Summer 7-12-2018

High-Performance Testbed for Vision-Aided Autonomous Navigation for Quadrotor UAVs in Cluttered Environments

Shakeeb Ahmad
University of New Mexico

Recommended Citation

Shakeeb Ahmad

Candidate

Electrical & Computer Engineering

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Dr. Rafael Fierro, Chair

Dr. Francesco Sorrentino, Member

Dr. Yin Yang, Member

High-Performance Testbed for Vision-Aided Autonomous Navigation for Quadrotor UAVs in Cluttered Environments

by

Shakeeb Ahmad

B.E., National University of Sciences and Technology, Pakistan 2015

THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Electrical Engineering

The University of New Mexico

Albuquerque, New Mexico

July, 2018

Dedication

To my parents and siblings for their support, love, and encouragement throughout my life.

Acknowledgments

High-Performance Testbed for Vision-Aided Autonomous Navigation for Quadrotor UAVs in Cluttered Environments

by

Shakeeb Ahmad

B.E., National University of Sciences and Technology, Pakistan 2015

M.S., Electrical Engineering, University of New Mexico, 2018

Abstract

This thesis presents the development of an aerial robotic testbed based on Robot Operating System (ROS). The purpose of this high-performance testbed is to develop a system capable of performing robust navigation tasks using vision tools such as a stereo camera. While ensuring the computation of robot odometry, the system is also capable of sensing the environment using the same stereo camera. Hence, all the navigation tasks are performed using a stereo camera and an inertial measurement unit (IMU) as the main sensor suite. ROS is used as a framework for software integration due to its capabilities to provide efficient communication and sensor interfaces. Moreover, it also allows us to use C++ which is efficient in performance especially on embedded platforms. Combining together ROS and C++ provides the necessary computation efficiency and tools to handle fast, real-time image processing and planning which are the vital parts of navigation and obstacle avoidance on such scale.

The main application of this work revolves around proposing a real-time and efficient way to demonstrate vision-based navigation in UAVs. The proposed approach is developed for a quadrotor UAV which is capable of performing defensive maneuvers in case any obstacles are in its way, while constantly moving towards a user-defined final destination. Stereo depth computation adds a third axis to a two dimensional image coordinate frame. This can be referred to as the depth image space or depth image coordinate frame. The idea of planning in this frame of reference is utilized along with certain precomputed action primitives. The formulation of these action primitives leads to a hybrid control law for feasible trajectory generation. Further, a proof of stability of this system is also presented. The proposed approach keeps in view the fact that while performing fast maneuvers and obstacle avoidance simultaneously, many of the standard optimization approaches do not work in real-time on-board due to time and resource limitations.

Moreover, to verify the performance of the testbed, we developed an application inspired by the autonomous drone racing competition held at the IEEE International Conference on Intelligent Robots and Systems 2017. The goal was to design a navigation strategy for a quadrotor UAV to pass through square targets. The targets' positions were only approximately known so that we were unable to fully rely on any predefined map. In addition, due to drift and noise in visual odometry passing through a target based on a predefined map becomes challenging. In order to accommodate the on-board resources without compromising thrust to weight ratio, a custom quadrotor platform was developed. This system integration helps getting RGB images, depth map, 3D point cloud and 6-DOF position tracking data from the stereo vision which were further processed to accomplish autonomous navigation in obstacle populated environments.

Contents

Contents

Contents

List of Figures

List of Figures

List of Tables

Glossary

$\mathbb{R}$	Set of real numbers
$\mathcal{W}$	Workspace where $\mathcal{W} \subset \mathbb{R}$
$\mathcal{O}$	Obstacles in workspace where $\mathcal{O} \subset \mathcal{W}$
$\mathcal{C}$	Configuration space
$\boldsymbol{q}$	Point in configuration space where $\boldsymbol{q} \in \mathcal{C}$
$\mathcal{A}(\boldsymbol{q})$	Set of occupied points by the robot while maintaining a configuration $\boldsymbol{q}$ where $\mathcal{A}(\boldsymbol{q}) \subset \mathcal{W}$
$\mathcal{G}(\boldsymbol{q})$	Set of occupied points by the camera's field of view while maintaining a configuration $\boldsymbol{q}$ where $\mathcal{G}(\boldsymbol{q})$
$\mathcal{C}_{obs}$	Configuration in collision
$\mathcal{C}_{obs}$	Collision-free configuration
$\boldsymbol{q}_d(k)$ or $\boldsymbol{q}_d^k$	Desired final trajectory
$\boldsymbol{q}_s(k)$ or $\boldsymbol{q}_s^k$	Trajectory generated by the controller for a T_h to check for collision
T_h	Time horizon for which trajectories $\boldsymbol{q}_s^k$ are generated
T_s	Sampling time

Glossary

T_{safety}	Minimum time before the predicted collision where switching occurs
$\boldsymbol{q}_0$	Initial Configuration
$\boldsymbol{q}_g$	Final/Goal configuration
$\boldsymbol{R}_E^B$	Z-X-Y Rotation matrix
$\boldsymbol{u}$	Control input to the quadrotor UAV
$\boldsymbol{r}$	Point in pixel coordinates
$\boldsymbol{P}$	Point in 3-D world coordinates
$\{B\}$	Body frame of reference
$\{E\}$	Inertial frame of reference
$\{\mathcal{F}_\mathcal{W}\}$	Workspace frame of reference
$\boldsymbol{q}_T$	Configuration point $\boldsymbol{q}$ in camera coordinates
$\boldsymbol{L}$	Set of hybrid control modes
$\boldsymbol{E}$	Set of all the transitions in an hybrid automaton
$G(l, l')$	Guard for a transition $\boldsymbol{e} = (l, l')$
$\mathcal{U}^p(\boldsymbol{x}(k))$	Set of motion primitives from the state $\boldsymbol{x}(k)$
$\boldsymbol{x}(k)$	Set of state space variables
$V(\boldsymbol{x})$	Lyapunov function
λ	Eigen value
D_s	Switching domain

Chapter 1

Overview

1.1   Introduction

Our problem focuses on developing an approach using active perception and hybrid control aspects for fast, vision-based navigation in cluttered environments. This can also be referred to as a system which is 'flying like a bird'. Our approach is inspired by the 'race the sun' game in which the UAV has to go as fast as possible, saving its energy and avoiding obstacles on its way. Our problem is derived from the fact that a bird flies forward to reach a certain destination while changing its strategy in case of any obstacles on the way followed by forward motion to reach its final destination again. From a controls perspective this can be seen as a hybrid control strategy assisted by a fast perception scheme. Moreover, since it is always moving forward towards its destination, it is not required to keep memory, which is typically stored in the form of an occupancy grid/3D map. Additionally, it is not required to search the whole 3D space for feasible paths if the system can look for closer escapes routes if obstacles hinder its way. For this purpose, we used the concept of collision checking in disparity space similar to [14], [26], and [12]. However, in the former two the single point query is performed while collision checking. The idea is to perform C-space expansion on the depth image in all of the dimensions. This expansion is

1

related to the safety radius of the quadrotor which is approximated by a sphere. Although, it presents an easy single point query strategy for collision checking, the expansion on the whole image takes a lot of computational resources while all the image space is not being queried for collision checking. Our perception strategy is a version of [12]. We adopted the idea of projecting the 2D vehicle image (at a certain configuration to be queried for feasibility) in the depth space and comparing the actual depth with the position of the 2D vehicle image. In addition, for planning the trajectories we used precomputed action primitives as our set of hybrid controllers with certain switching strategies. The first controller is responsible for taking the system to a global goal while the second controller performs the diverting maneuver in case of any obstacle.

1.2   Motivation

Quadrotor UAVs have recently been influencing many areas of research. They are useful in various applications including agriculture, search and rescue, transportation, photography and others which require cooperative execution of tasks. Some of them require outdoor navigation while some are concerned with indoor navigation and sometimes mapping. Due to an increasing interest in their applications, the autonomous navigation issue is gaining importance. Moreover, the system should also be agile and able to recognize and avoid obstacles in its way. GPS is considered a ready-to-use option in outdoor environments. However, there are currently many inevitable problems associated with GPS. While it does not guarantee reliable communication with the necessary satellites, efforts are needed to ensure accurate position tracking and navigation especially indoors. Moreover, GPS modules with better resolution and accuracy are expensive and are only responsible for giving position data. Motion Capture systems solve this problem to an extent by providing fast and accurate pose information to a robot. However, a completely on-board source

of feedback is still required for a number of tasks to make a robot independent of external stationary sensors. In this kind of situation camera based navigation solves these problems. Based on the processor speed, images can be used to receive pose information. Moreover, they can be processed to help a quadrotor sense the environment better via depth maps, point clouds and other related information. In other words, it is an all-in-one solution for autonomous navigation.

Figure 2.2 shows the test bed at Multi-agent Robotics and Heterogeneous Systems (MARHES) Laboratory at the University of New Mexico. The previous testbed consisted of AscTech Hummingbird quadrotors at the lab which could only fly within a VICON motion capture area. Due to various hardware constraints and limitations the Hummingbird quadrotors could not be upgraded with vision tools. Therefore, there was a need to develop a new architecture for experiments based on quadrotors equipped with vision sensors and interfaces. For this purpose, a ROS-based architecture exploiting the capabilities of a single stereo camera as a main position feedback sensor is proposed. The architecture is then adopted by different revisions of quadrotors at the MARHES Lab to come up with a fully customizable light weight platform for agile navigation. The new setup helped to introduce a fleet of quadrotors to a family of three hummingbird quadrotors. These new quadrotors could not only fly in the motion capture system but also made it possible to equip them with vision capabilities as described in this document. The first version was fully built from scratch to enable it to carry a Jetson TK1 single-board computer and required hardware for camera and other interfaces. Jetson is a powerful processor made by NVIDIA. It is one of the best embedded computers available so far for vision and artificial intelligence applications for use on-board autonomous vehicles. TK1 is the first version in the the Jetson series. This version of the quadrotor equipped with the required hardware made it possible to fly indoors with visual-inertial odometry (VIO) to accomplish the desired tasks autonomously and independent of any external feedback in real-time. The final revision was built utilizing a Luminier QAV250 frame. It was

then customized to include a small sized supercomputer Jetson TX2 by NVIDIA and a ZED mini stereo camera by Stereo Labs. This platform is lighter and faster which enabled the implementation of agile navigation with real-time obstacle avoidance in indoor cluttered environments. After building such an interesting platform, the versatile capabilities of ROS and Ubuntu running on the on-baord supercomputer were exploited for various applications discussed individually in the following chapters.

1.3   Literature Review

Vision-guided Quadrotor UAV navigation has been the focus of many researchers for the past decade. From a practical standpoint, the real-time functionality of the strategy is required along with adequate proofs of a system's stability during aggressive maneuvers. There has been a lot of improvements in the efficiency of the algorithms proposed during this time in order to guarantee the real-time fast execution speeds of these algorithms. These research advancements are also thankful to advances in the computational efficiencies of more recent processing units. Among the prominent works starting from the early 2000s include [6], which describes the use of mixed-integer linear programming (MILP) to solve the non-convex obstacle avoidance optimization problem. Similarly there a sensor path planning work by Ferrari et al. [7] [8] [24]. This can be extended to involve the perception aspects and the real-time performance of the system in cluttered environments. Moreover, solving an MILP optimization problem in real-time might be slower and the number of constraints may also increase with an increase of the number of obstacles. Finding those constraints real-time using on-board perception adds another layer of expensive computations. There had been a lot of progress in the methods to facilitate the optimization and cost-function changes on-board as well. Some of the works include [16], [15] and [13]. Some other works around this time include [3] and [4] which use the concept of simultaneous localization and mapping (SLAM) for indoor flights in GPS-

denied environments. Later, the works similar to Oishi's and Tapia's deal with the use of stochastic reachability for collision avoidance [10] and [11]. This can be extended to guarantee the real-time feasibility of their algorithms. Moreover, in a completely unknown environment the perception techniques are of significant importance while determining the efficiency of the algorithms along with the path planning techniques. These works, however, either rely on on-board trajectory optimizations, mapping or both.

On the other hand, there are methods developed for efficient perception for fast UAV navigation like [5], [17], [18] and [19]. Authors of [5] claim to be the first one in developing a system with complete on-board processing for way-point navigation in complex environments. This lead to the development of various much efficient in this area afterwards. It relies on calculating 3D maps of the environment on-board. [17] presents a very efficient event-triggered approach to depth perception. The idea is that the system performs the disparity checks when needed. However, they are using sampling-based techniques to demonstrate the planning aspect of their problem. [18] presents an approach to use different sensors to self-learn the distance on-board. This approach can be extended for self-learning different environment parameters using monocular cameras as a primary sensor. [19] highlights an approach to use ego-centric cylinders for environment perception as a better approach to saving and processing a complete environment map in memory. This requires some memory to save temporary 360 degree views before they are forgotten which depends on a forgetting factor.

