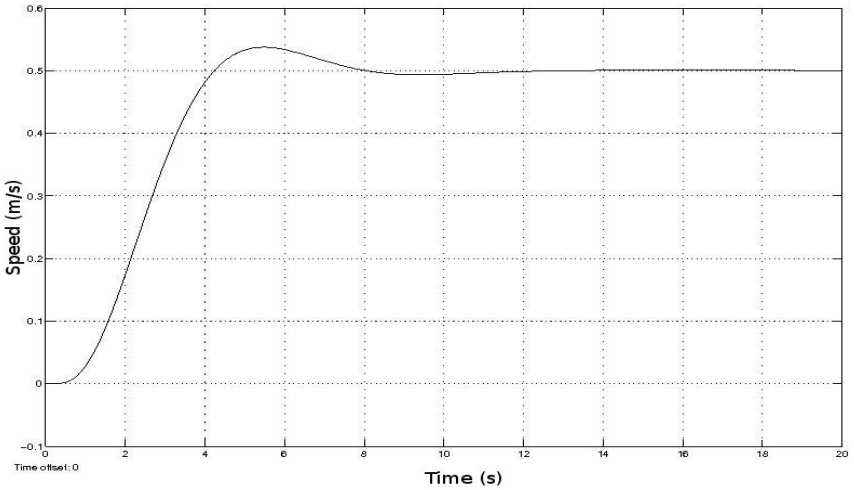


Figure 5.14: Simulation results for speed hold

5.3.2 Rig Test Results

Before the quadrotor can be flown, the performance of the designed controllers must be tested in a safe manner to reduce risk of damage to the vehicle in flight. This also allows further tuning of the PID gains to improve vehicle performance. Therefore, a test rig as shown in Fig. 5.15 was used to analyse the roll, pitch and yaw stability. Tests were conducted about each individual axis, the gains were tuned further and the corresponding data was recorded using a serial connection between the PC and the on-board controller.

Fig. 5.16 shows the response of the quadrotor to pilot commands about the X axis (roll). Fig. 5.17 shows the response of the quadrotor to pilot commands about the Y axis (pitch).

An integral term had to be introduced to reduce steady state error. Thus, the final gains for pitch stability were $K_P = 2.75,\ K_I = 0.5,\ K_D = 0$ for G_1 and $K_P = 0.9,\ K_I = 0,\ K_D = 0.05$ for G_2. For roll stability the control gains were $K_P = 2.3,\ K_I = 0.5,\ K_D = 0$ for G_1 and

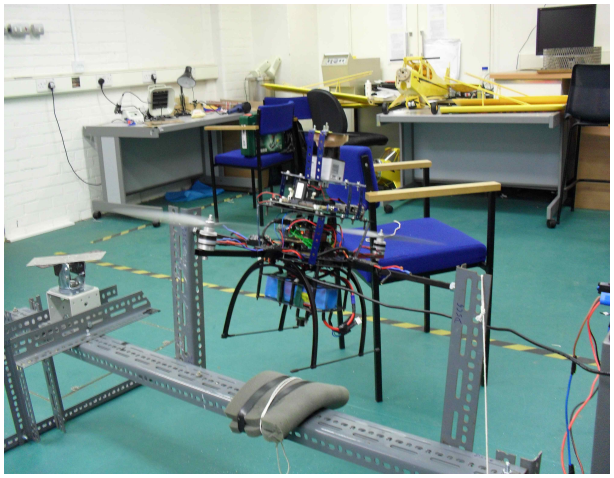


Figure 5.15: Quadrotor Stability Test Rig

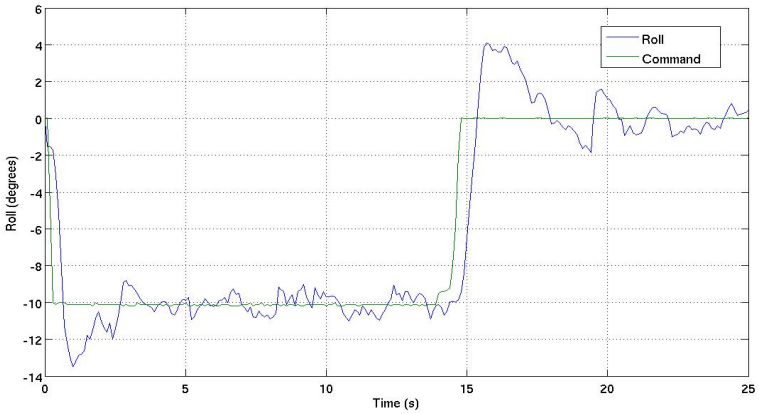


Figure 5.16: Quadrotor roll stability on rig

$K_P = 0.9,\ K_I = 0,\ K_D = 0.09$ for G_2.

Fig. 5.18 shows the stabilisation results in Yaw. The quadrotor is disturbed to a heading of -90 degrees and the desired heading is -108.5 degrees. The PID controller for heading hold has the gains $K_P = 1.5$, $K_I = 0.9$ and $K_D = 0.01$.

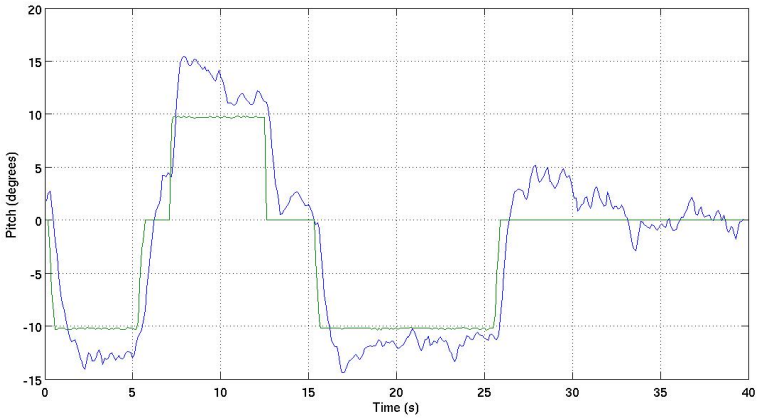


Figure 5.17: Quadrotor pitch stability on Rig

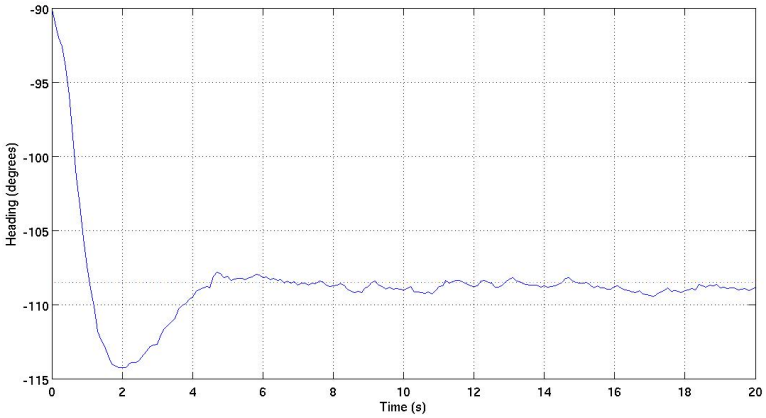


Figure 5.18: Quadrotor yaw stability on rig

CHAPTER 6

Autonomous Navigation

To navigate in an unstructured indoor environment the quadrotor must be able to safely avoid obstacles in a highly dynamic environment within its own kinematic constraints. The operator's job is to provide the on-board computer with a target location. Using the laser scans from the on-board LIDAR and state estimates from the localisation algorithm, the real-time navigation algorithm running on the on-board computer must be able to work out motion commands required to navigate to the target location. It must be kept in mind that the vehicle is not provided any prior information about its surroundings. Its entire knowledge is built up in real-time from the on-board sensors. While the SLAM algorithm estimates the vehicle's state and maps the visual features in 3-D, it is up to to the 'motion planning' algorithms to guide the vehicle to its destination.

If a vehicle is to successfully traverse through an unknown environment, it must be able to use its sensor measurements to quickly react to the dynamics of its surroundings and plan its motion accordingly. This problem has been extensively researched in the robotics community. On the one side there are approaches which generate an optimal path from a start point to a target location for a known environment. The *Configuration Space* (C-Space) has been successfully employed as a representation in this scope. In this representation, the robot is treated as a single point. Then, on the other side, there are methods which are able to deal with dynamical environments, calculating the motion commands periodically in real-time. These approaches are called *Reactive Navigation* systems.

For the purpose of this project, a reactive navigation method developed by Blanco et al. [18] has been adopted. In their paper, Blanco et al. [18] have described a method which overcomes the issues of kinematic restrictions and real-time operation in a dynamic environment. They have achieved this by decoupling the kinematic restrictions and obstacle avoidance using path models to transform from kinematic-compliant paths and real-world obstacles into lower complexity space called *Trajectory Parameter Space* (TP-Space). The task of obstacle avoidance in the transformed space is relegated to a Nearness Diagram based approach. Blanco et al. [18] have generalized path models through a novel tool called *Parametrized Trajectory Generator* (PTG) which permits handling any number of transformations and they propose a navigation system which manages multiple paths simultaneously, each corresponding to a different PTG. Their system makes the best selection at each instant of time (in terms of path length and clearance) which yields a more robust approach than traditional methods relying on circular paths. In the proceeding section, a brief description of the method described in [18] is presented. Certain key equations from [18] are reproduced here for the reader's benefit.

6.1 Reactive Navigation Using Parametrized Trajectory Generators

The first and most important step in obstacle avoidance is to calculate the distance to obstacles. This provides the necessary information to the robot for choosing its next movement. Blanco et al. describe the distances directly in C-Space. The possible paths, when seen in C-Space form 3D surfaces (x,y,pose). These surfaces are called *sampling surfaces* because the distance to an obstacle can be computed as the distance from the current robot pose to the intersection of the sampling surface with obstacles in C-Space. Blanco et al. [18] reduce this 3 dimensional space to 2 dimensions called the TP-Space. On the TP-Space, each point corresponds to a robot pose lying on a C-Space sampling surface.

TP-Space representation uses polar coordinates with angular component α and distance d. α represents the trajectory mapped from C-Space to TP-Space, while d determines the distance of the pose along the selected trajectory. Blanco et al. state that *"TP-Space transforms the free-space detection problem for non-holonomic robots from C-Space into a collision avoidance problem in transformed space".* The transformation is applied at each step of the navigation process, thus the robot is always at the origin while surrounding obstacles are mapped to the unit circle.

A Parametrized Trajectory Generator (PTG) maps points in polar coordinates (α,d) from TP-Space to poses (x,y,ϕ) in C-Space such that straight paths from the origin in TP-Space are transformed into kinematically compliant paths. A PTG is defined by Blanco et al. [18] as:

$$\begin{cases} PTG : A \times D \subset \Re \rightarrow \Re \times S^1 \\ (\alpha, d) \rightarrow (x, y), \phi \end{cases} \tag{6.1.1}$$

Where $A = \alpha | \alpha \in [-\pi, \pi]$ and $D = d | d \in [0, 1]$ define the domain of the PTG and S^1 is the circular topology of the robot's heading.

The robot's path is mapped from TP-Space into C-Space in terms of trajectories. Thus, time is introduced to substitute the distance component d. If $V(\alpha, t) = [v(\alpha, t)\ \omega(\alpha, t)]^T$ be the velocity vector for a given trajectory where the components are the linear and angular velocity. Then, the functions $v(\alpha, t)$ and $\omega(\alpha, t)$ are called *"Design Functions"* by Blanco et al. [18], in the context of a PTG, since they are used to design the trajectory. If $P(\alpha, t) = [x(\alpha, t)\ y(\alpha, t)\ \phi(\alpha, t)]^T$ is the robot pose at an instant, then these poses are calculated by the integration of the kinematic constraint equation (refer to [18]).

To ensure that the PTG given by a design function is valid, certain conditions are imposed by Blanco et al. [18]:

1. *"An arbitrary path model is not applicable to reactive navigation as the decision process does not rely on past information, hence it must generate consistent reactive trajectories."*
2. *"Only one trajectory can exist which takes the robot from its current state to any other WS location (x, y), regardless of the orientation. Otherwise, the target position would be observed at two different directions (straight lines) in TP-Space."*
3. *"The trajectory must be continuous."*

To perform reactive navigation in TP-Space, two transformations are performed. The obstacles need to be mapped to TP-Space and the goal location has to be translated to TP-Space. The target/goal location is transformed to TP-Space using the inverse PTG function. Blanco et al.

assume obstacles to be points (true for Laser Scanners). These points are mapped to TP-Space as regions called TP-Obstacles [18]. Only those C-Space obstacles that intersect with a sampling surface are transformed to TP-Space. This is done to keep computational load low.

The robot's physical environment is treated like a rectangular grid whose individual cells store their associated TP-Obstacle, built from the set of pairs (α,d) that lead to collision. To allow the robot to traverse through narrow passages, the grid must provide a high resolution (e.g. 1 to 2 cm).

The motion command generated in the TP-Space is a pair giving the desired speed and direction, this motion command is mapped to a real robot movement in two steps as described in [18]:

1. *"Obtain a normalized velocity command as* $V_{norm}(\alpha_m) = v(\alpha_m, 0) = [v(\alpha_m, 0)\ \omega(\alpha_m, 0)]^T$ *, through the evaluation of the PTG design functions at* $\alpha = \alpha_m$*. Take the initial response (at t=0) since the PTG reference system is the robot current pose, i.e. the robot is always at the origin of trajectories in the TP-Space"*

2. *"The velocity command for the real robot* V_{rob} *is computed by scaling* V_{norm} *according to the holonomic velocity s in TP-Space (provided by the holonomic method)."*

6.1.1 Nearness Diagram Based Obstacle Avoidance

The obstacle avoidance method used in [18] is based on the Nearness Diagram Approach developed by Minguez and Montano [35]. The obstacle avoidance system relates the relative positions of the robot, obstacles and target within a set of defined situations. These situations help classify the kind of motion that the robot should undertake based on certain safety parameters, namely a security zone around the robot. [35] Relate the the robot and goal locations by means of the *"free walking area"*. The *"free walking area"* is extracted as follows. First, search for open spaces in the obstacle distribution, and determine the regions from two connected gaps. Next, the region nearest to the target is selected and checked whether it is navigable by the robot. Then, these relations are used to define the set of situations which are resolved using the decision rules formulated by Minguez and Montano [35]) as follows:

1. *"Safety criterion. Depending on whether obstacles are present in the safety zone. There are two safety situations, Low Safety or not High Safety. In Low Safety, the first two situations are obtained by applying the next criterion"*

2. Dangerous obstacle distribution criterion

   (a) *"Low Safety 1 (LS1): The robot is said to be in LS1 when the obstacles in the security zone are only on one side of the gap (closest to the goal) of the free walking area "*

   (b) *"Low Safety 2 (LS2): The robot is said to be in LS2 when the obstacles in the security zone are on both sides of the gap (closest to the goal) of the free walking area"*

3. Goal within the free walking area criterion

   (a) *"High Safety Goal in Region (HSGR): The robot is in HSGR when the goal location is within the free walking area"*

4. *Free walking area width criterion. A free walking area is wide if its angular width is larger than a given angle, and narrow, otherwise."*

   (a) *"High Safety Wide Region (HSWR): The robot is in HSWR when the free walking area is wide "*

(b) *"High Safety Narrow Region (HSNR): The robot is in HSNR when the free walking area is narrow "*

The actions of the robot are defined as follows in [35]:

1. *"Low Safety 1 (LS1): This action moves the robot away from the closest obstacle, and toward the gap (closest to the goal) of the free walking area"*
2. *"Low Safety 2 (LS2): Centers the robot between the two closest obstacles at both sides of the gap (closest to the goal) of the free walking area, while moving the robot toward this gap"*
3. *"High Safety Goal in Region (HSGR): Drives the robot toward the goal "*
4. *"High Safety Wide Region (HSWR): Moves the robot alongside the obstacle "*
5. *"High Safety Narrow Region (HSNR): Directs the robot through the central zone of the free walking area"*

At each instant, the action resolves the reactive navigation task, which is to avoid obstacles while moving the robot toward the target. In Low Safety, it is achieved because both actions avoid the obstacles while moving the robot toward the gap (closest to the goal) of the free walking area. In High Safety, since there is no imminent danger, there is no need to avoid collisions. The actions drive the robot toward the target, toward the gap of the free walking area, or toward the central zone of the free walking area.

6.2 Complete Reactive Navigation Sysyem

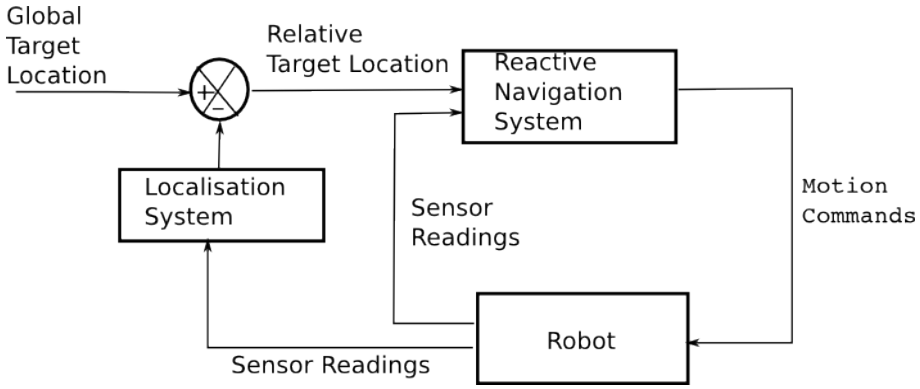


Figure 6.1: System description of autonomous navigation [18]

The complete system is made up of a control loop where the sensor readings along with the estimate of the target's relative location (from SLAM algorithm) are supplied to the reactive navigation system (Fig. 6.1). The reactive navigation system generates a velocity and turn rate command that is sent back to the robot, closing the control loop. This process is repeated periodically (e.g. at 10Hz), which steers the robot towards the target simultaneously responding to dynamic obstacles. The target location is given in a fixed coordinate system i.e. the global frame initialized in the SLAM filter.

6.3 Simulation Results

To test the applicability of the above mentioned algorithm, a C++ script was written using the MRPT libraries to simulate the reactive navigation system in a virtual environment. The quadrotor is modelled as a circular robot with radius equal to the sum of its arm length, radius of propeller and a safety margin. The safety margin is decided at 50 % of the arm length. This safety margin is kept high to avoid a near-miss situation. Fig. 6.2 to Fig. 6.5 shows the vehicle in its simulated environment traversing to a goal location marked as 'X'. The simulation time step is kept at 100 ms (10 Hz) which is the frequency of the navigation algorithm in real-time operation. The speed of the quadrotor is limited to 0.5 m/s for safe operation and angular speed is restricted to 15 degrees per second. Besides safe operation, a slow operating speed would prevent motion blur in the camera image which can cause the visual SLAM algorithm to lose track of features. Fig. 6.6 and 6.7 show the speed and angular velocity of the quadrotor respectively during simulation. The reactive navigation algorithm generates speed and turn rate commands which are sent to the attitude controller to convert to attitude commands.