Among the latest work more closely related to our problem is [22], [21] and [25]. The work by Barry, Florence and Tedrake ([22]) uses a library of seven precomputed trajectories. The trajectories are then connected and executed on-board in real-time. However, the stability gaurantee while switching between precomputed trajectories is out of the scope of their work. Secondly, they considered all the trajectories starting from a single state. This means that every time the trajectory has to start from

a state which is not one of the starting states for the library trajectories, the system slowly is brought closer to the trajectory using another controller. [21] describes another way of approaching the problem of agile UAV navigation in cluttered environments. They use an A* algorithm for path planning with local and global map information. They are also performing trajectory optimization on-line. The work by Richter and Roy ([27]) is also a great contribution which combines deep learning and vision-guided navigation. It mostly scopes how to ensure safety in case of strange input on which a machine learning network is not trained. Among other recent works are [44] and [45].

1.4   Problem Formulation

The problem setup includes obstacles $\mathcal{O}_i \subset \mathcal{W}$, of unknown geometries and locations, uniformly distributed in a workspace $\mathcal{W} \subset \mathbb{R}^3$. Let $\mathcal{F}_\mathcal{W}$ be the reference frame fixed with the workspace $\mathcal{W}$. The workspace is assumed to be compact. Let $\boldsymbol{q} = (x, y, z, \theta) \in \mathcal{C}$ be a point in configuration space $\mathcal{C}$ of the quadrotor. The workspace quadrotor $\mathcal{A}(\boldsymbol{q}) \subset \mathcal{W}$ is expressed as a set of all the points occupied by the quad-rotor while maintaining a certain state $\boldsymbol{q}$. $\mathcal{G}(\boldsymbol{q}) \subset \mathcal{W}$ is expressed as a set of all the points occupied by the cameras' field of view while the quad-rotor maintains a certain state $\boldsymbol{q}$. The configuration for which the quad-rotor will collide with the obstacle is given as $\mathcal{C}_{obs} = \{\boldsymbol{q} \in \mathcal{C} : \mathcal{A}(\boldsymbol{q}) \cap \mathcal{O}_i \neq \phi\}$. In order to ensure collision free maneuver, the allowed configuration is $\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obs}$.

The problem is to find a trajectory $\boldsymbol{q}_d(k) : [0 \quad k_f] \rightarrow \mathcal{C}_{free}$ such that $\boldsymbol{q}_d(0) = \boldsymbol{q}_0$ and $\boldsymbol{q}_d(k_f) = \boldsymbol{q}_g$, where k_f is the final time and $\boldsymbol{q}_0$ and $\boldsymbol{q}_g$ are the initial and goal configurations respectively. The goal configuration forms a plane in $y - z$ axis as inspired by 'Race the sun' game. This means that $\boldsymbol{q}_g = (x_g, y, z, \theta)$. Here y, z, θ can be any real number while x_g defines the location of the destination plane. However, our strategy can easily be extended to include a certain point as a destination.

Moreover, it is also assumed that the vehicle is facing towards its goal initially *i.e.* $\mathcal{F}_{\mathcal{W}}$ coincides with the vehicle's reference frame.

1.5 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 describes the simplified quadrotor UAV model and an introduction to the hardware and software components used in this thesis. Chapter 3 describes the stereo camera model and the perception techniques used for obstacle avoidance. The control strategy is presented in Chapter 4 along with the required proves to confirm the hybrid controller stability. Chapter 5 goes through the experiments related to various applications related to vision-based navigation in cluttered environments using the MARHES testbed. Moreover, it also presents the comparison between the proposed hybrid trajectory generation and artificial potential field approaches. The implementation and simulation results are also shown. Lastly, Chapter 6 presents the conclusion and future work.

Chapter 2

System Overview

2.1 Simplified Quadrotor UAV Dynamic Model

The quad-rotor model is inherently non-linear. The system has four inputs to facil-
itate the maneuver in 3-dimensional space. Inertial frame of reference corresponds
to the coordinate system with respect to the earth while the body frame of reference
corresponds to the coordinate system with respect to vehicle body. Roll, pitch and
yaw refers to the rotation of the quad-rotor around x,y and z axis respectively. ϕ,
θ and ψ denote these angles in the corresponding order. These angles are measured
with respect to the body frame. If a quad-rotor starts from rest to a certain point
in space, it has to undergo translations and rotations. The rotation matrix $\boldsymbol{R}_E^B$
transforms the coordinates from body fixed frame to inertial frame and is given as

$$\boldsymbol{R}_E^B = \begin{pmatrix} c\psi c\theta - s\phi s\psi s\theta & -c\phi s\psi & c\psi s\theta + c\theta s\phi s\psi \\ c\theta s\psi + c\psi s\phi s\theta & c\phi c\psi & s\psi s\theta - c\psi c\theta s\phi \\ -c\phi s\theta & s\phi & c\phi c\theta \end{pmatrix}, \tag{2.1}$$

where c(.) and s(.)represent cos and sin functions respectively. In a quad-rotor UAV
each rotor generates its own thrust according to the revolution per minute (RPM)
of the motor and propeller profile. The following relations relate the thrust and

moment generated by a rotor with its angular velocity.

$$F_i = k_f \omega_i^2. \tag{2.2}$$

$$M_i = k_m \omega_i^2, \tag{2.3}$$

where F_i, M_i and ω_i are the thrust, moment and RPM of the ith rotor respectively and k_f and k_m are the constants depending on the propeller profile.

Four inputs to the system are the total thrust of all four rotors and rolling, pitching and yawing torques.
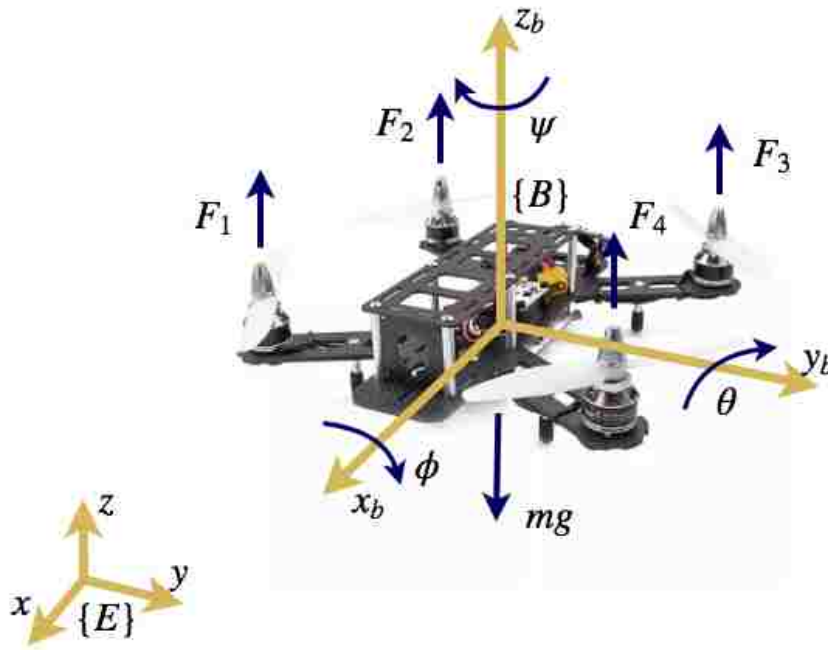
Figure 2.1: A quadrotor UAV physical model.

Figure 2.1 shows a typical physical model of a quad-rotor UAV. The rolling and pitching torques are generated through the angular momentum and thrust difference between opposite rotors corresponding to each axis. Rotors 1 and 3 are responsible for generating pitching torque and rotors 2 and 4 affect rolling torque. The four

inputs can be written as follows:

$$u_1 = \Sigma F_i,$$

$$u_2 = l(F_2 - F_4),$$

$$u_3 = l(F_1 - F_3),$$

$$u_4 = M_1 - M_2 + M_3 - M_4,$$

where M_i is the moment generated by the ith rotor perpendicular to its plane of rotation.

As stated earlier, a quadrotor is an under-actuated system with four inputs. The conventional inputs to a quadrotor system are

$$\boldsymbol{u} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}^T$$

where u_1 is the total thrust generated by all four rotors and u_2, u_3, u_4 represent the rolling, pitching and yawing torques respectively. The states and outputs for the system are given as

$$\boldsymbol{x}_m = \begin{bmatrix} x & y & z & \phi & \theta & \psi & \dot{x} & \dot{y} & \dot{z} & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T,$$

$$\boldsymbol{y}_m = \begin{bmatrix} x & y & z & \psi \end{bmatrix},$$

where x, y and z represent position of the center of quadrotor with respect to the inertial frame of reference $\{E\}$ whereas ϕ, θ and ψ corresponds to roll, pitch and yaw of the quadrotor body frame with respect to the inertial frame. The model derived from Newton-Euler equations under simplified assumptions ([1]- [2]) can be derived

as:

$$\Sigma \boldsymbol{F}_r = m\boldsymbol{a}, \tag{2.4}$$

$$\boldsymbol{R}_B^E \begin{bmatrix} 0 \\ 0 \\ u_1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} = m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix}, \tag{2.5}$$

$$\Sigma \boldsymbol{\tau}_r = \boldsymbol{I}\boldsymbol{\alpha}, \tag{2.6}$$

$$\begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \boldsymbol{I} \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \boldsymbol{I} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}, \tag{2.7}$$

where p, q and r represent angular velocities in 3 dimensional body frame of reference $\{B\}$. $\boldsymbol{F}_r$ and $\boldsymbol{\tau}_r$ are the three dimensional forces and torques acting on the quadrotor as a result of thrusts generated by the four rotors. $\boldsymbol{R}_B^E$ is the rotation matrix from body fixed frame to world frame. Z-X-Y convention is considered for this matrix. It is generally useful to express the second set of equations in body frame. Moreover, using the small angle assumption for linear systems the equations in state space form

are :

$$\dot{x}_1 = x_7,$$

$$\dot{x}_2 = x_8,$$

$$\dot{x}_3 = x_9,$$

$$\dot{x}_4 = x_{10},$$

$$\dot{x}_5 = x_{11},$$

$$\dot{x}_6 = x_{12},$$

$$\dot{x}_7 = (\cos\psi\sin\theta + \cos\theta\sin\phi\sin\psi)U_1/m,$$

$$\dot{x}_8 = (\sin\psi\sin\theta - \cos\theta\sin\phi\cos\psi)U_1/m,$$

$$\dot{x}_9 = (\cos\phi\cos\theta)U_1/m - g,$$

$$\dot{x}_{10} = x_{11}x_{12}(I_{yy} - I_{zz})/I_{xx} + U_2/Ixx,$$

$$\dot{x}_{11} = x_{10}x_{12}(I_{zz} - I_{xx})/I_{yy} + U_2/Iyy,$$

$$\dot{x}_{12} = x_{10}x_{11}(I_{xx} - I_{yy})/I_{zz} + U_3/Izz.$$

(2.8)

The nonlinear model given by Equation (2.8) is linearized about the equilibrium point,

$$\boldsymbol{x}_m^* = \begin{bmatrix} \bar{x} & \bar{y} & \bar{z} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T.$$

where $\bar{x}$, $\bar{y}$ and $\bar{z}$ are the coordinates in $\{E\}$ at hover. A general form of linearized system is given by

$$\dot{\boldsymbol{x}}_m = \boldsymbol{A}\boldsymbol{x}_m + \boldsymbol{B}\boldsymbol{u},$$

$$\boldsymbol{y}_m = \boldsymbol{C}\boldsymbol{x}_m + \boldsymbol{D}\boldsymbol{u},$$