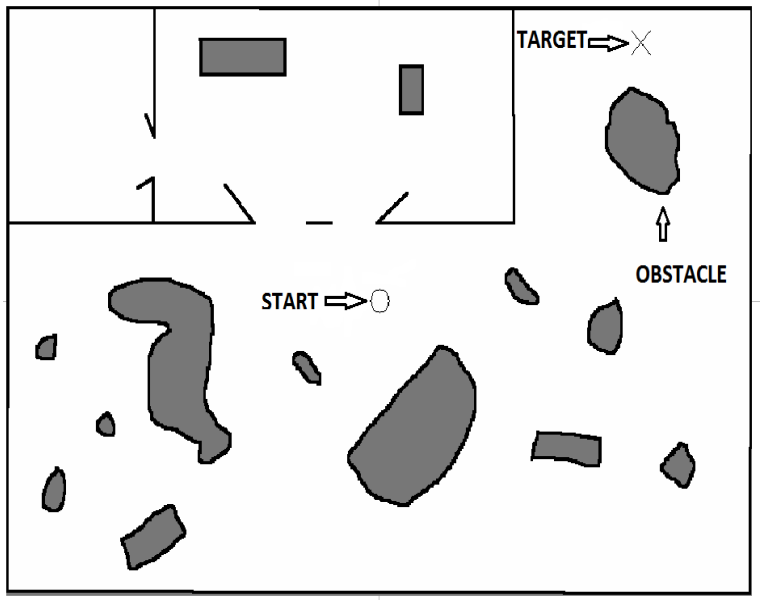


Figure 6.2: Starting: Simulation of reactive navigation for quadrotor in a virtual environment (**X** represents target while **O** the quadrotor)

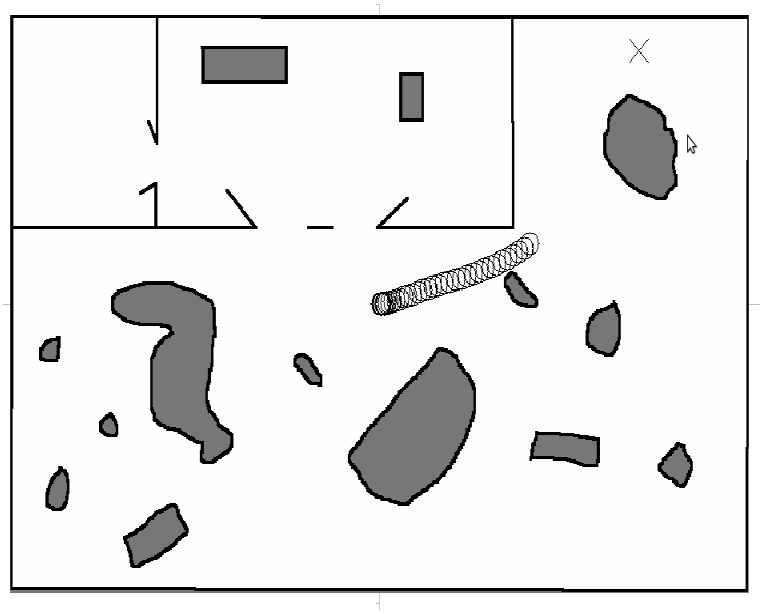


Figure 6.3: En Route: Simulation of reactive navigation for quadrotor in a virtual environment (**X** represents target while **O** the quadrotor)

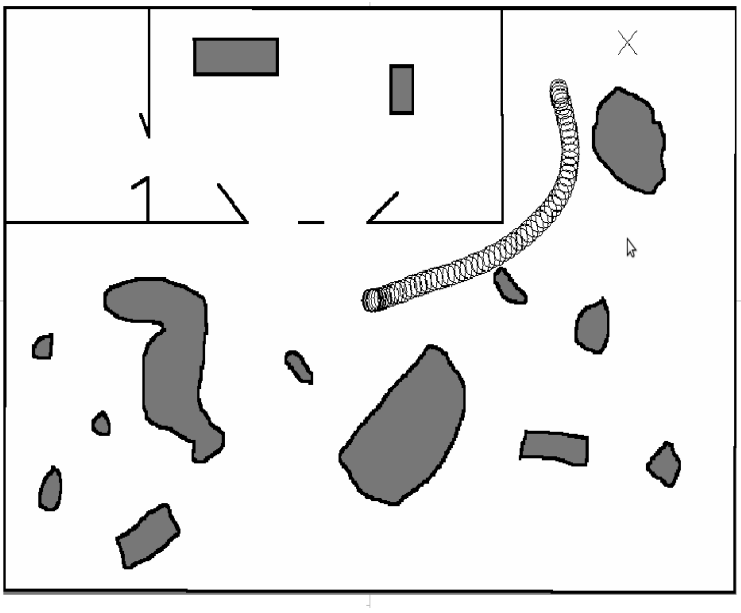


Figure 6.4: En Route: Simulation of reactive navigation for quadrotor in a virtual environment (**X** represents target while **O** the quadrotor)

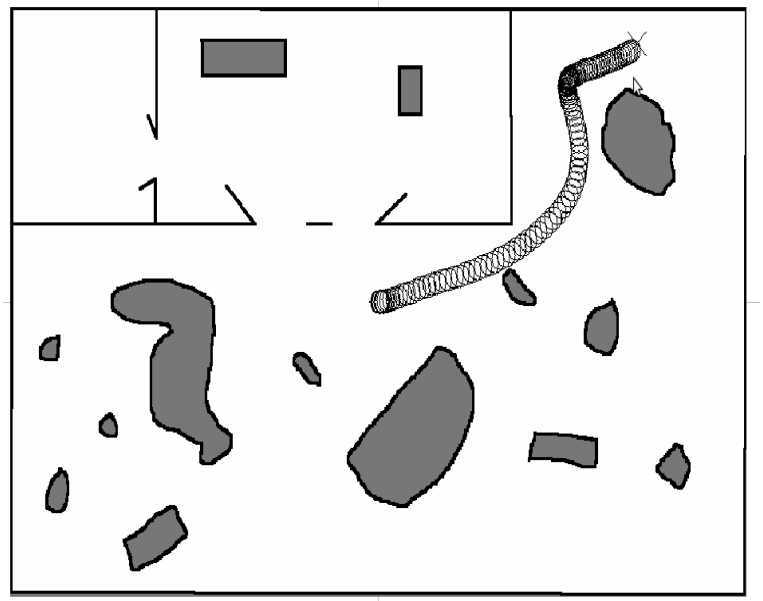


Figure 6.5: Reached Target: Simulation of reactive navigation for quadrotor in a virtual environment (**X** represents target while **O** the quadrotor)

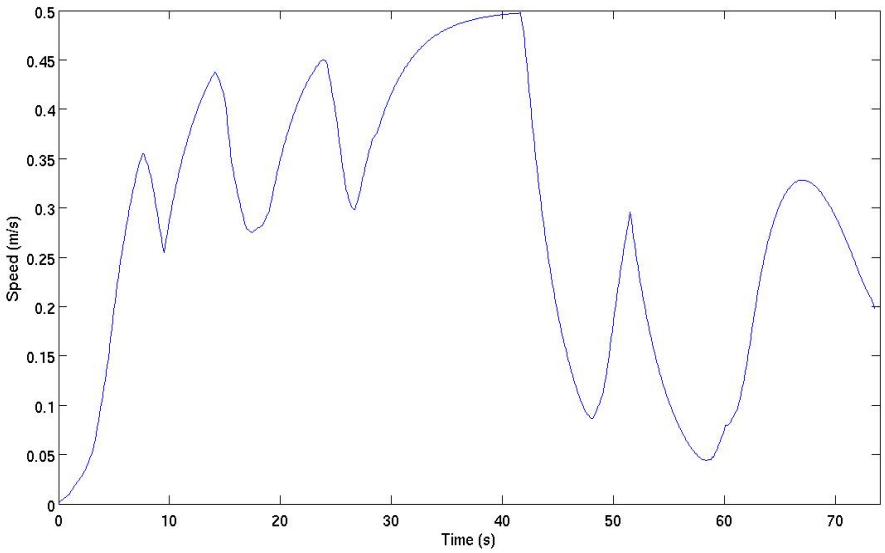


Figure 6.6: Quadrotor speed during navigation

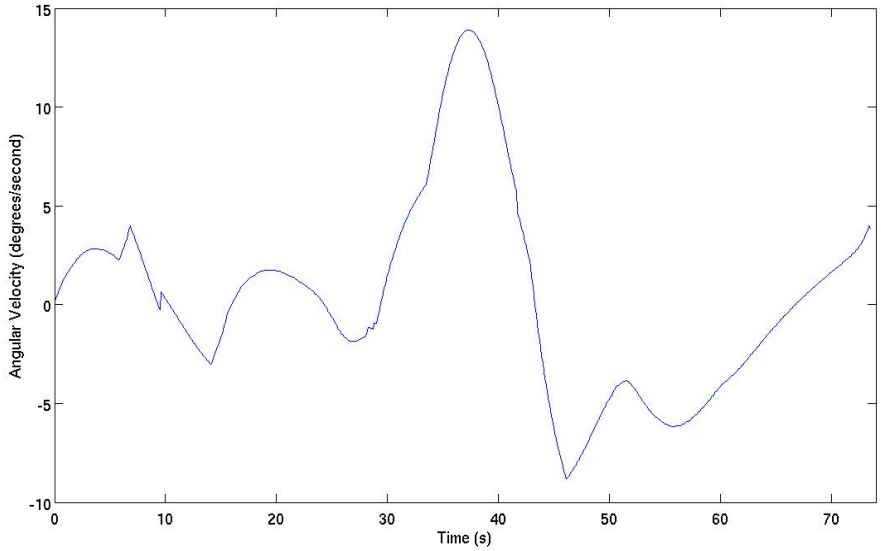


Figure 6.7: Quadrotor angular velocity during navigation

CHAPTER 7

System Integration and Flight Testing

Section 1.4 introduced the three level control strategy that has been proposed for the quadrotor. At the lowest level (CL_1), the Attitude Controller performs the hover stabilisation, at control level CL_2 the visual SLAM algorithm performs the global localisation and state estimation using the video feed from the monocular camera and the IMU data. The velocity estimates are fed to the velocity feedback controller which generates the requisite attitude command for the attitude controller to follow. At the top most level, the reactive navigation algorithm accepts distance measurements from the LIDAR and computes the velocity and yaw rate commands. The yaw rate commands are fed directly to the attitude controller.

Fig. 7.1 shows the overall system hierarchy design.

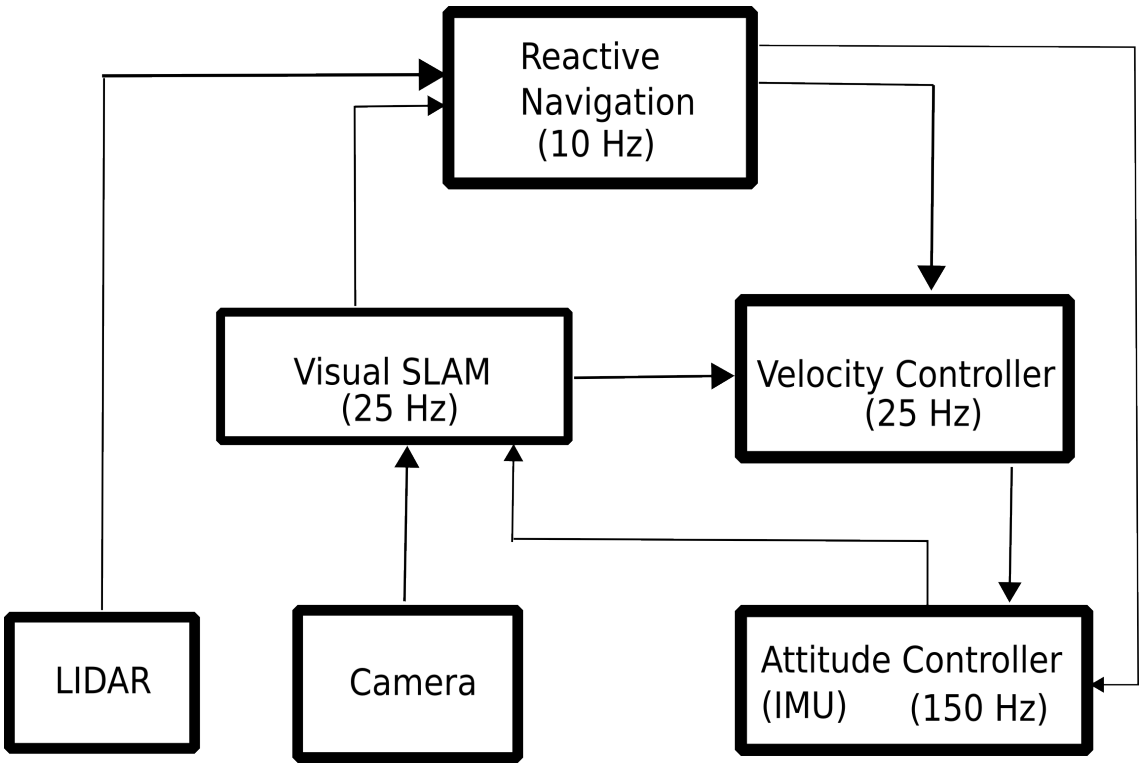


Figure 7.1: Hierarchal control system architecture

Figure 7.2: The fully assembled BumbleBee quadrotor

7.1 Systems Integration

All the individual components described in Section 4.3 needed to be integrated with each other for the quadrotor to function properly. This section describes the different kinds of software/hardware techniques developed to interface all the sensors and actuators with their respective controllers and the communications protocol developed to share information between the Arduino Controller, Single Board Computer and Ground Control Station.

7.1.1 IMU

The SBG Systems IG-500A Inertial Measurement Unit can work on a power supply of up to 30V and draws 10 mA of current. Thus, it was powered using the 5 V output available from the voltage regulator on-board the Arduino Mega. One key hurdle faced was that the device outputs data using a serial RS-232 protocol, however, the Arduino Mega's serial communication ports only accept TTL signals. Therefore, a converter had to built using a serial level converter chip (MAX232CPE) as shown in Fig.7.3. This converter, accepts RS-232 signals from the inertial sensor and converts it to a TTL output which is sent to the Arduino Mega's *Serial 1 RX* pin (Pin 19).

The IMU is configured using the vendor provided Windows based software configuration tool to output Kalman Filtered (150 Hz) Euler Angles, Angular Rates and Linear Accelerations continuously at a baud rate of 115200 bps. Using the low-level communications protocol provided as part of the software development kit, a C++ based code was developed which uses the *Serial Communication Library* of the Arduino development platform to decode the data transmitted by the IMU. Fig. 7.4 shows the format of the data buffer transmitted by the IMU. All measurements are transmitted as 4 byte float values.

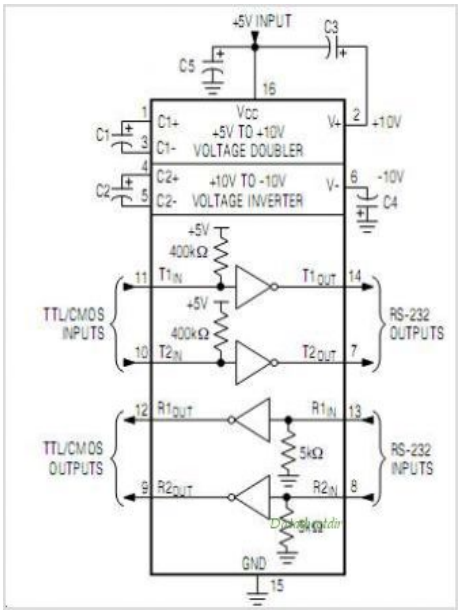


Figure 7.3: MAX232CPE functional diagram

Field	SYNCX	STX	CMD	LEN (MSB)	LEN (LSB)	DATA	CRC (MSB)	CRC (LSB)	ETX
Size (bytes)	1	1	1	1	1	0 to 504	1	1	1
Description	Sync. byte	Start of Tx byte	Command	Length of the DATA section		Additional data	CRC		End of Tx byte
Value	0xFF	0x02	-	-	-	-	-	-	0x03

Figure 7.4: Serial data buffer format

7.1.2 Radio Receiver

The AR7000 receiver is powered through its throttle output port which is connected to the 5 V supply generated by the ESC. It outputs PWM signals corresponding the commands sent by the transmitter. These signals are sent to the analog inputs pins of the Arduino Mega (Pins A7-A14) which are connected to the 16 bit A/D converters of the Atmega 2560 chip. To speed up the process of reading the PWM signals, interrupt handlers are used to measure the time interval between the signal high and low voltages to calculate the duty cycle. This helps lower the latency of the controller. The Arduino board reads the receiver signals at a rate of 25 Hz.

7.1.3 Motors

The 4 motors are the sole actuators available to control the attitude and velocity of the quadrotor. These motors are controlled by their respective Electronic Speed Controllers which accept the RPM set point as PWM signals from the Arduino board. The Arduino has 14 PWM output pins of which pins 8,9,10 and 12 are used to command the motors. The *analogWrite* function provided in the Arduino development library is used to generate the PWM command which outputs the signal at a rate of 490 Hz. The motor command values are updated at the same rate as the main control loop i.e. 150 Hz.

7.1.4 Sonar

There are 3 ways to interface the MaxSonar with the Arduino board.

1. Analog
2. Pulse Width
3. Serial communication

The analog output is used as it is the easiest to interface. The sonar outputs are read at pin A7 at a rate of 5 Hz using the *analogRead* function of the Arduino development library. The sonar outputs a voltage of $V_{CC}/512$ Volts per inch. Since the 5V supply from the Arduino is used, a voltage of 9.8 mV is measured per inch of depth. To remove erroneous measurements, a mode filter is used which works by recording 9 sensor measurements every time the sonar measurement loop is activated (5 Hz).

7.1.5 LIDAR

The LIDAR is powered using the 5V output available from the Electronic Speed Controllers and interface with the onboard computer using a mini-USB cable. The MRPT [9] software library provides convenient functions to access the LIDAR measurements essentially giving it a plug and play capability.

7.1.6 Communication

Serial communication was achieved in two ways:

1. Connecting the Xbee module directly to the Arduino board using the Xbee shield. This configuration is very useful while flight testing as it can be used to set up direct communication between the Ground Control Station and the xbee module for low level data sharing, updating control gains etc. Using the Open Source Aeroquad Serial Communications protocol [1], the *Aeroquad Configurator* software was used to trim the gains during flight and rig-testing.
2. Connecting the Arduino board to the on-board Single Board Computer via a USB cable and subsequently interfacing the on-board computer with the Ground Controller Station using the USB explorer dongle. This configuration is used for autonomous flight, since the Arduino sends the vehicle states to the on-board computer which generates the guidance and navigation commands and sends them back. This increases the data transfer speed from the Arduino board as the USB cable can operate at 115200 bps compared to 57600 bps with the xbee. Then, the on-board computer can communicate with the Ground Control Station to send the vehicle states using the Xbee module at a slower rate since this part of the communication is not crucial to the safety of the vehicle.

A compact serial communication protocol was developed to transfer critical data like vehicle states and guidance commands using the MRPT serial communications library [9] in C++. The protocol works by sending a buffer containing float values separated by colons and each message is initiated with the character "X" and terminated with the character "Y". A typical buffer sent from the Arduino to the on-board computer looks like this: DataBuffer = [X, ϕ, θ, ψ, ω_x, ω_y, ω_z, a_x, a_y, a_z, Altitude, Roll Command, Pitch Command, Yaw Command, Y]. A standard command buffer which holds the velocity, turn rate and Enable/Disable switch value sent from the on-board computer is: Command Buffer: [X, v, ω, Enable/Disable, Y]

7.2 EKF-Based Monocular SLAM in Flight

Whilst, the simulation and small scale real-time performance for EKF-Based visual SLAM was successful, the fast quadrotor dynamics coupled with vibrations made it difficult for the EKF-SLAM to function in flight. The filter faced difficulty in maintaining an accurate track of the visual features and in initialising new features when old ones went out of view. This could be attributed to the fact that, feature position matches exhibited significantly high innovation, which caused the filter to drift. Also, the uncertainty in the depth of the features could not be sufficiently reduced, which led to high uncertainty in the state estimates. Tuning the process noise characteristics, i.e. the standard deviation for linear and angular impulse (a, α) did lead to slight changes in performance, with the filter sometimes keeping track for a long duration before drifting. However, this issue must be fully resolved before EKF based visual SLAM can be used in flight. Further difficulty was caused with the inability of the Fire-i camera to function correctly with the MRPT software libraries. There were issues like automatic exposure variation which completely upset the feature matching step of the SLAM filter. This made it highly difficult to keep track of the features as their appearance changed without change in camera position which affects the motion update step of the SLAM filter.