(2.9)

where

$$
\boldsymbol{A} =
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & gx_6^* & g & gx_4^* & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -g & gx_6^* & gx_5^* & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I_x x_{12}^* & I_x x_{11}^* \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I_y x_{12}^* & 0 & I_y x_{10}^* \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I_z x_{11}^* & I_z x_{10}^* & 0
\end{bmatrix},
$$

$$
\boldsymbol{B} =
\begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
(x_5^* + x_4^* x_6^*)/m & 0 & 0 & 0 \\
(-x_4^* + x_6^* x_5^*)/m & 0 & 0 & 0 \\
1/m & 0 & 0 & 0 \\
0 & 1/I_{xx} & 0 & 0 \\
0 & 0 & 1/I_{yy} & 0 \\
0 & 0 & 0 & 1/I_{zz}
\end{bmatrix},
$$

$$
C = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix},
$$

here

$$
Ix = \frac{I_{yy} - I_{zz}}{I_{xx}},
$$

$$
Iy = \frac{I_{zz} - I_{xx}}{I_{yy}},
$$

$$
Iy = \frac{I_{xx} - I_{yy}}{I_{zz}}.
$$

2.2   Hardware
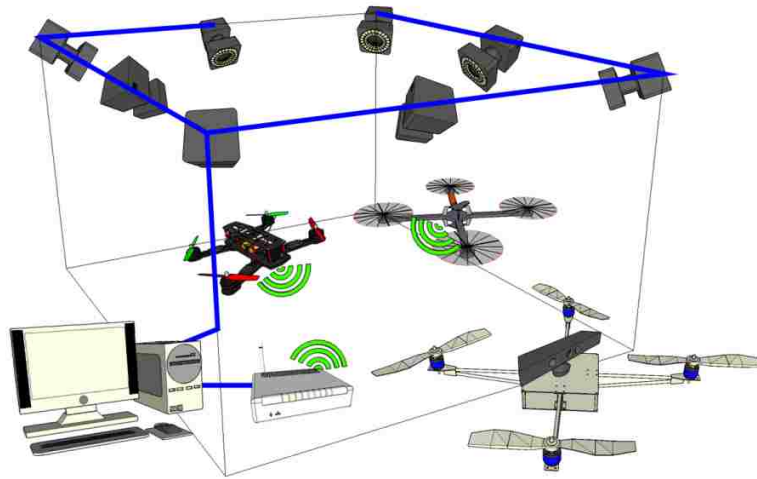
Figure 2.2: MARHES Aerial testbed

The MARHES aerial testbed consists of three AscTec Hummingbirds quadrotors flying under vicon motion capture system. The network architecture being used is shown in Figure 2.2. This setup is capable of proving highly accurate feedbacks upto

millimeter accuracy at very fast speeds. However, there was a need for expansion of this aerial testbed especially for robotic swarm and vision based navigation applications. Among the new additions to the testbed includes the new quad fleet of quadrotors based on Luminer QAV250 frames and newly released Intel Aero drones. This new family of quadrotors are equipped with vision capabilities to make them independent of any external feedback sources. This makes them capable of flying inside the motion capture system as well as outside. All the robots are ROS based. This adds a whole set of new possibilities of system modifications. Moreover, it provides more flexibility for a robotics software developer on such kind of systems. Stereo camera is used as a primary vision sensor along-with the IMU to provide VIO in real-time. However, the systems are capable of interfacing more vision sensors such as downward facing monocular cameras for VIO on-board. The new quadrotors have higher payload capacity to accommodate additional sensors. The low -level attitude control system is provided by an open source firmware PX4 on all the systems. Different computers are used for high-level trajectory generation and image processing keeping in view the computational power and efficiency required. Three of those robots are described as follows

2.2.1   LoboDrone
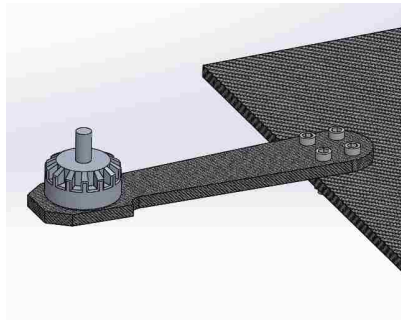
Figure 2.3: SolidWorks model of an attached arm.
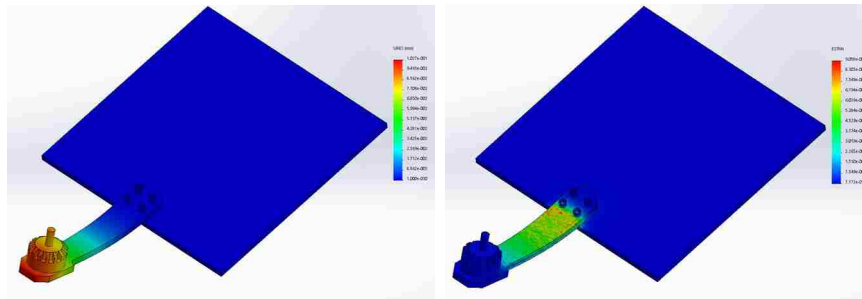
Figure 2.4: Strain and stress measurements.

Figure 2.5: LoboDrone v1.0 quadrotor frame with actuators.

This quadrotor is built from scratch making it fully compatible with the required sensors for perception and planning. The carbon fiber plates are cut to built the frame according to the right size. The case on top carrying the cameras and a USB 3.0 hub is 3D printed. Figure 2.5 shows the picture of the assembled frame without computer and camera.

The frame designed consists of a 4mm carbon fiber bottom plate, on which are bolted four commercially available quadrotor arms. Stand-offs are used to add a second platform, this time made of 2mm carbon fiber. The Jetson is mounted on the bottom plate, and the PixRacer hangs from the top plate. This way, all onboard computing and flight control devices are protected, and the top plate is free to

mount sensors like the ZED camera. The battery is strapped to the bottom of the vehicle. The biggest concern with the custom platform is the joint between the motor arms and the bottom plate, as this joint must carry the weight of the vehicle and remain rigid amidst potentially high forces and torques. SolidWorks is used to run a stress analysis at the joint [42]. To simulate the greatest stress, the bottom plate is fixed and a force is applied to the motor equal to its maximum thrust of 2 kg. The analysis concluded with a maximum possible displacement of 0.103 mm, stress of $1.860e^{-7}N/m^2$ which we determined, should not pose any issue. Figures 2.3, 2.4, 2.5 show the model of an arm, stress and strain measurements, and physical quadrotor respectively. Pixracer autopilot is used for low-level computational tasks like attitude stabilization with Jetson TK1 for high level trajectory generation and image processing. Both of them are described in the following subsections.

Tiger F80 1900kV motors are used for this platform carrying 6 inch propellers. With this configuration it can carry approximately 100 grams of payload while having a flight time of approximately 3 minutes with a 4s 3300 mAh battery. The airframe itself is 350mm in length when measured from motor shaft to motor shaft diagonally. The weight of this whole platform is approximately 1500 grams including battery and all the electronics.

2.2.2   Lumenier QAV250

The newer version of LoboDrone includes a supercomputer with one of the most advanced stereo camera technology on-board. The frame used is Lumenier QAV250 airframe [50]. The airframe weighs 170 grams and is 250mm in length when measured diagonally from motor shaft to motor shaft.

This version is smaller, lighter and has better on-board computer than the first version of LoboDrone. With a 4s 3300mAh battery the flight time is recorded as approximately 10 minutes. The stereo camera used is Zed mini. This is the newest version of Zed camera by Stereo Labs and is designed to be used for mixed-reality
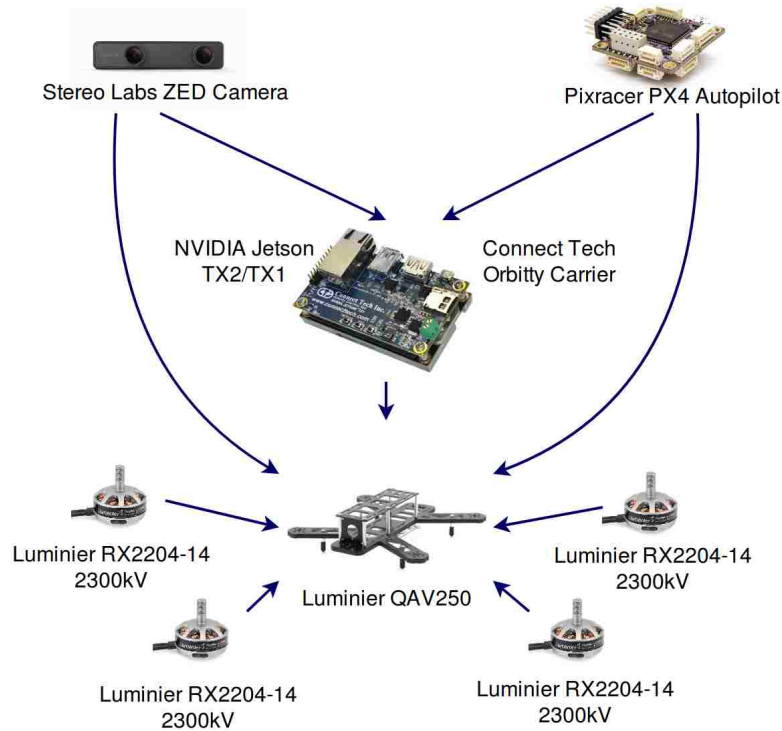
Figure 2.6: LoboDrone v2.0 quadrotor assembly.

applications. The weight of the whole system including the battery and all the electronics is approximately 900 grams. The actuators used are Lumenier 2204 2300kV motors. The controllers used for these motors are Luminier BL Heli F390 30A electronic speed controllers with autoshot/active braking capability. Oneshot serves as a faster ESC protocol replacing the old pulse width modulation (PWM) protocol. With 5 inch propellers this setup can carry more payload as well *i.e.* ≈ 800 grams. Additionally, pixracer flight controller is responsible for performing the low level computational tasks like attitude stabilization while NVIDIA Jetson TX2 is the supercomputer on-board which is used for high level perception and planning computations.

2.2.3   Intel Aero

Intel ® Aero flight platform comes as a ready to fly drone kit which is a pre-assembled quadrotor UAV ready to fly through an RC transmitter. Main sensors and computer boards mounted on this platform are an aero compute board, a flight controller with PX4 autopilot, and an Intel Realsense camera and other required peripherals for flight. Its compute board is an Intel ® Atom ™ x7-Z8750 processor with 4 GB LPDDR3-1600 RAM and 32 GB eMMC memory. Attached to it are a forward facing 8MP RGB camera and a downward facing monochrome VGA camera. Moreover, Intel Realsense is well known for its computational capabilities for depth maps. The on-board version of this camera is R200 which is very old now but it can easily be replaced by a newer one. It supports external USB3.0 devices as well. This pre-assembled architecture serves as a good platform for development especially for perception algorithms on-board. Similar to ARM processors, it can run Ubuntu and ROS which can be easily installed on them. Figures 2.7 and 2.13 show the items in an Intel Aero RTF kit and its compute board respectively.

Figure 2.7: Intel Aero Ready to Fly (RTF) kit.

2.2.4   NVIDIA Jetson

NVIDIA took the computational power on robots especially on small autonomous drones to the next level [48]. Jetson TK1 is the first one in the series with NVIDIA 4-Plus-1 ™ Quad-Core ARM ® Cortex ™ -A15 CPU and NVIDIA Kepler GPU with 192 CUDA Cores. It has 16 GB internal eMMC memory. Its dimensions are 127mm x 127mm. The newer ones are Jetson TX1 and TX2. They both are supercomputers contained in small modules. Jetson TX1 is the first ever supercomputer on a module. They run on NVIDIA Maxwell ™ and Pascal ™ GPUs respectively. They both have 256 CUDA Cores and are 50 mm x 87 mm in size which is reasonably small for a small scale quadrotor UAV. CUDA is a programming model and framework for parallel computing. All of them have USB 3.0 and USB 2.0 devices compatibility, 1 Gigabit Ethernet, 802.11ac WLAN and Bluetooth. These features make them very suitable for deep learning, vision and other GPU computing applications on small scale robots. Figures 2.8 and 2.9 show how these modules look like.   Jetson TK1 is

Figure 2.8: Jetson TK1.

used in the first prototype of LoboDrone. The main drawback of using it over the other two is that its much larger in size and support Ubuntu 14.04 which does not support the latest softwares like ZED camera SDKs for the newer cameras. Jetson TX1/TX2 has support for Ubuntu 16.04 which solves these problems. Moreover, it can also run ROS Kinetic, the latest stable version of ROS. However, Jetson
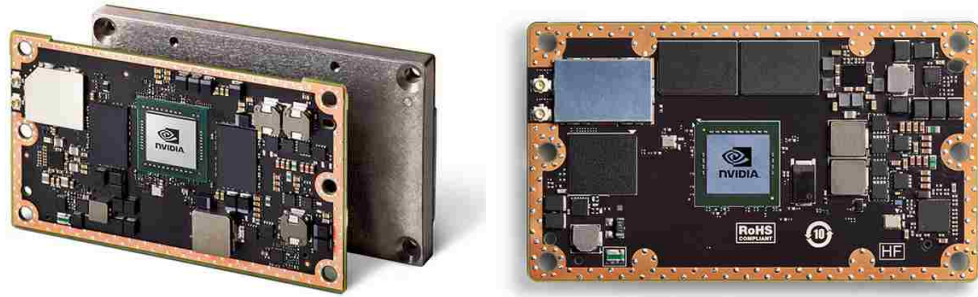
Figure 2.9: Jetson TX1 and TX2 modules.

Figure 2.10: Orbitty carrier board on Jetson TX1.