7.3 Alternate Approach to Visual SLAM in flight

To demonstrate the feasibility of Aerial SLAM. A second set of flight tests was conducted using the *Parallel Tracking and Mapping* (PTAM) methodology developed by Klein et al. [33]. PTAM takes a different approach from the filter based system described in chapter 3. PTAM divides the localization and mapping tasks into two separate threads, i.e. tracking thread and mapping thread. This takes advantage of multi-core processors now available as standard for personal computers. The tracking thread, as the name suggests, tracks the selected visual features in successive frames and calculates an estimate of the relative camera motion. Only FAST corners are tracked and used for the pose estimation. The mapping thread selects a subset of the incoming frames to build a 3D point map of the surroundings. These selected frames are called keyframes and their selection is based on certain pre-determined heuristics. After that a batch optimization is applied on the joint state of map points and keyframe poses. There are several key differences compared to the EKF based visual SLAM approach, namely:

1. An EKF based state estimation is not used
2. Uncertainties are not modelled, which saves computational load
3. Order of number of features used is much higher
4. Accuracy is maintained by local and global batch optimisation

PTAM has been used by a few other researchers for aerial applications. For example, Ghadiok et al. [30] used PTAM for a quadrotor platform to perform autonomous gripping of an object kept on the ground. Blosch et al. [19] used PTAM on a quadrotor and demonstrated autonomous trajectory following.

In my test scenario, due to time constraints, there was only sufficient time to demonstrate PTAM on the quadrotor. Thus, it was flown manually within the lab environment and it mapped the features in its surrounding without using prior information. As opposed to the EKF based visual SLAM, PTAM does not face difficulty in tracking features even with rapid motion and vibrations. This is mainly because it does not use a filter based approach. Fig. 7.5 shows the 3D position as estimated by the SLAM algorithm. Fig. 7.6 shows the on-board camera image while running PTAM. Fig. 7.7 shows the 3D features as estimated by the algorithm along with the camera pose at when keyframes are recorded.

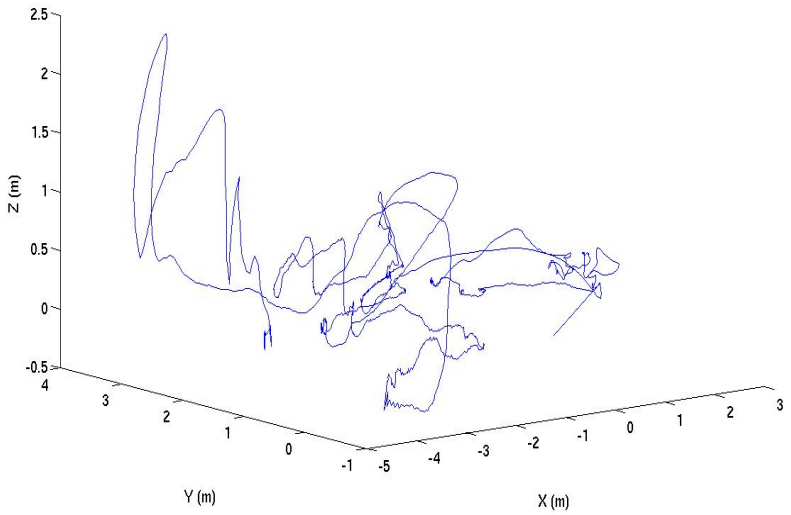


Figure 7.5: 3D Position of quadrotor

Figure 7.6: Onboard camera image: point features

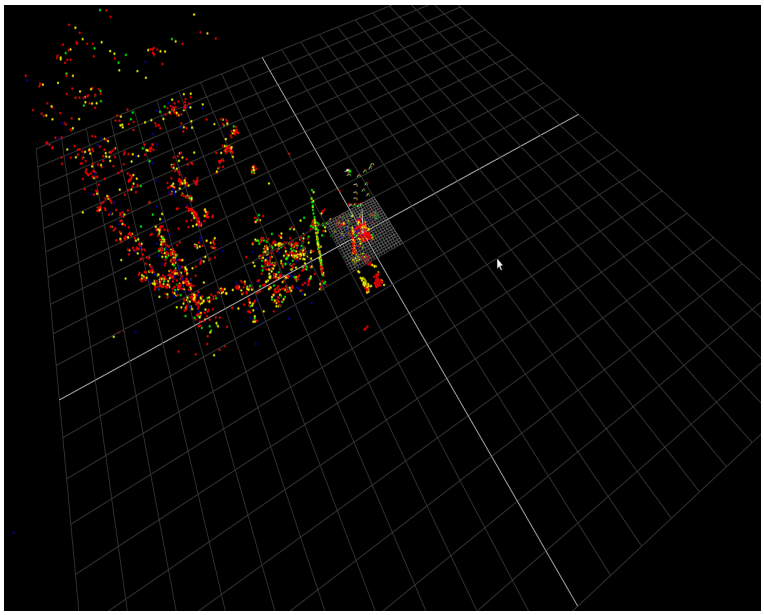


Figure 7.7: 3D features

Figure 7.8: Quadrotor in flight

CHAPTER 8

Discussion & Future Work

8.1 Discussion

Initial research into vision based navigation and localisation yielded a wealth of information about the kind of methods and algorithms that would be best suited to aerial SLAM. Firstly, a simple algorithm for visual navigation based on optical flow was developed and tested on a Parrot A.R Drone. It works well for a corridor like environment and is capable of avoiding large obstacles. It was also tested in an unstructured lab environment where it was seen to face difficulty when faced with large planar obstacles. Visual odometry was also experimented with, during the initial stage of the project. A Python code was developed based on the 5 Point Algorithm [37] to determine the trajectory of a monocular camera. It was found to be able to run at 4 Hz which was not suitable for real time performance for a highly dynamic platform. Also, there was no method to determine the scale of the camera motion. Thus the focus was shifted to Monocular SLAM [28]. The simulation and real-time test results validated the accuracy of this algorithm. It is able to run at a processing rate of 25 Hz and can perform loop closure to correct for drift.

Once the visual SLAM algorithm was implemented, the design and development of the quadrotor followed. The platform development for this project was carried out keeping in mind the unique requirements from the vehicle. A quadrotor was found to be the most suitable flying platform, with its large payload carrying capacity and ability to hover. The complete development cycle of the quadrotor was carried out with modest resources and successfully achieved performance comparable to commercial systems at less than half the cost. The attitude control system utilises a state-of-the-art IMU running an on-board Kalman filter and noise rejection algorithms. It was integrated with a low cost open source embedded Arduino platform running a PID based stability augmentation system. A reliable serial communication protocol was developed for sharing critical data between the on-board computer and Arduino board. It allowed motion commands to be sent from the computer to the Arduino board and IMU readings from the Arduino to the computer.

After the quadrotor and visual SLAM algorithm were completed. They were integrated and tested on the quadrotor. A careful analysis revealed that EKF based visual SLAM faced certain difficulties in coping with the rapid dynamics and vibrations on the aerial platform. The vibrations caused the SLAM filter to drift which lead me to test an alternative method on the quadrotor i.e. Parallel Tracking and Mapping System (PTAM) developed by Klein et al. [33]. PTAM was able localise and map a complex lab environment while the camera was mounted on the quadrotor. This was mainly because it did not maintain a filtered estimate of the vehicle state, instead using global and batch optimisation and storing only keyframes for the SLAM map.

A reactive navigation system relying on depth measurements from the LIDAR was tested in simulation. It is able to safely navigate the vehicle to its target location without a priori map information while avoiding dynamic obstacles in real time. This software is highly modular and can be adapted for future work on ground robots.

Overall, a unique flying platform equipped with multiple state-of-the-art sensors was developed. It is capable of sustained stable flight. Numerous software solutions ranging from communications to visual SLAM were developed and tested. The key feature of this platform is that it is capable of performing all computations on-board.

8.2 Future Work

The system developed in this project, though capable of quite a few tasks is not yet fully matured. They key future work required to carry this work forward would be:

1. Modifying the EKF filter for visual SLAM: By varying the visual measurement update rate to reach an optimal value such that vibrations can be rejected and at the same time the camera motion can be tracked successively between frames. Further tuning of modelling parameters (process noise) to obtain most suitable values for operation in flight.
2. To increase the consistency of the map over large number of features, an intelligent map management system can be introduced such that the computational complexity does not exceed beyond the limitations of the computer with increasing landmarks.
3. Developing an external positioning system to validate the on-board SLAM with the ground truth data. This could either be a vision based tracking system or a more commonly used motion capture system available commercially.
4. Integrating the LIDAR based reactive navigation system with visual SLAM to enable autonomous flight.

CHAPTER A

Appendix A

Aerodynamic Modeling

The aerodynamics calculations elaborate the behaviour of the propellers in air. The two most significant quantities of importance to us are the Thrust and Drag. The analysis can be done according to the combined momentum and blade element theory which merges the concepts of two analysis [15].

A.1 Momentum Theory

The rotor can be modeled as a disk which imparts energy into the air, receiving a counter reaction from it. The hypothesis of this theory are:

- A flux of air crossing the disk does not interact with external air
- The propeller has infinite number of blades
- The thickness of the disk is negligible
- The vertical speed of air is continuous through the disk
- The air is considered a perfect gas and incompressible

In this description, T_{MT} is the thrust of the propeller. The air speeds are defined with respect to the rotor disk; $v_{-\infty}[m \cdot s^{-1}]$ is asymptotically over, $v_1[m \cdot s^{-1}]$ is directly over, $v_2[m \cdot s^{-1}]$ is directly under $v_{+\infty}[m \cdot s^{-1}]$ is asymptoticall under. The air pressures are defined relative to the rotor as; $p_{-\infty}[Pa]$, $p_1[Pa]$, $p_2[Pa]$ and $p_{+\infty}[Pa]$.