TX1/TX2 cannot be mounted directly on a quadrotor since it does not provide easy interface for the required peripherals. Figure 2.10 shows the orbitty carrier board sold be Connect Tech Inc. It is a 87mm x 50mm lightweight board which provides interface for gigabit ethernet, USB 3.0, USB 2.0, HDMI, MicroSD, 3.3V UART, I2C, and GPIOs. This makes it easier to connect external devices to the Jetson TX1/TX2 modules. This setup is used in the latest revision of LoboDrone using a QAV250 airframe.

Figure 2.11: ZED and ZED mini stereo cameras.

2.2.5   Stereo Labs ZED Camera

ZED camera being the world's first 3D camera for motion tracking and depth sensing is the state of the art in stereo vision to facilitate robot navigation [49]. There are two versions of ZED cameras available commercially as shown in Figure 2.11. The older standard ZED camera is a little bigger than the newer ZED mini. Both of them weigh ≈ 160 and ≈ 63 respectively. Both are capable of 6-axis positional tracking and stereo-inertial simultaneous localization and mapping (SLAM). However, the former can compute depth at longer distances *i.e.* from 0.5 to 20 meters while the later is capable of computing depth from 0.1 to 12 meters. They both have highly accurate motion sensors like accelerometer and gyroscope with sampling rate of 500Hz. They have a position and orientation accuracy of $+/-1$ mm and 0.1^o respectively with up to 100 Hz refresh rate. The standard ZED can go up to 2K resolution while ZED mini can go up to 2.2K resolution also claiming to be the fastest depth cameras. ZED mini is the first camera for mixed-reality applications. Stereo Labs provides a ROS wrapper to provide an interface for use with ROS. For use with Jetson and ROS on-board, the resolution and refresh rate options become limited accordingly.

2.2.6 Autopilot PX4

PX4 is an open-source autopilot firmware mainly designed for hobbyist. Using any hardware like pixhawk and its variants, makes it easier to design and test low level control and high level planning algorithms. The main advantage of using such framework is that it is compatible with ROS. It accepts commands in the form of ROS topics from any computer running ROS. Since ubuntu and ROS can easily be installed on ARM based processors like Odroid and Jetson, PX4 is very ideally suited as an autopilot for development and other customizations. There are a lot of different variants of hardware supporting PX4 firmware. One of them is pixracer which is used on both versions of LoboDrones. It has 180 MHz ARM Cortex $^®$ M4 with single-precision FPU and 256 KB SRAM. It has its own WiFi telemetry for getting live data and for firmware upgrades. Moreover, it provides interfaces for the external peripherals needed for flight like PPM input for Spektrum RC receiver, port for FrSky telemetry, and PWM out port for ESCs, SD card slot for logging and safety switch port and other connectors. Most essentially it has the sensors embedded in the module for flight like barometer, magnetometer/compass, gyroscope and accelerometer. Typically, an on-board computer is used to send trajectories in the form of position set points to the autopilot running PX4 which is assumed to take care of low level attitude stabilization while following the way-points. However, the autopilot-ROS interface, as described in the software section, allows the computer to send low level commands as well if needed. Figure 2.12 shows a pixracer hardware module which can run PX4 autopilot firmware [51]. There are some other platforms which have the autopilot and on-board computer embedded on a single chip. It eliminates the need to buy and mount separate computer and autopilot modules on a robot. Intel Aero is one of the platforms which have such an architecture. It has an intel aero compute board mounted which provides all the computing required for a flight including the PX4 autopilot firmware. Figure 2.13 shows a picture of an Intel aero compute board.
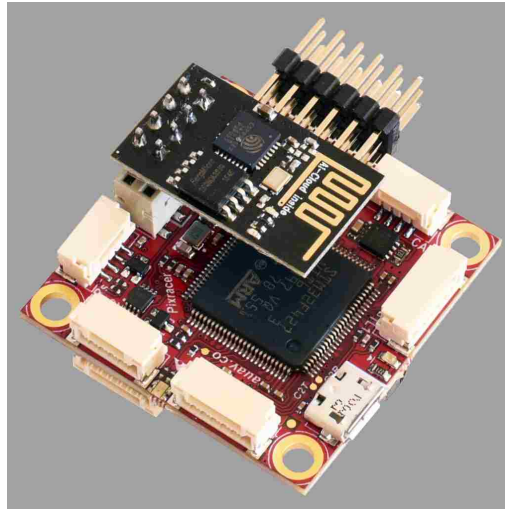
Figure 2.12: A pixracer module

Figure 2.13: An aero compute board.

2.3 Software and Communication

2.3.1 ROS

Robot Operating System (ROS) provides the communication and low level framework for the architecture used. It has a modulated structure where nodes (containing C++ or Python functions) talk to each other using pre-defined classes of packets called ros messages. They can either use publish/subscribe model to route ros messages over user defined topics or use service/client. The former one is ideally suited for many to

many communication while the later is used for request/reply interactions. The whole architecture however, provides support for inter and intra robot node communication. Figure 2.14 shows a simplified version of such a system. Summarizing it, having a lot more to it than just mentioned, ROS is well suited for many robotics applications. It not only provides the communication framework but also software interfaces to many peripherals widely used in robotics. Moreover, its vast open source community adds a lot to the reuse-able software contributions by the developers.
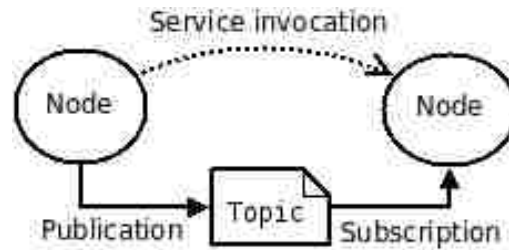
Figure 2.14: Basic ROS concept.

2.3.2 MAVLink Protocol

MAVLink is a communication protocol usually used by the UAVs for the data transfer to and from other devices. PX4 based autopilot modules typically use MAVLink protocol for communication. The wrapper for this protocol for use with ROS is mavros. It converts the MAVLink messages to and from ROS messages on the on-board computer running ROS while the autopilot being connected to any USB port on the computer.

2.3.3 Jetson Setup

As mentioned earlier NVIDIA Jetson is used on most of the vision equipped robots at the MARHES testbed. These Jetson board accompany JetPack which is a software that is used to flash the board with customized versions of Linux, opencv, CUDA

toolkit and many other vision and deep learning software libraries. Its latest release is JetPack 3.2 which comes with Linux for Tegra (L4T) 28.2, which is a lighter Ubuntu 16.04 adapted to use with Jetson TX1, TX2/TX2i. It also comes with CUDA 9.0 which is required by ZED mini software development kit (SDK) and ROS-wrapper. It requires a host PC running Ubuntu 14.04 or 16.04 to flash the board. Once the JetPack installation is complete, ROS and can easily be installed and its packages can be developed. Similar procedure can be used to setup TK1 but it only supports the outdated versions of these softwares which makes the development more difficult.

2.3.4 ZED Camera Setup

A ZED camera comes with its own SDK for developers. However, by using ros-wrapper for a particular SDK, the development environment can be changed to ROS which provides necessary messages and topics to subscribe from and publish to. The latest version of ZED SDK is 2.4 which is supported by Jetson TX1 and TX2. It works with JetPack 3.2 and CUDA 9.0. As mentioned earlier when SDK is used with ROS-wrapper it publishes the odometry, the coordinate frames transformations, the rectified and raw image topics for all the cameras and depth and the point cloud in the form of ROS messages.

Chapter 3

Perception

3.1   Stereo Camera Model

Stereo camera is used to compute depth of the scene. A stereo camera takes advantage of two cameras to calculate the disparity through triangulation. This is governed by epipolar geometry.

3.1.1   Epipolar Geometry

Epipolar geometry can be visualized in Figures 3.1 and 3.2. Figure 3.1 shows two camera planes at some angle. O_L and O_R represent the optical centers of the two cameras. An optical center is a point from where all the projection lines of a camera must pass. This is also known as camera center. Stereo vision set up includes two epipoles, one for each camera. They are labeled as e_L and e_R respectively. The epipole e_L refers to the point where the projection line from the optical center of the right camera intersects the image plane of right camera. In other words it is the image of the right camera's optical center as seen by the left camera. The opposite applies to the epipole e_R. Consider a point P in space. The line joining the optical center of left image O_L and the point P is projected to a point p_L in the left image
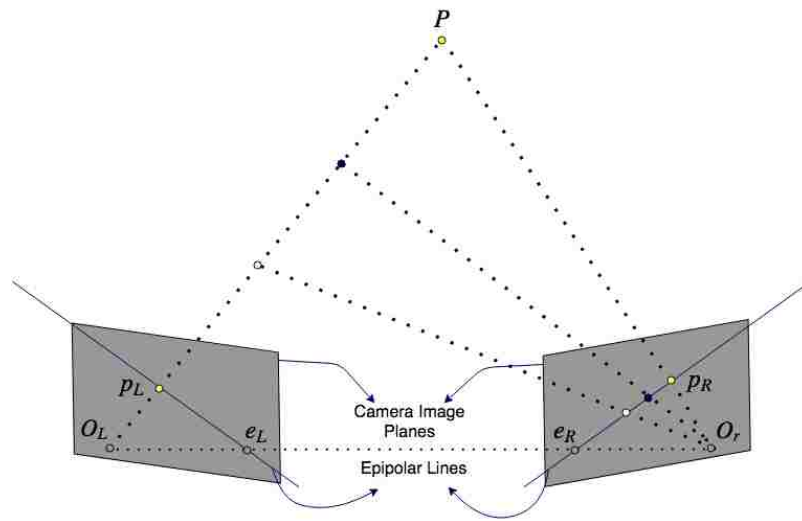
Figure 3.1: Epipolar geometry with non-parallel camera frames.
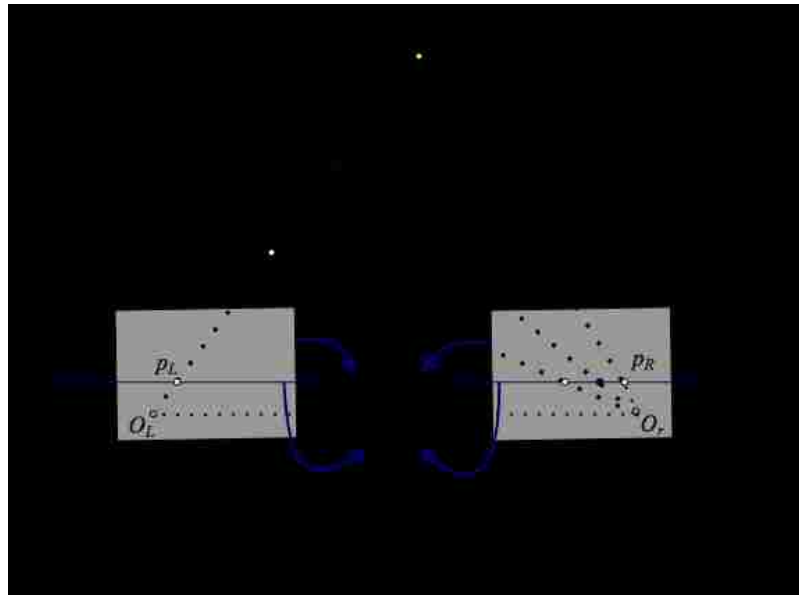
Figure 3.2: Epipolar geometry with parallel camera planes.

plane. The projection of this line in the right image plane is known as epipolar line. The point P and both O_L and O_R lie in one plane called epipolar plane. The epipolar plane intersects the image planes at the epipolar lines.

Any point projected in the left image plane can be the projection of one of the points on line PO_L, in the right image plane. Therefore, the point p_L must appear on the epipolar line $p_R e_R$. This is called *epipolar constraint*. Once that point is found depth is calculated through triangulation. In case of cameras with parallel image planes, the epipolar line will become a horizontal line as shown in Figure 3.2 and the point p_L can be found by matching it with the pixels on this line.