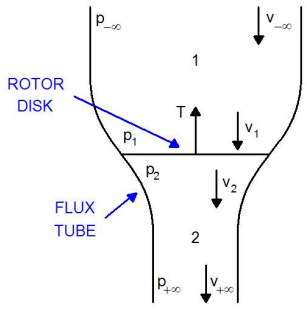


Figure A.1: Momentum Theory [22]

Fig. A.1 shows the momentum theory model graphically. The thrust generated by the rotation of the propeller is proportional to the pressure difference through the disk. It can also be defined as the change in momentum of the air passing through the disk.

$$\begin{cases} T_{MT} = A(p_1 - p_2) \\ T_{MT} = \dot{m}_A(v_{-\infty} - v_{+\infty}) = \rho_A A v_1 (v_{-\infty} - v_{+\infty}) \end{cases} \tag{A.1.1}$$

Where $A[m^2]$ is the area of the rotor disk, $\dot{m}_A$ is the mass flow rate through the disk and $\rho_A[kg \cdot m^{-3}]$ is the air density. According to the proposed hypothesis, the air speed directly over the rotor v_1 is equal to that one directly under the rotor v_2. The Bernoulli equation between sections $-\infty$ and 1 and between 1 and 2 are:

$$p_{-\infty} + \frac{1}{2}\rho_A v_{-\infty}^2 = p_1 + \frac{1}{2}\rho_A v_1^2 \tag{A.1.2}$$

$$p_2 + \frac{1}{2}\rho_A v_2^2 = p_{+\infty} + \frac{1}{2}\rho_A v_{-\infty}^2 \tag{A.1.3}$$

Rearranging equations A.1.2 and A.1.3, and since $p_{+\infty} = p_{-\infty}$, v_1 can be written as:

$$v_1 = \frac{v_{-\infty} + v_{-\infty}}{2} \tag{A.1.4}$$

The inflow speed $v_I[m \cdot s^{-1}]$ is

$$v_I = v_1 - v_{-\infty} = \frac{v_{+\infty} - v_{-\infty}}{2} \tag{A.1.5}$$

Therefore, one can write the thrust as,

$$T_{MT} = 2\rho_A A v_1 v_I \tag{A.1.6}$$

In hover condition (our usual operating point), $v_{-\infty} = 0$, thus $v_1 = v_I$. The thrust generated by one propeller is equal to the weight carried by it W_P, i.e. a quarter of the vehicle weight. Therefore, one can say,

$$W_P = 2\rho_A A v_I^2 \tag{A.1.7}$$

The inflow ratio λ is used to relate the inflow speed to the rotor tip speed,

$$\lambda = \frac{v_i}{\omega_H R_P} \tag{A.1.8}$$

Where, ω_H is the angular speed of the blade in hovering condition and R_P is its radius.

A.2 Blade Element Theory

The Momentum Theory provides very little details about the performance of the propellers. Thus, to ahieve a higher degree of accuracy in the modeling we use the Blade Element Theory with some contributions from the Momentum Theory. Fig. A.2 shows the model of the propeller seciton.

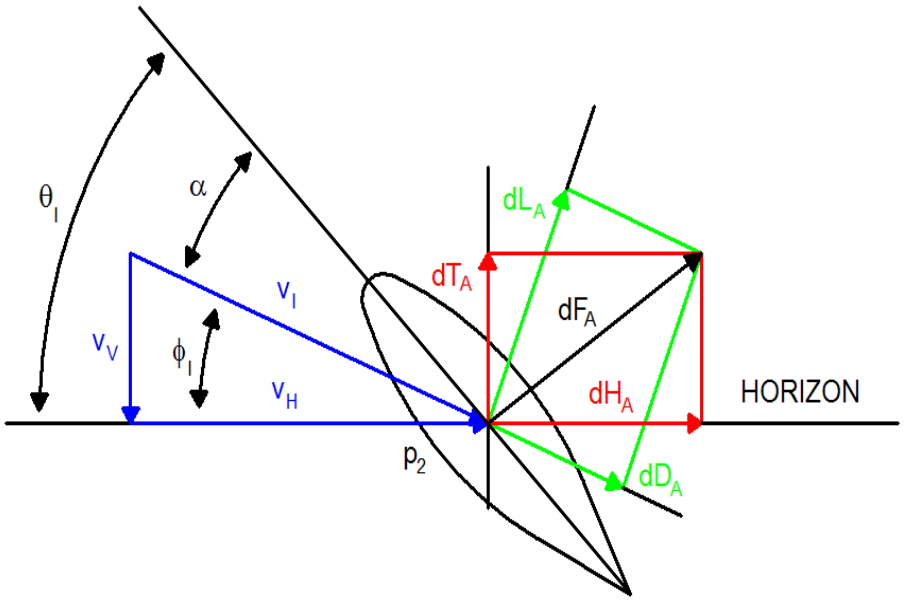


Figure A.2: Blade Element [22]

The horizon line is perpendicular to the rotor shaft (hovering condition). $\theta_I[rad]$ is the angle of incidence between the horizon line and the blade chord line. $\alpha[rad]$ is the angle of attack between the blade chord line and the local air flow velocity vector $v_T[m \cdot s^1]$. v_T is the vector sum of the horizontal $v_H[m \cdot s^{-1}]$ and vertical $v_V[m \cdot s^1]$ air flow velocity. The angle between the horizon and the local velocity vector is the local inflow angle $\phi_I[rad]$. $dD_{BET}[N \cdot m^1]$ is the infinitesimal drag force while $dL_{BET}[N \cdot m^1]$ is the infinitesimal lift force. The vector sum of dD_{BET} and dL_{BET} is the infinitesimal aerodynamic resultant force $dF_{BET}[Nm^{-1}]$. dF_{BET} can be also be divided into two infinitesimal aerodynamic vertical $dT_{BET}[N \cdot m^1]$ and horizontal $dH_{BET}[N \cdot m^1]$ components.

The v_V velocity due to the inflow motion, thus, it is uniform for every section. The v_H velocity vector is due to the angular speed of the blade element. Since the velocity of each blade element depends on its radial position from the rotation shaft it is a function of r.

$$v_V = v_I = \omega_P R_P \lambda \tag{A.2.1}$$

$$v_H = \omega_P r = \omega_P R_P \frac{r}{R_P} \tag{A.2.2}$$

From basic aerosdynamics theory [15], we know that the lift and drag generated by the blade section is determined by,

$$dL_{BET} = \frac{1}{2} \rho_A v_H^2 C_L c dr \tag{A.2.3}$$

$$dD_{BET} = \frac{1}{2}\rho_A v_H^2 C_D c dr \tag{A.2.4}$$

Where, C_L and C_D are the lift and drag coefficient and c is mean chord of the propeller blade.

$$C_L = a\alpha = a(\theta_I - \phi_I) \tag{A.2.5}$$

a is the lift slope and for thin aerofoils it is assumed to be 2π [15] and α is the angle of attack.

The blade twist is assumed to vary linearly with radial position. Thus the model includes the two constants zero angle of incidence ${}_{I0}[rad]$ and twist angle of incidence ${}_{Itw}[rad]$. Hence,

$$\theta_I = \theta_{I0} = \theta_{Itw}\frac{r}{R_P} \tag{A.2.6}$$

The angular velocity is significantly higher than the total inflow through the blade. Hence, small angle approximations can be made to define the inflow angle ϕ_I as

$$\phi_I = \frac{v_V}{v_H} \tag{A.2.7}$$

Combining the above information, the infinitesimal lift force can be rewritten as,

$$dL_{BET} = \frac{1}{2}\rho_A v_H^2 a(\theta_{I0} - \theta_{Itw}\frac{r}{R_P} - \frac{v_V}{v_H})cdr \tag{A.2.8}$$

Since the drag is order of magnitude smaller than the lift, the thrust dT_{BET} is assumed equal to the lift.

$$T_{BET} = N_B \int_0^{R_P} dT_{BET} = N_B \rho_A a c \omega_p^2 R_p^3 (\frac{\theta_{I_0}}{6} - \frac{\theta_{tw}}{8} - \frac{\lambda}{4}) \tag{A.2.9}$$

To calculate the horizontal component dH_{BET} can be simplified considering the small angle approximations. However, the components of the lift and drag are both of similar magintude therefore both need to be included.

$$dH_{BET} = dD_{BET}\cos\phi_I + dL_{BET}\sin\phi_I \approx dD_{BET} + dL_{BET}\frac{v_V}{v_H} \tag{A.2.10}$$

The torque Q_{BET} generated by the propeller is calculating by multiplying the infinitesimal forces by the arm and integrating over the entire propeller.

$$Q_{BET} = N_B \int_0^{R_P} (dD_{BET} + dL_{BET}\frac{v_V}{v_H}) = N_B \rho_A a c \omega_p^2 R_p^4 (\frac{C_D}{8} + a\lambda(\frac{\theta_{I_0}}{6} - \frac{\theta_{tw}}{8} - \frac{\lambda}{4})) \tag{A.2.11}$$

CHAPTER **B**

Appendix B

Modeling of Quadrotor Dynamics[22]

B.1 Assumptions

The model used for the development of the quadrotor assumes the following:

1. The structure is rigid
2. The structure is symmetrical
3. There is no blade flapping in the propellers
4. Thrust and drag are proportional to the square of the propellers speed

The two most common ways of describing the dynamics of a vehicle are using Newton-Euler formalism and Euler-Lagrange formalism. Newton-Euler is the preffered method, however, the Euler-Lagrange method is also described in some detail below to form a better understanding of the underlying fundamentals.

B.2 Modelling using Euler-Lagrange Formalism

The rotation dynamics of the test-bench are modelled in this section using Euler-Lagrange Formalism. Let us consider earth fixed frame E and body fixed frame B. The airframe orientation in space is given by a rotation R from B to E, where R is the rotation matrix.

$$L = T - V \tag{B.2.1}$$

General form of motion in the Lagrange Method.

$$\Gamma = \frac{d}{dt}(\frac{\delta L}{\delta \dot{q_i}}) - \frac{\delta L}{\delta q_i} \tag{B.2.2}$$

Where,
q_i: Generalised Coordinates
Γ: Generalised Forces
T: Kinetic Energy

V: Potential Energy

The earth fixed frame E is $[\vec{X}, \vec{Y}, \vec{Z}]$ and body fixed frame B is $[\vec{x}, \vec{y}, \vec{z}]$. If a point on the body B undergoes three successive rotations, it can be expressed by:

$$r_{X,Y,Z} = R(\phi, \theta, \psi) \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{B.2.3}$$

Then,

$$\begin{cases} r_X = cos(\psi)cos(\theta)x + (cos(\psi)sin(\phi) - sin(\psi)cos(\phi))y + (cos(\psi)sin(\theta)cos(\phi) + sin(\psi)sin(\phi))z \\ r_Y = sin(\psi)cos(\theta)x + (sin(\psi)sin(\theta)sin(\phi) + cos(\psi)cos(\phi))y + (sin(\psi)sin(\theta)cos(\phi) - cos(\psi)sin(\phi))z \\ r_Z = -sin(\theta)x + cos(\theta)sin(\phi)y + cos(\theta)cos(\phi)z \end{cases} \tag{B.2.4}$$

The corresponding velocities are obtained by differentiation of the above equation and the magnitude of the velocity for any point is,

$$v^2 = v_x^2 + v_y^2 + v_z^2 \tag{B.2.5}$$

From the equation and assuming the inertia matrix is diagonal (the structure was assumed to be syymetrical). The kinetic energy is,

$$T = \frac{1}{2}I_{xx}(\dot{\phi} - \dot{\psi}sin(\theta))^2 + \frac{1}{2}I_{yy}(\dot{\theta}cos(\phi) + \dot{\psi}sin(\phi)cos(\theta))^2 + \frac{1}{2}I_{zz}(\dot{\theta}sin(\phi) - \dot{\psi}cos(\phi)) \tag{B.2.6}$$

The potential energy is,

$$V = g\int(-sin(\theta)\cdot x + sin(\phi)cos(\theta)\cdot y + cos(\phi)cos(\theta)\cdot z)dm(r) \tag{B.2.7}$$

Using the potential energy formula, eqn. B.2.7 can be expressed in earth frame E as,

$$V = \int xdm(x)(-gsin(\theta)) + \int ydm(y)(gsin(\phi)cos(\theta)) + \int zdm(z)(gcos(\phi)cos(\theta)) \tag{B.2.8}$$

Using the Lagrangian and the derived formula for equations of motion (eqn. B.2.2), the 3 equations of motion are:

$$\begin{cases} I_{xx}\ddot{\phi} = \dot{\theta}\dot{\psi}(I_{yy} - I_{zz}) \\ I_{yy}\ddot{\theta} = \dot{\phi}\dot{\psi}(I_{zz} - I_{xx}) \\ I_{zz}\ddot{\psi} = \dot{\theta}\dot{\phi}(I_{xx} - I_{yy}) \end{cases} \tag{B.2.9}$$

Torques generated due to thrust difference of each pair of motors,

$$\begin{cases} \tau_x = bl(\Omega_4^2 - \Omega_2^2) \\ \tau_y = bl(\Omega_3^2 - \Omega_1^2) \\ \tau_z = d(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \end{cases} \tag{B.2.10}$$

Gyroscopic effect from propeller rotation,

$$\left\{\tau_x^{'} = J_r\omega_y(\Omega_1 + \Omega_3 - \Omega_2 - \Omega_4)\tau_y^{'} = J_r\omega_x(\Omega_2 + \Omega_4 - \Omega_1 - \Omega_3)\right. \tag{B.2.11}$$

The quadrotor dynamic model describing the roll, pitch and yaw rotations contains then, three terms which are the gyroscopic effect resulting from the rigid body rotation, the gyroscopic effect resulting from the propeller rotation coupled with the body rotation and finally the actuators action:

$$\begin{cases} I_{xx}\ddot{\phi} = \dot{\theta}\dot{\psi}(I_{yy} - I_{zz}) - J\dot{\theta}\Omega_r + \tau_x \\ I_{yy}\ddot{\theta} = \dot{\phi}\dot{\psi}(I_{zz} - I_{xx}) + J\dot{\phi}\Omega_r + \tau_y \\ I_{zz}\ddot{\psi} = \dot{\theta}\dot{\phi}(I_{xx} - I_{yy}) + \tau_z \end{cases} \tag{B.2.12}$$

The DC motors have well known equations,

$$\begin{cases} L\frac{di}{dt} = u - R_{mot}i - k_e\omega_m \\ J_m\frac{d\omega_m}{dt} = \tau_m - \tau_d \end{cases} \tag{B.2.13}$$

As a small motor with very low inductance is used, the second order DC motor dynamics may be approximated by:

$$J_m\frac{d\omega_m}{dt} = -\frac{k_m^2}{R_{mot}}\omega_m - \tau_d + \frac{k_m}{R_{mot}}u \tag{B.2.14}$$

By introducing the propeller model, the equation can be re-written as :

$$\begin{cases} \dot{\omega_m} = -\frac{1}{\tau}\omega_m - \frac{d}{r^3J_t}\omega_m^2 + \frac{1}{k_m}u \\ \frac{1}{\tau} = \frac{k_m^2}{RJ_t} \end{cases} \tag{B.2.15}$$

The equation above can be linearised around a point $\dot{w_0}$ to,

$$\dot{w_m} = -Aw_m + Bu + C \tag{B.2.16}$$

Where,

$$A = \frac{1}{\tau} + \frac{2dw_0}{\eta r^3 J_t}, B = \frac{1}{\tau}, C = \frac{dw_0}{\eta r^3 J_t} \tag{B.2.17}$$

B.3 Newton-Euler Formalism

The equations of motion are generally written in the body frame 'B'. The origin of the body fixed frame is assumed to be located at the center of mass of the vehicle. Also, the axes of the frame 'B' coincide with the body principal axis of inertia. This makes the inertia matrix I diagonal which allows us to take advantage of the symmetry of the design and simplifies calculations.

The forces acting in the body frame are written as,

$$m(\dot{V}^B + \omega^B \times V^B) = F^B \tag{B.3.1}$$

Where m is the mass of the vehicle, F^B is the force vector in body frame, V^B is the velocity vector in body frame and ω^B is the angular velocity in body frame. Eqn. B.3.2 describes the rotational dynamics of the quadrotor. $I_{3\times3}$ is the identity matrix, I is the inertia matrix, ω^B is the vector of angular rates in the body frame and τ^B is the vector of torque acting on the vehicle in the body frame.

$$I\dot{\omega}^B + \omega^B \times (I\omega^B) = \tau^B \tag{B.3.2}$$

Putting the above equations in a matrix formulation yields,

$$\begin{bmatrix} mI_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & I \end{bmatrix} \begin{bmatrix} \dot{V}^B \\ \dot{\omega}^B \end{bmatrix} + \begin{bmatrix} \omega^B \times (mV^B) \\ \omega^B \times (I\omega^B \end{bmatrix} = \begin{bmatrix} F^B \\ \tau^B \end{bmatrix} \tag{B.3.3}$$

A generealized force vector Λ can be defined according to the Eqn. B.3.4.

$$\Lambda = \begin{bmatrix} F^B & \tau^B \end{bmatrix} = \begin{bmatrix} F_x & F_y & F_z & \tau_x & \tau_y & \tau_z \end{bmatrix}^T \tag{B.3.4}$$

Therefore, one can write the dynamics equations in the following form,

$$M_B\dot{\nu} + C_B(\nu)\nu = \Lambda \tag{B.3.5}$$

Where $\dot{\nu}$ is the generalized acceleration vector in the body frame 'B'. M_B is the system inertial matrix and $C_B(\nu)$ is the Corioli Centripetal matrix. M_B is a diagonal matrix as shown below,

$$M_B = \begin{bmatrix} mI_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & I \end{bmatrix} = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{xx} & 0 & 0 \\ 0 & 0 & 0 & 0 & I_{yy} & 0 \\ 0 & 0 & 0 & 0 & 0 & I_{zz} \end{bmatrix} \tag{B.3.6}$$

The Coriolis-Centripetal matrix is,

$$\begin{bmatrix} 0_{3\times3} & -mS(V^B) \\ 0_{3\times3} & -S(I\omega^B) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & m\omega & -mv \\ 0 & 0 & 0 & m\omega & 0 & mu \\ 0 & 0 & 0 & mv & -mu & 0 \\ 0 & 0 & 0 & 0 & I_{zz}r & -I_{yy}q \\ 0 & 0 & 0 & -I_{zz}r & 0 & I_{xx}p \\ 0 & 0 & 0 & I_{yy}q & -I_{xx}p & 0 \end{bmatrix} \tag{B.3.7}$$

In the above equation $S(\cdot)$ is the skew-symmetric operator. For a general vector $k = [k_1\ k_2\ k_3]^T$, the skew symmetric matrix of k is defined as,

$$S(k) = -S^T(k) = \begin{bmatrix} 0 & -k_3 & k_2 \\ k_3 & 0 & -k_1 \\ k_2 & k_1 & 0 \end{bmatrix} \tag{B.3.8}$$

The force vector can be split into 3 major contributors i.e. Gravity, Gyroscopic Effects and Actuator input. These are described in the following sub-sections.

1. **Gravity**
   The contribution of the gravitational force on the body does not produce any torque as it acts through the center of mass. $G_B(\xi)$ is the vector of gravitational forces on the quadrotor in the body frame 'B' and R^{BE} is the Euler rotation matrix (Body to Earth frame).

$$G_B(\xi) = \begin{bmatrix} F_G^B \\ 0_{3\times1} \end{bmatrix} = \begin{bmatrix} R^{BE^{-1}} F_G^E \\ 0_{3\times1} \end{bmatrix} = \begin{bmatrix} mg\sin(\theta) \\ -mg\sin(\phi)\cos(\theta) \\ -mg\cos(\phi)\cos(\theta) \end{bmatrix} \tag{B.3.9}$$

2. **Gyroscopic Effects**

   The gyroscopic effects produced by the rotation of the propellers is according to Eqn. B.3.10.

$$O_B(\nu)\Omega = \begin{bmatrix} 0_{3\times1} \\ -\sum_1^4 J_{TP} \left(\omega^B \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} (-1)^k \Omega_k \right) \end{bmatrix} = \begin{bmatrix} 0_{3\times1} \\ J_{TP} \begin{bmatrix} -q \\ p \\ 0 \end{bmatrix} \Omega \end{bmatrix} = J_{TP} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ q & -q & q & q \\ -p & p & -p & p \\ 0 & 0 & 0 & 0 \end{bmatrix} \vec{\Omega} \tag{B.3.10}$$

   $O_B(\nu)$ is the gyroscopic matrix and J_{TP} is the total rotational moment of inertia of the rotor (propeller + motor shaft) about the the propeller axis.