3.1.2   Pinhole Camera Model and Triangulation
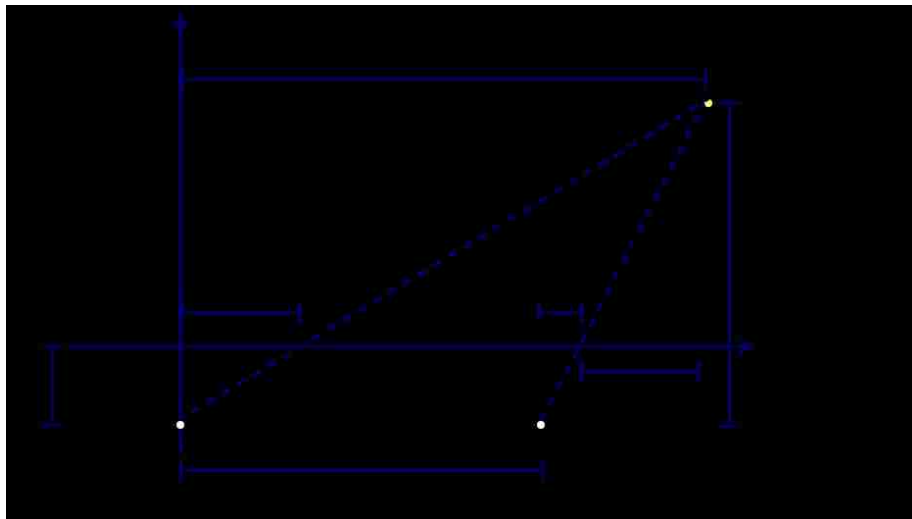
Figure 3.3: Triangulation geometry.

A pinhole camera model for both the cameras can be used to calculate the depth from disparity by a process called triangulation. Let f be the focal length of the camera. It is defined as the distance from the optical center of a camera to its image plane. b is the stereo baseline which refers to the distance between two optical centers. The stereo vision setup is shown in Figure 3.3. If we know the pixel values of the projection of point P we know the projection lines and hence we can utilize

some principles from geometry. From similar triangles,

$$\frac{z}{f} = \frac{x}{xl},$$

$$\frac{z}{f} = \frac{x - b}{xr},$$

$$\frac{z}{f} = \frac{y}{yr} = \frac{y}{yl}.$$

The depth can be calculated from triangulation:

$$z = f \times b/(xl - xr) = f \times b/d,$$

$$x = xl \times z/f = (b + xr) \times z/f,$$

$$y = yl \times z/f = yr \times z/f.$$

It can be noted that the disparity d is inversely proportional to depth of a pixel. A pinhole camera model is considered for depth space collision checking.

3.1.3 Calibration Parameters

The calibration of a stereo camera yields two set of parameters for each of the two cameras. They are known as intrinsic and extrinsic parameters. The former refers to the parameters such as focal length, the location of the optical center in an image (in pixels) while the later contains parameters such as relative rotation and translation between the two cameras. Since the camera is modeled based on pinhole model, the conversion from homogeneous camera to homogeneous image coordinates is governed by the following projection equations:

$$\boldsymbol{r} \sim \boldsymbol{I}_C \boldsymbol{P} = \begin{bmatrix} fs_x & 0 & c_x & 0 \\ 0 & fs_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

If we want to write the projection equations from the homogeneous robot to homogeneous image coordinates we have to include the rotation and translation of the camera frame with respect to the robot frame and ultimately the world (or inertial frame). The camera frame is attached to the robot frame and all the computations

are done in the world coordinates. The above equation can be written as [12]:

$$
\boldsymbol{r} \sim
\begin{bmatrix}
fs_x & 0 & c_x & 0 \\
0 & fs_y & c_y & 0 \\
0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
\boldsymbol{R}_I^C & \boldsymbol{T} \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}.
\tag{3.1}
$$

Here $\boldsymbol{r}$ is the pixel coordinate corresponding to point $\boldsymbol{P}$. s_x and s_y are the pixel dimensions and c_x and c_y are the image frame coordinates of the location of camera optical axis.

3.2 Perception Strategy

Figure 3.4: Computation of the artificial shields and their transformations and projections in the depth image space.

While solving for path planning and collision avoidance problems in real-time, the need for an algorithm for efficient collision checking is inevitable. One way to do this is to make a 3-D occupancy grid and update it continuously in real-time using local vision data. However, this requires slightly more computation than necessary. A lot of work has been done using this idea of mapping the local subsets of the

whole configuration space. However, by performing the collision checking directly in depth-image space proved to be much faster. It is also more effective in the presence of image noise. This can also be called planning in perception space. The perception problem can be formulated as following.

Let the discrete trajectory generated by the controller to be $\boldsymbol{q}_s(\boldsymbol{x}_0, l_i, n) \subset \mathcal{C}$ where $\boldsymbol{x}_0$ is the initial state, l_i is the current mode and $0 < n < (T_h + T_{safety})/T_s$ is the local time of the generated potential trajectory. This trajectory has to be checked for collision before appending it in the final set of desired trajectories $\boldsymbol{q}_d(k)$. Here k is the global trajectory time since the start of the system. For simplicity the potential patch of the trajectory to be checked for collision and the final desired trajectory will be referred to as $\boldsymbol{q}_s^n$ and $\boldsymbol{q}_d^k$ respectively. The perception problem is to find the function $P : \mathcal{C} \to CState$ where $CState = \{0, 1, 2\}$ represents the collision state of the configuration. 0 and 1 collision states mean that $\boldsymbol{q}_s^n \in \mathcal{C}_{free}$ and $\boldsymbol{q}_s^n \in \mathcal{C}_{obs}$ respectively, while 2 means that the configuration is out of the field of view at a particular time when the check is performed *i.e.* $\boldsymbol{q}_s^n \notin \mathcal{G}(\boldsymbol{q}(k_c))$. Here $\boldsymbol{q}(k_c)$ is the current configuration of the quadrotor when the check is performed.

The collision detection is performed purely in depth space by exploiting the projection equations and the stereo camera model as described above. The motion planner generates a discrete trajectory in the form of set of points through the configuration space. Given a point in the configuration space, it can be checked whether it is in collision by projecting the robot in depth space. The quadrotor projection in depth space can be simplified as a projection of set of all points contained in a square. This square can be considered as a 'shield' for a quadrotor. The size of this square is assumed to be slightly larger than the longest dimension of the quadrotor while its position is such that its placed at some distance ψ_x in front of the the quadrotor in $\mathcal{F}_\mathcal{W}$. This is a pessimistic version of the strategy presented in [12] which considers the projections of the surface normals of a quadrotor as a shield. This shield keeps on changing shape depending on the configuration. However, we considered the con-

stant size of this square corresponding to the maximum size as of that proposed by the above reference.

The idea is that given a query point $\boldsymbol{q}_s^n$ in $\mathcal{F}_\mathcal{W}$, a corresponding artificial square shield is generated (Figure 3.4). This square refers to the set $\boldsymbol{s}(\boldsymbol{q}_s^n) = \{\boldsymbol{q} : \boldsymbol{q}_s^n(1) - \psi_y < \boldsymbol{q}(1) < \boldsymbol{q}_s^n(1) + \psi_y \;\; \& \;\; \boldsymbol{q}_s^n(2) - \psi_z < \boldsymbol{q}(2) < \boldsymbol{q}_s^n(2) + \psi_z \;\; \& \;\; \boldsymbol{q}(0) = \boldsymbol{q}_s^n(0) + \psi_x\}$, where ψ_x, ψ_y and ψ_z are the safety margins in corresponding directions. After the computation of set $\boldsymbol{s}(\boldsymbol{q}_s^n)$ for the query point $\boldsymbol{q}_s^n$, all the points contained in the set are first transformed in the vehicle body coordinate system and ultimately the camera coordinate system depending on the current quadrotor pose at time instant k_c and the pose of the attached stereo camera with respect to the vehicle body. This transformation is followed by a projection of $\boldsymbol{s}(\boldsymbol{q}_s^n)$ on the depth image. Combining these expressions we get Equation (3.1). Here $\boldsymbol{R}_I^C$ refers to the rotation from world (or inertial) frame to camera coordinate frame.

$$\boldsymbol{R}_I^C = \boldsymbol{R}_B^C \times \boldsymbol{R}_I^B. \tag{3.2}$$

where $\boldsymbol{R}_B^C = \boldsymbol{R}_y(-\pi/2) \times \boldsymbol{R}_x(\pi/2)$ is the rotation matrix from body to camera coordinate system and $\boldsymbol{R}_I^B$ is the ZXY rotation matrix corresponding to the current orientation of the robot.

The query trajectory point $\boldsymbol{q}_s^n$ is assumed to be collision free if all the points contained in the corresponding set $\boldsymbol{s}(\boldsymbol{q}_s^n)$ are collision free. It is outside the field of view if any point on the set does not correspond to a valid image point. Similarly, it is in collision if any point in that set is under collision. Given any point $\mathcal{C} \supset \boldsymbol{q} = (x, y, z, \theta)$ in world frame, it is transformed to a corresponding point $\mathcal{C} \supset \boldsymbol{q}_T = (x_T, y_T, z_T, \theta)$ in camera coordinates and projected to pixel $\boldsymbol{r}$ on the depth image.

$$\begin{bmatrix} x_T \\ y_T \\ z_T \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_I^C & \boldsymbol{T} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Formally,

$$P(\boldsymbol{q}) = 0 \quad \text{if} \quad D(\boldsymbol{r}) > z_T \ \forall \ \boldsymbol{q} \in s(\boldsymbol{q}_s^n),$$

$$P(\boldsymbol{q}) = 1 \quad \text{otherwise},$$

It can be noted here that rotation in place (θ) for the query trajectory points will not affect the collision detection because the quadrotor geometry is approximated by a rectangular 'shield' with safety margins in $\mathcal{F}_\mathcal{W}$. Function $D(\boldsymbol{r})$ refers to the depth of a pixel $\boldsymbol{r}$.

3.3 Escape Strategy
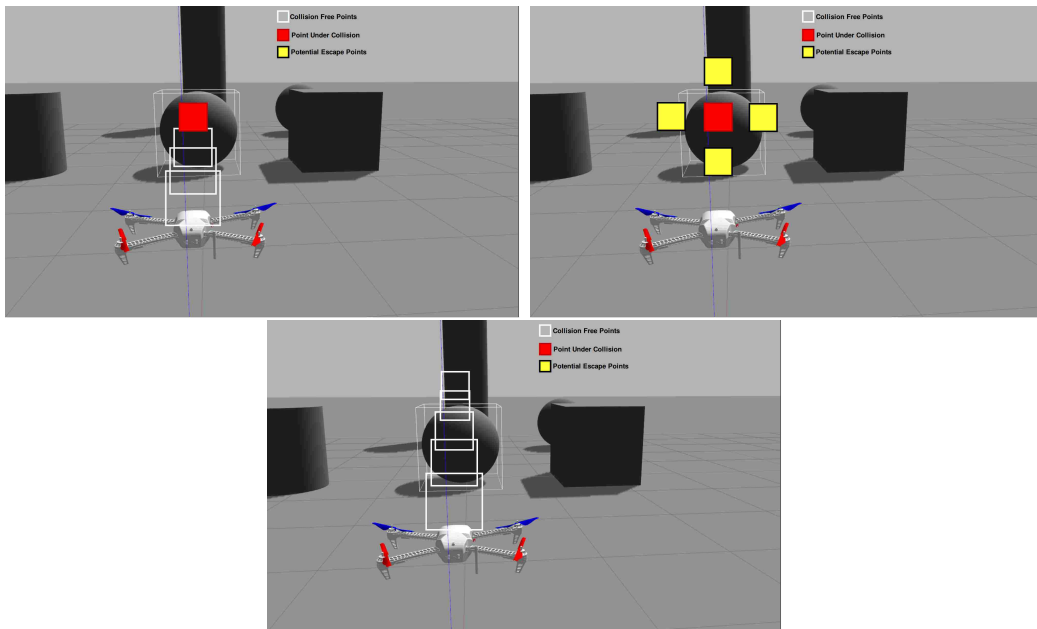
Figure 3.5: Escape point computation by querying potential points in 4 directions.

In the event $P(\boldsymbol{q}_s^n) = 1$, *i.e.* if the predicted trajectory experiences collision, the vehicle has to be deviated to the safe location. This location is referred to as escape point, the corresponding configuration and state vector is $\boldsymbol{q}_{esc}$ and $\boldsymbol{x}_{esc}$ respectively,

considering θ and the final velocities as 0. Let any trajectory point $\boldsymbol{q}_s^n$ be in potential collision, the image space in the neighborhood of this point in $y-z$ plane is searched for the high depth areas. Since searching the whole depth image in a continuous way is computationally expensive and time consuming, the search is performed in 4 different directions with discrete intervals. This quadrotor is projected in the depth space some distance d_l away from $\boldsymbol{q}_s^n$ in up, down, left and right directions and checked for collisions. This process is repeated with more projections at distance d_l from previously checked points, until a collision free point is found. This point is the escape point. To prevent the planner from getting stuck at a place in case of another obstacle suddenly appearing after the first one, the projections are performed in an $y-z$ plane a little way from the plane associated with $\boldsymbol{q}_s^n$. Figure 3.5 shows how the escape point check is performed in depth image space.

Chapter 4

Control Strategy

4.1 Trajectory Generation

The trajectory generation is performed in 3-D, utilizing a pre-computed set of control laws. While the first feedback law $\boldsymbol{u}_a$ tries to take the system to a global goal, the other feedback law $\boldsymbol{u}_b$ is responsible to divert the system from possible collisions. The state vector in continuous time can be written as,

$$\boldsymbol{x} = \begin{bmatrix} x & \dot{x} & y & \dot{y} & z & \dot{z} \end{bmatrix}^T.$$

It is assumed that the vehicle's frame always coincide with $\mathcal{F}_\mathcal{W}$ and a low level controller takes care of maintaining the heading. A double integrator model is used where:

$$\ddot{x} = u_1, \tag{4.1}$$

$$\ddot{y} = u_2, \tag{4.2}$$

$$\ddot{z} = u_3. \tag{4.3}$$

The differential equations are discretized with a sampling time of T_s to obtain discrete system matrices.

$$\boldsymbol{x}(k+1) = \boldsymbol{A}_d \boldsymbol{x}(k) + \boldsymbol{B}_d \boldsymbol{u}(k), \tag{4.4}$$

$$\boldsymbol{y}(k) = \boldsymbol{C}\boldsymbol{x}(k) + \boldsymbol{D}\boldsymbol{u}(k). \tag{4.5}$$

The two control laws are given as,

$$\boldsymbol{u}_a = \boldsymbol{K}_a \boldsymbol{x}, \tag{4.6}$$

$$\boldsymbol{u}_b = \boldsymbol{K}_b \boldsymbol{x}. \tag{4.7}$$

Trajectories are generated in a receding horizon way with the time horizon of T_h. In order to ensure real-time flight speeds the trajectory planning and execution are performed in parallel using multi-threading in Robot Operating System (ROS). First thread subscribes to the depth image data whenever it arrives, queries the next patch of trajectory from the controller according to the current mode, checks each point in the trajectory for collisions, and appends the valid or an empty trajectory at the end of desired trajectory $\boldsymbol{q}_d^k$. Each patch $\boldsymbol{q}_s^n$ appended is of time T_h. While this thread keeps on appending the trajectories irrespective of where the vehicle is, the other thread keeps sending the way-points from the trajectory $\boldsymbol{q}_d^k$ after every T_s. While querying a potential patch of trajectory $\boldsymbol{q}_s^n$ for collision, if any point is in collision the trajectory is discarded and the system enters the obstacle-avoid mode, while in case of a collision free predicted trajectory it is kept. However, if any point is out of the field of view its simply discarded without any mode change so the planner can keep on querying, in a hope that the trajectory will get into the field of view while the quadrotor maneuvers result in changes of position and orientation of the attached camera.

To improve functionality of the algorithm the potential patch of trajectory sent by the controller is slightly longer than time T_h. This is done to ensure that the collision-free trajectory being appended does not take the quadrotor very close to any obstacle from where getting out becomes impossible under camera's field of view and other vehicle dynamics limitations. This trajectory is therefore generated for $T_h + T_{safety}$ and queried for collisions. In case of no collisions the first half of the trajectory for time T_h is kept and appended. This ensures that while the vehicle maintains a particular mode the collision is detected and planned to be avoided before T_h to T_{safety} seconds. Here T_{safety} is the the safety time and is set smaller

than the time horizon T_h.

4.2   Hybrid System Perspective

$$G(l_0, l_1) = \{ \boldsymbol{x}(k) : \boldsymbol{q}_s(\boldsymbol{x}(k), l_0, n) \in C_{obs}, \ 0 < n < (T_h + T_{safety})/T_s \}$$

l_0

$\boldsymbol{x}(k+1) = \boldsymbol{A}_d\boldsymbol{x}(k) + \boldsymbol{B}_d\boldsymbol{u}(k)$
$\boldsymbol{u}(k) = \boldsymbol{u}_a(\boldsymbol{x}(k))$

$\boldsymbol{x}_{ref} = \boldsymbol{x}_{goal}$

l_1

$\boldsymbol{x}(k+1) = \boldsymbol{A}_d\boldsymbol{x}(k) + \boldsymbol{B}_d\boldsymbol{u}(k)$
$\boldsymbol{u} = \boldsymbol{u}_b(\mathbf{x}(k))$

$\boldsymbol{x}_{ref} = \boldsymbol{x}_{esc}$

$$G(l_1, l_0) = \{ \boldsymbol{x}(k) : \boldsymbol{x}(k) = \boldsymbol{x}_{esc} \}$$

$$G(l_1, l_0) = \{ \boldsymbol{x}(k) : \boldsymbol{x}(k) = \boldsymbol{x}_{goal} \}$$

l_2

$\boldsymbol{x}(k+1) = \boldsymbol{A}_d\boldsymbol{x}(k) + \boldsymbol{B}_d\boldsymbol{u}(k)$
$\boldsymbol{u} = 0$
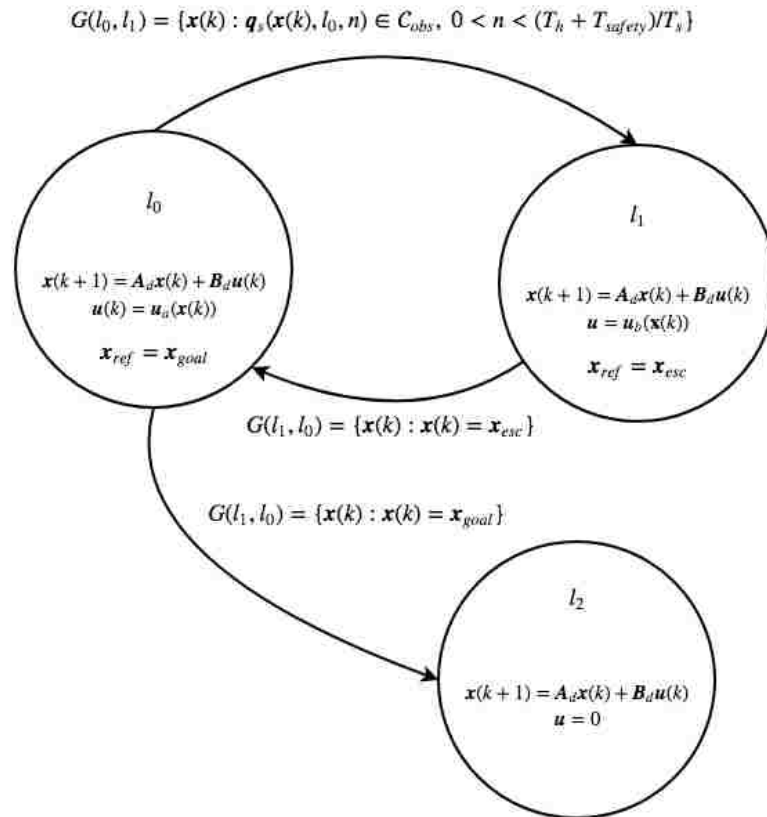
Figure 4.1: Hybrid automaton.

The hybrid automaton $\boldsymbol{H}$ is a collection $\boldsymbol{H} = (\boldsymbol{L}, \boldsymbol{X}, \boldsymbol{Init}, \boldsymbol{f}, Inv, \boldsymbol{E}, G, R)$ [30], where

1. $\boldsymbol{L}$ is a set of discrete variables (or modes)

2. $\boldsymbol{X} \in \mathbb{R}^6$ is a set of state variables for a quad-rotor dynamical system

3. **$Init \subseteq L \times X$** is a set of initial states

4. **f** $= \{f_1, f_2, f_3\}$ is a set of vector fields or systems dynamics for each $l \in L$

5. *Inv* assigns to each $l \in L$ an invariant set *i.e. $Inv(l) \subset$ **x***

6. **E** is the collection of discrete transitions

7. G assigns to each **e** $= (l, l') \in$ **E** a guard

8. R assigns to each **e** $= (l, l') \in$ **E** and **x** $\in$ **X** a reset condition

The system undergoes two different types of manuevers defined by the motion primitives at each state **x**(k) depending on the current discrete mode l_k. [31] Let the allowed motion primitives for any state **x**(k) be a set $\mathcal{U}^p(\boldsymbol{x}(k)) = \{\boldsymbol{u}_a, \boldsymbol{u}_b\}$. For each sampling time interval (*i.e.* from **x**(k) to **x**$(k+1)$) the system experiences an input $\boldsymbol{u} \in \mathcal{U}^p$ related to the state **x**(k) by Equation 4.6. The hybrid automaton comprises of 2 modes.

Mode 0

Mode 0 (l_0) is responsible for taking the quad-rotor to the goal position with an objective to minimize the state error as well as the energy. In other words, the controller for this mode is defined as:

$$\boldsymbol{u}_a = \{\boldsymbol{u} : min_u \sum (\boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{u}^T \boldsymbol{R} \boldsymbol{u})\},$$

where

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 \end{bmatrix},$$

$$\boldsymbol{R} = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{bmatrix}.$$

Two of the guard associated with this mode are $G(l_0, l_1)$, $G(l_0, l_2)$. $G(l_0, l_1)$ is enabled when a collision is detected in trajectory $\boldsymbol{q}_s^n$ for any $0 < n < (T_h + T_{safety})/T_s$ *i.e.* the collision in some future time becomes inevitable if the quadrotor stays in this mode. $G(l_0, l_2)$ is enabled when the quadrotor reaches goal configuration q_{goal} at 0 velocity in all three dimensions as mentioned in Section 1.4. This final goal state $\boldsymbol{x}_{goal}$ corresponds to the $\boldsymbol{q}_{goal}$ given the velocities in three dimensions. Therefore, there is an easy mapping between a state $\boldsymbol{x}$ and the configuration at that state $\boldsymbol{q}$.

Hence, the objective of controller in this mode is to take the quadrotor to its destination $y - z$ plane with zero final velocity while minimizing the energy/fuel consumption. However, in case of an obstacle on the way, the system goes to mode 2 where the objective is primarily to deviate quickly without caring much about the energy minimization.

Mode 1

Mode 1 (l_1) is responsible for taking the quad-rotor to the temporary escape config-uration with an objective to minimize more the state error rather than the energy. In other words, the controller for this mode is defined as:

$$\boldsymbol{u}_b = \{\boldsymbol{u} : min_u \sum (\boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{u}^T \boldsymbol{R} \boldsymbol{u})\},$$

where

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 \end{bmatrix},$$

$$\boldsymbol{R} = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}.$$

Two of the guard associated with this mode are $G(l_0, l_1)$, $G(l_1, l_0)$. $G(l_0, l_1)$ is explained in the above section. $G(l_1, l_0)$ is enabled when the quadrotor reaches escape configuration $\boldsymbol{q}_{esc}$ at 0 velocity in all three dimensions. The perception algorithm makes sure that before switching to this mode from mode 0, it performs the check in four predefined directions from the predicted collision point and tries to find a collision free point in 3D that is closest to the collision point in $y - z$ plane corresponding to the collision point. The given 3D point corresponds to the configuration in $\mathcal{C}$ because θ is assumed to be 0. This configuration can be easily converted to state vector $\boldsymbol{x}$ by appending 0s for the velocity terms.

Hence, the objective of controller in this mode is to take the quadrotor to an escape point with zero final velocity to have a clearer view of what is beyond the obstacle. Less weights on the energy is inspired from the fact that in case of a sudden interruption in vechicle's way, it must take measures to quickly maneuver away to a point from where it can potentially see its destination clearly.

4.3 Proof of Hybrid System's Stability

Since the trajectory generation follows either of the two pre-computed LQR feedback laws, the concern of system's stability while mode switching is of great importance. As mentioned before the first feedback law $\boldsymbol{u}_a$ tries to take the system to a global goal while the second feedback law $\boldsymbol{u}_b$ takes the system to a temporary escape point as mentioned in Equations 4.6. Two different controls objectives leads to two Lyapunov functions. Figures 4.2 and 4.3 show these set of Lyapunov functions for a particular case. Here $\boldsymbol{x}_{ref}$ and $\boldsymbol{x}_{esc}$ are the equilibrium points of first and second Lyapunov functions respectively.

An interesting fact about combining real-time perception and hybrid control theory is that even though the first Lyapunov function remains the same for the whole maneuver, the second Lyapunov function changes its location *i.e.* it slides relative to the first Lyapunov function based on where the obstacle is encountered. Every time the obstacle is encountered at a different location the overlapping regions of the two Lyapunov functions change. Therefore, we will prove that under certain assumptions on the regions in Lyapunov functions the switching is always stable. These assumptions include that the obstacle is always closer than the global goal and that the switching occurs before the quadrotor reaches obstacle location. For instance, in a particular case of Figure 4.3, $\boldsymbol{R}_e^i$, $i \in \{1, 2\}$ refers to the region where the system cannot enter practically while staying in the corresponding mode. Since the obstacle is located at 4m, the system is assumed to enter mode 1, before entering region $\boldsymbol{R}_e^2$.