$$\Omega = -\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4 \tag{B.3.11}$$

$$\vec{\Omega} = \begin{bmatrix} \Omega_1 \\ \Omega_2 \\ \Omega_3 \\ \Omega_4 \end{bmatrix} \tag{B.3.12}$$

B.3.1 Actuators

3. **Actuators**
   The final contribution is produced from the main actuator input i.e. thrust and drag on acting on the propellers. If these contributions due to the control inputs are called U_B then we have,

$$U_B(\Omega) = E_B\omega^2 \begin{bmatrix} 0 \\ 0 \\ b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ bl(\Omega_4^2 - \Omega_2^2) \\ bl(\Omega_3^2 - \Omega_1^2) \\ d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \end{bmatrix} \tag{B.3.13}$$

Where l is the moment arm of the propeller axis of rotation about the center of gravity of the quadrotor and U_1, U_2, U_3 and U_4 are the movement commands. It is possible to write the movement command vector $U_B(\Omega)$ as the product of a constant matrix E_B and Ω^2.

$$E_B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ b & b & b & b \\ 0 & -bl & 0 & bl \\ bl & 0 & -bl & 0 \\ -d & d & -d & d \end{bmatrix} \tag{B.3.14}$$

Taking Eqn. B.3.5 and formulating the Force vector Λ as a sum of $G_B(\xi)$ the gravity vector, $O_B(\nu)$ the gyroscopic moments vector, and $U_B(\Omega)$ the actuator inputs. The dynamics of vehicle can be expressed as,

$$M_B\dot{\nu} + C_B(\nu)\nu = \Lambda = G_B(\xi) + O_B(\nu) + E_B\Omega^2 \tag{B.3.15}$$

The dynamics can be expressed in a system of equations in the body frame B as,

$$\begin{cases} \dot{u} = (vr - wq) + g\sin(\theta) \\ \dot{v} = (wp - ur) - g\cos(\theta)\sin(\phi) \\ \dot{w} = (uq - vp) - g\cos(\theta)\cos(\phi) + \frac{U_1}{m} \\ \dot{p} = \frac{I_{yy} - I_{zz}}{I_{xx}}qr - \frac{J_{TP}}{I_{xx}}q\Omega + \frac{U_2}{I_{xx}} \\ \dot{q} = \frac{I_{zz} - I_{xx}}{I_{yy}}pr - \frac{J_{TP}}{I_{yy}}p\Omega + \frac{U_3}{I_{yy}} \\ \dot{r} = \frac{I_{xx} - I_{yy}}{I_{zz}}pq + \frac{U_4}{I_{zz}} \end{cases} \tag{B.3.16}$$

Where,

$$\begin{cases} U_1 = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ U_2 = bl(-\Omega_2^2 + \Omega_4^2) \\ U_3 = bl(\Omega_1^2 - \Omega_3^2) \\ U_4 = d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \end{cases} \tag{B.3.17}$$

The quadrotor dynamic system described above is in the Body reference frame. However, it can be useful to depict the angular rates w.r.t Body Frame and linear equations w.r.t Earth Frame. In this hybrid system, the equations of motion are transformed to:

$$\begin{cases} \ddot{X} = (sin(\psi)sin(\phi) + cos(\psi)sin(\theta)cos(\phi))\frac{U_1}{m} \\ \ddot{Y} = (-cos(\psi)sin(\phi) + sin(\psi)sin(\theta)cos(\phi)) \\ \ddot{Z} = -g + cos(\theta)cos(\phi)\frac{U_1}{m} \\ \dot{p} = \frac{I_{yy}-I_{zz}}{I_{xx}}qr - \frac{J_{TP}}{I_{xx}}q\Omega + \frac{U_2}{I_{xx}} \\ \dot{q} = \frac{I_{zz}-I_{xx}}{I_{yy}}pr - \frac{J_{TP}}{I_{yy}}p\Omega + \frac{U_3}{I_{yy}} \\ \dot{r} = \frac{I_{xx}-I_{yy}}{I_{zz}}pq + \frac{U_4}{I_{zz}} \end{cases} \tag{B.3.18}$$

Where X, Y, Z are the position coordinates in the Earth Frame.

Bibliography

[1] Aeroquad- the opensource quadcopter, www.aeroquad.com.

[2] Apc propellers, www.apcprop.com.

[3] A.r drone autopylot, http://home.wlu.edu/levys.

[4] A.r drone software development kit, projects.ardrone.org.

[5] Ascending technologies gmbh, www.asctec.de.

[6] Calibrated 5 point solver, http://vis.uky.edu/ stewe/fivepoint/.

[7] Diy drones, diydrones.com.

[8] Hokuyo automatic co., ltd., www.hokuyo-aut.jp.

[9] Mobile robot programming toolkit, www.mrpt.org.

[10] Neato robotics, www.neatorobotics.com.

[11] Sbg systems, www.sbg-systems.com.

[12] Unibrain ieee1394 camera modules, www.unibrain.com.

[13] E. Altug and C. Taylor. Vision-based pose estimation and control of a model helicopter. In *Proceedings of the IEEE International Conference on Mechatronics*, pages 316–321, 2004.

[14] A. Angeli, S. Doncieux, and D. Filliat. Real-time visual loop-closure detection. In *IEEE International Conference on Robotics and Automation*, May 2008.

[15] M. Arra. *L'elicottero*. Hoepli, 2001.

[16] A. Bachrach, S. Prentice, R. He, and N. Roy. Range- robust autonomous navigation in gps-denied environments. *Journal of Field Robotics*, 2011.

[17] A. Beyeler, J. Zufferey, and D. Floreano. 3d vision-based navigation for indoor microflyers. In *IEEE International Conference on Robotics and Automation*, 2007.

[18] J.L Blanco, J. Gonzalez, and J.A Madrigal. Extending obstacle avoidance methods through multiple parameter-space transformations. *Autonomous Robots*, 24, 2008.

[19] M. Blosch, S. Weiss, D. Scaramuzza, and R. Siegwart. Vision based mav navigation in unknown and unstructured environments. In *IEEE International Conference on Robotics and Automation*, 2010.

[20] S. Bouabdallah. *Design and Control of Quadrotors With Application to Autonomous flying*. PhD thesis, Ecole Polytechnique Federale De Lausanne, 2007.

[21] J.Y Bouguet. Pyramidal implementation of the lucas kanade feature tracker description of the algorithm. Technical report, Intel Corporation, 2000.

[22] T. Bresciani. Modelling, identification and control of a quadrotor helicopter. Master's thesis, Lund University, October 2008.

[23] K. Celik, S.J. Chung, M.Clausman, and A. Somani. Monocular vision slam for indoor aerial vehicles. In *IEEE/RSJ International Conference on Intelligent Systems and Robotics*, St. Louis, USA, October 2009.

[24] J. Civera, A.J. Davison, and J.M.M Montiel. Inverse depth parametrisation for monocular slam. October 2008.

[25] J. Civera, O.G. Grasa, A.J. Davison, and J.M.M Montiel. 1-point ransac for ekf filtering: Application to real-time structure from motion and visual odometry. *Journal of Field Robotics*, October 2010.

[26] L. Clemente, A.J. Davison, I.D. Reid, J. Neira, and J.D Tardos. Mapping large loops with a single hand-held camera. In *Robotics: Science and Systems Conference*, June 2007.

[27] A. Coates, P. Abbeel, and A.Y Ng. Learning for control from multiple demonstrations. In *Proceedings of the 25th International Conference on Machine Learning*, pages 144–151. ACM, 2008.

[28] A.J. Davison, I.D. Reid, N.D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, 2007.

[29] M.A. Garratt and A. Cheung. Obstacle avoidance in cluttered environments using optic flow. In *Australian Conference on Robotics and Automation*, December 2009.

[30] V. Ghadiok, J Goldin, and W. Ren. Autonomous indoor aerial gripping using a quadrotor. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2011.

[31] G.M Hoffman, H. Huang, S.L. Waslander, and C.J. Tomlin. Quadrotor helicopter flight dynamics and control. In *Proceedings of Guidance, Navigation and Control*, 2007.

[32] J.P. How, B. Bethke, A. Frank, D. Dale, and J. Vian. Real-time indoor autonomous vehicle test environment. In *Control Systems Magazine, IEEE*, volume 28, pages 51–64, 2008.

[33] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *International Symposium on Mixed and Augmented Reality*, 2007.

[34] E. Mikhail, J. Bethel, and J.C McGlone. *Introduction to Modern Pho- togrammetry.* Wiley, 2001.

[35] J. Minguez and L. Montano. Nearness diagram (nd) navigation: Collision avoidance in troublesome scenarios. *IEEE Transactions on Robotics and Automation*, 20(1):43–59, 2004.

[36] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fastslam: a factored solution to the simultaneous localization and mapping problem. In *Proc. of the AAAI National Conference on Artificial Intelligence*, pages 593–598, 2002.

[37] D. Nister. An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004.

[38] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry for ground vehicle applications. *Journal of Field Robotics*, 23(1), 2006.

[39] J. Shi and C. Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition*, June 1994.

[40] B. Sinopoli, M. Micheli, G. Donato, and T.J. Koo. Vision based navigation for an unmanned aerial vehicle. In *IEEE International Conference on Robotics and Automation*, 2001.

[41] N. Sunderhauf, S. Lange, and P.Protzel. Using the unscented kalman filter in mono-slam with inverse depth parametrization for autonomous airship control. In *IEEE International Workshop on Safety Security and Rescue Robotics*, 2007.

[42] T. Templeton, D.H Shim, C. Geyer, and S.S Sastry. Autonomous vision-based landing and terrain mapping using an mpc-controlled unmanned rotorcraft. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1349–1356, April 2007.

[43] S. Thrun. Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23(9), 2006.

[44] M. Tomono. Monocular slam using a rao-blackwellised particle filter with exhaustive pose space search. In *IEEE International Conference on Robotics and Automation*, April 2007.

[45] G.P. Tournier, M. Valenti, and J.P. How. Estimation and control of a quadrotor vehicle using monocular vision and moire patterns. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Keystone, Colorado, August 2006.

[46] S. Zingg, D. Scaramuzza, S. Weiss, and R. Siegwart. Mav navigation through indoor corridors using optical flow. In *International Conference on Robotics and Automation*, 2010.