Lets take the second order subsystem for x dimension,

$$\boldsymbol{x} = [x(k) \ x(k+1)]^T.$$

The corresponding Lyapunov functions for each control law are given as:

$$V_1(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{x}_{ref})^T \boldsymbol{P}_1 (\boldsymbol{x} - \boldsymbol{x}_{ref}), \tag{4.8}$$

$$V_2(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{x}_{esc})^T \boldsymbol{P}_2 (\boldsymbol{x} - \boldsymbol{x}_{esc}). \tag{4.9}$$
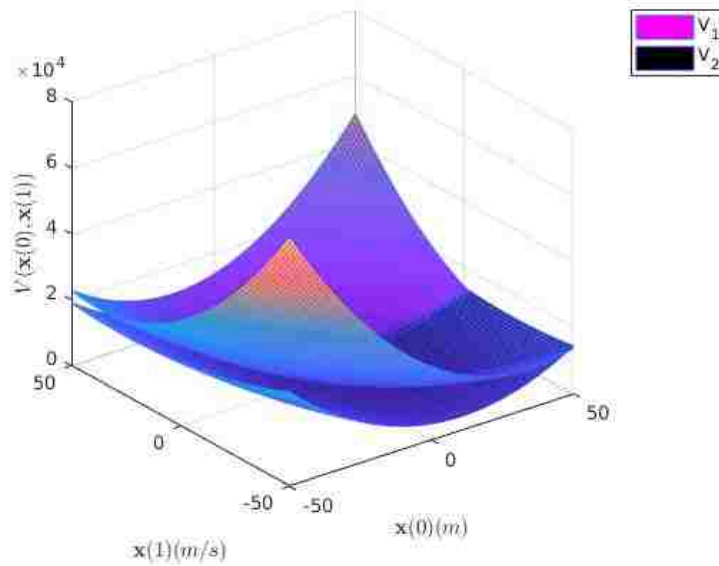
Figure 4.2: Lyapunov function For $\boldsymbol{x}_{ref} = \begin{bmatrix} 10 & 0 \end{bmatrix}$ and $\boldsymbol{x}_{esc} = \begin{bmatrix} 4 & 0 \end{bmatrix}$.
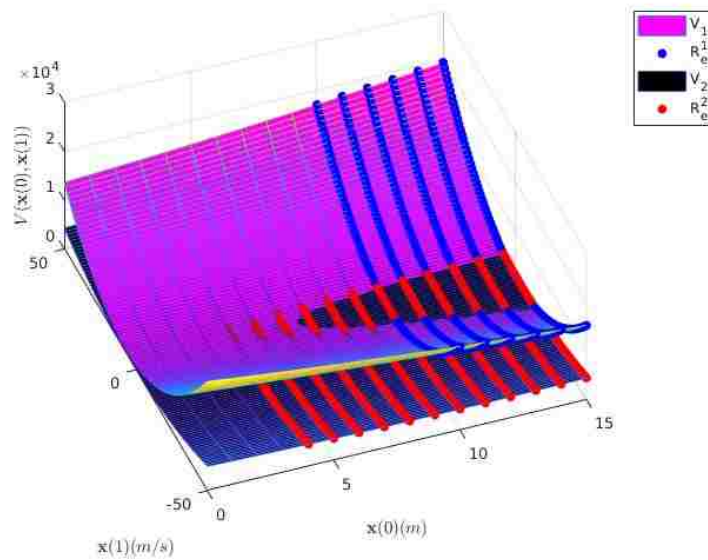
Figure 4.3: Lyapunov function (Zoomed-In) For $\boldsymbol{x}_{ref} = \begin{bmatrix} 10 & 0 \end{bmatrix}$ and $\boldsymbol{x}_{esc} = \begin{bmatrix} 4 & 0 \end{bmatrix}$.

Here $\boldsymbol{x}_{ref} \in \boldsymbol{x}$ and $\boldsymbol{x}_{esc} \in \boldsymbol{x}$ are the global reference state and the obstacle state in x dimension respectively. $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ are found by solving the Lyapunov equation for

discrete time systems:

$$\boldsymbol{A}^T \boldsymbol{P}_i \boldsymbol{A} - \boldsymbol{P}_i + \boldsymbol{Q} = 0 \quad i \in 0, 1. \tag{4.10}$$

According to Rayleigh-Ritz inequality for symmetric matrices,

$$\lambda_{1(min)} ||\boldsymbol{x} - \boldsymbol{x}_{ref}|| \leq V_1(\boldsymbol{x}) \leq \lambda_{1(max)} ||\boldsymbol{x} - \boldsymbol{x}_{ref}||, \tag{4.11}$$

$$\lambda_{2(min)} ||\boldsymbol{x} - \boldsymbol{x}_{esc}|| \leq V_2(\boldsymbol{x}) \leq \lambda_{2(max)} ||\boldsymbol{x} - \boldsymbol{x}_{esc}||. \tag{4.12}$$

The system is undergoing stable switching if ([47])

$$V_1(\boldsymbol{x}_s) - V_2(\boldsymbol{x}_s) > 0, \tag{4.13}$$

or

$$\lambda_1 ||\boldsymbol{x}_s - \boldsymbol{x}_{ref}|| - \lambda_2 ||\boldsymbol{x}_s - \boldsymbol{x}_{esc}|| > 0, \tag{4.14}$$

where $\boldsymbol{x}_s$ is any state at which switching occurs [29] [30]. The domain $D_s : \mathbb{R}^2 \to \mathbb{R}^2$ of $\boldsymbol{x}$ where the switching is possible to occur is given as:

$$D_s(\boldsymbol{x}) = \{\boldsymbol{x} \mid \boldsymbol{x}(0) < \boldsymbol{x}_{esc}(0) < \boldsymbol{x}_{ref}(0)\}. \tag{4.15}$$

It can be noted that our system has repeated eigen values for both modes, *i.e.* $\lambda_{1(min)} = \lambda_{1(max)} = \lambda_1 = 3.3781$ and $\lambda_{2(min)} = \lambda_{2(max)} = \lambda_2 = 1.6640$. We have to find out that within our domain of $\boldsymbol{x}_s$, whether the switching is stable. Expanding Equation 4.14,

$$(\lambda_1 - \lambda_2) ||\boldsymbol{x}_s|| + \lambda_1 ||\boldsymbol{x}_{ref}|| - \lambda_2 ||\boldsymbol{x}_{esc}||$$

$$-2\lambda_1 (\boldsymbol{x}_s(0)\boldsymbol{x}_{ref}(0) + \boldsymbol{x}_s(1)\boldsymbol{x}_{ref}(1))$$

$$+2\lambda_2 (\boldsymbol{x}_s(0)\boldsymbol{x}_{esc}(0) + \boldsymbol{x}_s(1)\boldsymbol{x}_{esc}(1)) > 0.$$

Since the two optimas have zero velocity, we can put them equal to zero,

$$(\lambda_1 - \lambda_2) ||\boldsymbol{x}_s|| + \lambda_1 \boldsymbol{x}_{ref}(0)^2 - \lambda_2 \boldsymbol{x}_{esc}(0)^2$$

$$-2\lambda_1 \boldsymbol{x}_s(0)\boldsymbol{x}_{ref}(0) + 2\lambda_2 \boldsymbol{x}_s(0)\boldsymbol{x}_{esc}(0) > 0.$$

Adding and subtracting $\lambda_1 \boldsymbol{x}_s(0)^2 - \lambda_2 \boldsymbol{x}_s(0)^2$ on both sides:

$$(\lambda_1 - \lambda_2) ||\boldsymbol{x}_s|| + \lambda_1 (\boldsymbol{x}_{ref}(0)^2 - 2\boldsymbol{x}_s(0)\boldsymbol{x}_{ref}(0) + \boldsymbol{x}_s(0)^2)$$

$$-\lambda_2 (\boldsymbol{x}_{esc}(0)^2 - 2\boldsymbol{x}_s(0)\boldsymbol{x}_{esc}(0) + \boldsymbol{x}_s(0)^2)$$

$$-(\lambda_1 \boldsymbol{x}_s(0)^2 - \lambda_2 \boldsymbol{x}_s(0)^2) > 0.$$

Chapter 4. Control Strategy

Simplifying further leads to:

$$(\lambda_1 - \lambda_2)\boldsymbol{x}_s(1)^2 + \lambda_1(\boldsymbol{x}_{ref}(0) - \boldsymbol{x}_s(0))^2 - \lambda_2(\boldsymbol{x}_{esc}(0) - \boldsymbol{x}_s(0))^2 > 0.$$

It should be noted that $\lambda_1 > \lambda_2$. Consequently, the first term is always greater than 0. The second and the third terms are also always greater than 0 if $\boldsymbol{x}_{ref}(0) > \boldsymbol{x}_{esc}(0) > \boldsymbol{x}_s(0)$ (*i.e.* if $\boldsymbol{x}_s \in D_s(\boldsymbol{x})$). Therefore, the hybrid system's stability is guaranteed in x-axis.

Chapter 5

Verification and Applications

5.1 Tracking Results

The experiments could not be performed in the VICON testbed because of the scale of these experiments. Exploiting the system's capabilities of computing visual-inertial odometry (VIO) from the on-board sensors provides a viable solution. We are using a single forward-facing camera for all vision related tasks. Before moving further with the experiments, the trajectory tracking capabilities have to be evaluated. This is done based on its performance to track a figure eight trajectory which can be a difficult trajectory to track if the system is unstable. The system is first evaluated in the motion capture system before moving on to using VIO. The trajectory is generated utilizing the concept of the lissajous figures. Combination of sine waves are used to compute the figure eight trajectory. Figure 5.1 (left) shows the results of figure eight trajectory tracking inside the VICON motion capture system. The performance seems better in this setup for obvious reasons but ZED camera brings the state-of-the-art stereo SLAM technology which makes the trajectory tracking stable even though the camera is mounted in the forward facing manner which seems quite reasonable. Figure 5.1 (right) shows the trajectory tracking in on-board vision setup.
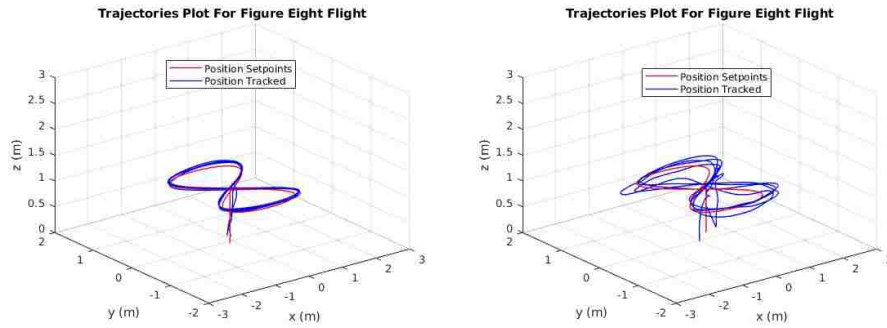
Figure 5.1: Figure eight trajectory tracking with motion capture (left) and VIO(right).

5.2 Flight Performance Through Square Targets

Initial project for the performance verification of this platform was inspired by the problem statement of a drone racing competition held at IEEE Conference on Intelligent Robot Systems (IROS) 2017.

5.2.1 Overview

In recent years, several efforts are made to address the problem of vision-based navigation. [33] describes an approach to solve this problem using RGB-D camera in the presence of some known markers. This paper considers low light conditions but does not take care of the problem if there are no markers in the environment. Some authors describe the development of test bed for vision based navigation in GPS denied environment like [2]. Computing odometry using vision tools is one of the main aspect in camera based navigation. Among the good contributions in this area are [37] [38] which demonstrate the use of visual and inertial measurement tools for localization and scene reconstruction. [35] also describes an approach to obtain visual odometry in such an environment. Moreover, [36] uses vision based approach in an interesting way to navigate along the forest trails by using deep learning on

the monocular images. The problem is to build a platform that has the ability to maneuver aggressively while passing through targets autonomously. While the existing literature and experimentation serve as good contributions in vision based navigation, there is a need to develop a comprehensive platform for the execution of a complete autonomous flight in such a scenario. It can, however, be extended for other vision based navigation problems. A similar approach, to the one that is adopted, is presented in [46]. The example problem is inspired by the fact that in usual situations of fire and rescue, there is a need for a platform that can navigate quickly passing through the targets or avoiding any obstacles on its way. The problem is formulated as follows. Let $q \in \mathcal{C}$ be a point in configuration space $\mathcal{C}$ of the Quadrotor. $\mathcal{O} \subset \mathcal{W}$ refers to the workspace obstacle, where $\mathcal{W} \in \mathbb{R}^{3 \times i}$ for any integer $i > 0$. The workspace Quadrotor $A(q) \subset \mathcal{W}$ is expressed as a set of all the points occupied by the quadrotor while maintaining a certain state q. The configuration for which the quadrotor will collide with the obstacle is given as $\mathcal{C}_{obs} = \{q \in \mathcal{C} | A(q) \cap \mathcal{O} \neq \phi\}$. In order to ensure collision free maneuver, the allowed configuration is $\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obs}$.

The problem statement can be divided into path planning and following problems. Former is to come up with a mapping $m : [0,1] \rightarrow \mathcal{C}_{free}$ such that $m(.) = p \in \mathcal{C}_{free}$ at some input in $[0,1)$, where p is the center of the target. Further, a position controller is responsible for efficient path following. Moreover, it is assumed that the locations of the targets are not completely known so the paths to take the quadrotor from the target's center cannot be computed offline.

The quadrotor is assumed to be able to follow the desired waypoints generated by the software. It essentially follows a path to reach a point $n \in \mathcal{C}_{free}$ at which it can see the full target in front of it. Since this path is considered obstacle-free, the quadrotor does not have to avoid any obstacle to reach there. Once it reaches the point n, it recognizes the target, and its center point using ZED stereo camera images. Using this information, it aligns itself in front of the center point, to complete its maneuver through the target.

5.2.2   Results

Processing in depth space is performed in order to fully visualize the square targets within a certain range. Waypoints are generated based on the location of the estimated center of the target in depth space. The perception function takes over when the quadrotor reaches the vicinity of the target from where the stereo camera can see the square target. The quadrotor is assumed to stop to see the target so that no transformation is required from inertial to image coordinates. Also, the camera image plane is assumed to be parallel to the target before the perception function takes over. Moreover, the waypoints are generated in the vehicle body coordinates. The depth image is first thresholded for a certain range in front of the camera in which the obstacle is supposed to appear. This results in a binary image. However, the image appears to be a bit noisy. Certain techniques are then used exploiting OpenCV functions in C++ to overcome the noise issue. Two very famous morphological operations are performed. Pixel dilation is first used to fill small holes in the noisy image followed by erosion to remove background false ones or noise. After performing such operations the binary image is then checked for contours particularly forming rectangles in the image. opencv_apps is a ROS package which typically performs similar operations in the ROS framework for convenience. It takes a binary image in the form of ROS message for images *i.e.* of type sensor_msgs/Image and outputs a message on the topic opencv_apps/RotatedRectArrayStamped. This output message contains information about all the rotated rectangles in the image from where it is easy to identify the rectangle corresponding to the target in view. The estimated location of the center point of the target in the image is then calculated based on the information from the opencv_apps output message. This output message contains information about all the rotated rectangles in the image from where it is easy to identify the rectangle corresponding to the target in view. The estimated location of the center point of the target in the image is then calculated based on the information from the opencv_apps output message. A proportional controller in

49

the image space is used to generate motions in four directions. The image center being the reference, the distance error in the image pixel coordinates generated in the Y-axis of the image commands up-down movements while the side-ways movements are caused by the pixel distance error in X-axis of the image. The quadrotor keeps on performing such maneuvers until the camera image center aligns with the estimated center point of the target with some success radius Δ. At this point the quadrotor performs forward maneuver to pass through the target. Figure 5.2 shows the flow diagram of the working of the system Figure 5.3 shows one instance of a maneuver.
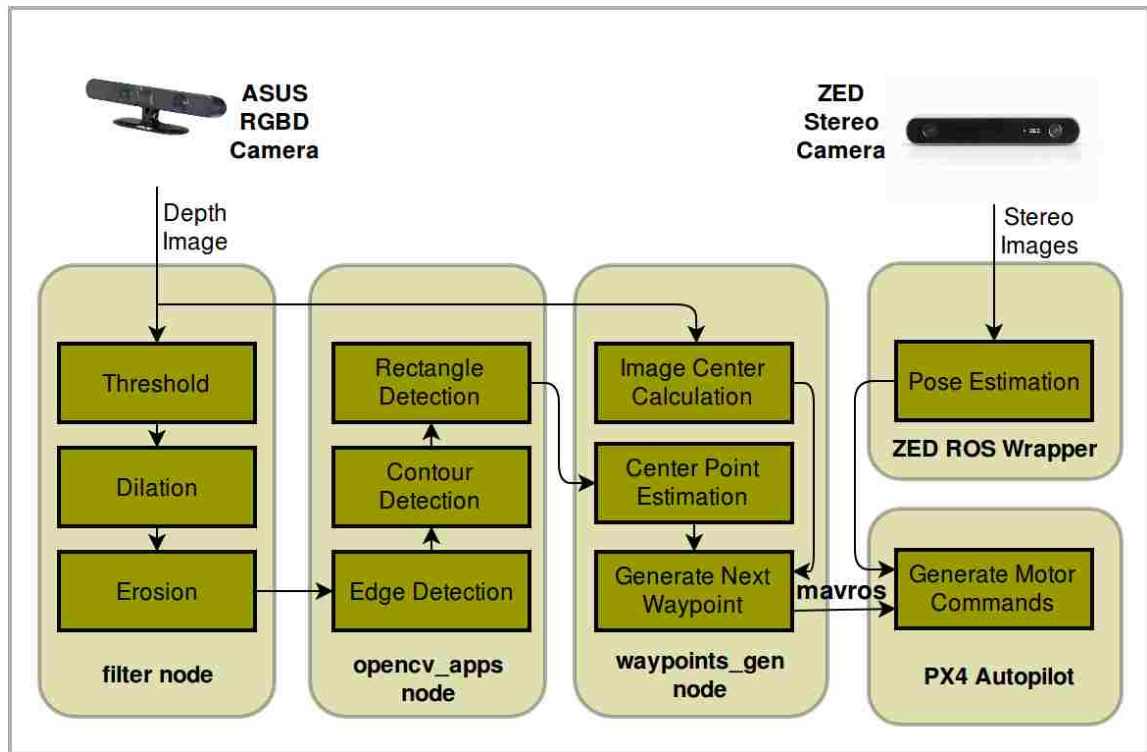

Figure 5.2: Flow diagram of the algorithm.

The quadrotor flies from the start configuration to a stationary configuration from where it can see the target. The proportional controller then performs maneuvers to align the image center with the center of the target in view to pass through.
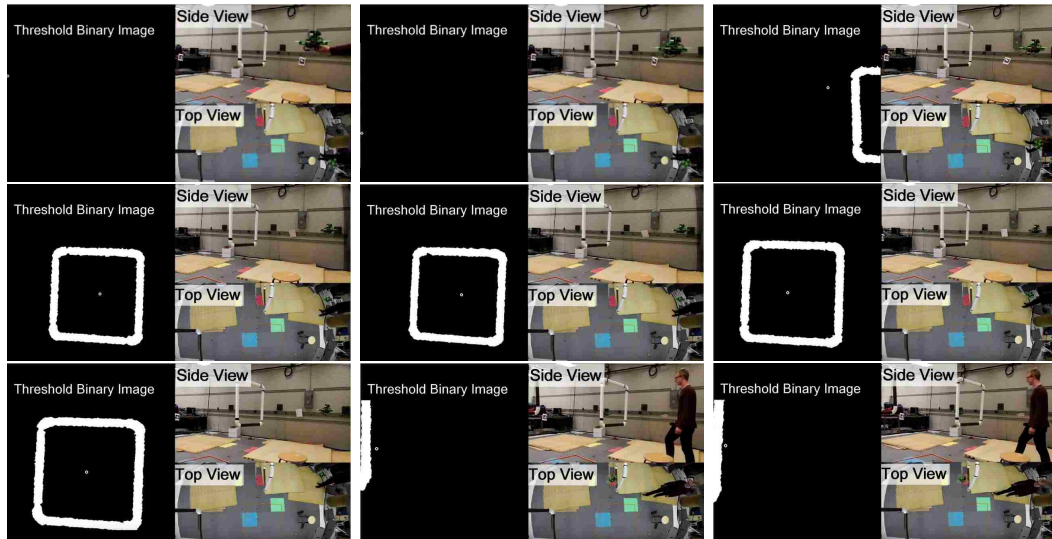
Figure 5.3: Snapshots to show one instance of manuever through a square target.

5.3 Simulation

The simulation of the set up was performed in Gazebo. Gazebo is a physics simulator which is good in simulating the real world parameters. Gazebo is fully compatible with ROS providing a functional package for ROS plugins. Besides simulating real-world environment Gazebo can easily interface and talk to ROS through publisher/subscriber or service/client protocol. Gazebo models are stored in urdf or sdf file formats which are popular formats for physics models. RotorS [32] is a ROS package providing models for Iris, AscTech Hummingbird, Pelican, Firefly and many other quadrotor UAVs. The algorithm is simulated on Iris quadrotor with a forward facing depth camera. Gazebo provides the flexibility to create custom environment models as well as customize the existing ones. Several different types of obstacles can be placed at our desired locations according to the problem. The setup includes the RotorsS package as a UAV simulator for Gazebo interacting with the ROS nodes programmed in C++ as described in the previous sections. These ROS nodes subscribe (receive) information about the environment and robot states and
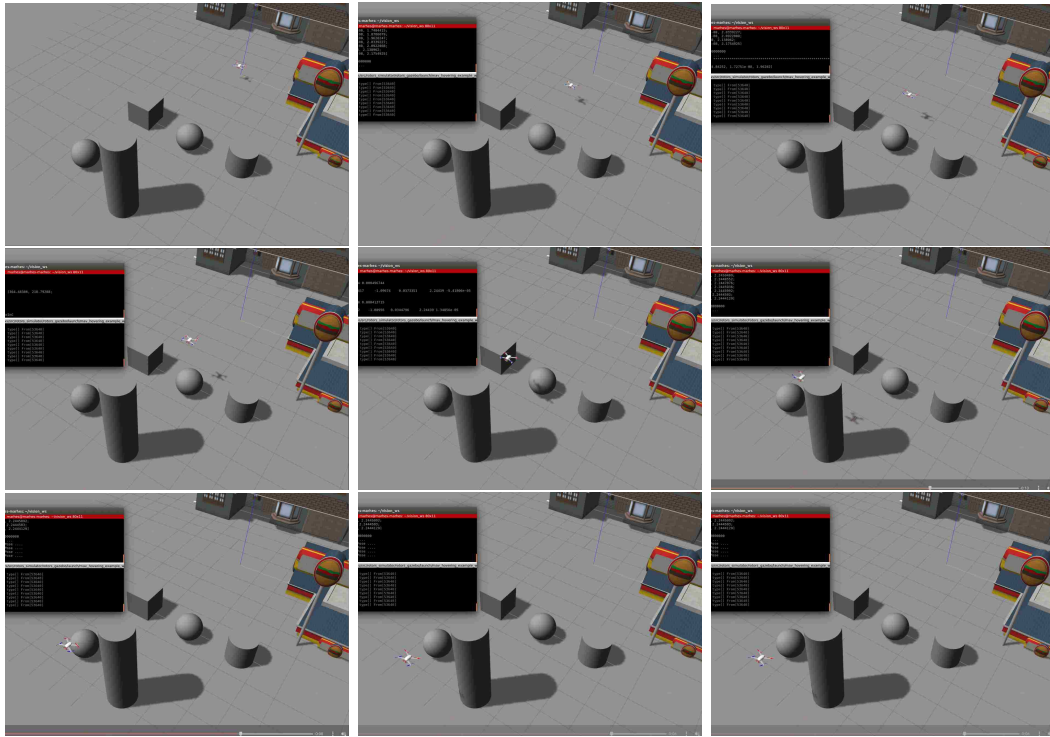
Figure 5.4: Snapshots to show one instance of manuever in an example workspace in Gazebo.

publish (send) the desired commands to the quadrotor like a real setup. Figure 5.8 shows how the functions are setup to talk to each other. A ROS node is written as an interface to communicate with sensors and the autopilot. C++ functions are setup to take those images and odometry messages to manipulate and form trajectories. *traj_gen* function is responsible for taking the trajectories from the LQR controller and send each point for collision check. *waypoint2collision* is setup to check a group of waypoints for collision by projecting the artificial shield associated with them in the depth space and checking them for collision as mentioned in the previous chapters. If collision is detected an escape point is found using *find_escape* function. The waypoints from the final trajectory are sent to the autopilot at the sampling time of T_s.

Different types of obstacles are placed at random locations to introduce different

scenarios for the quadrotor. Figure 5.4 shows one instance of such a setup where the robot has to avoid two different kinds of obstacles randomly placed in the workspace. Figure 5.5 and 5.6 shows the step by step process of trajectory generation. These results are derived from the Gazebo simulation. Each figure shows the trajectory sent by the LQR function for a fixed horizon as described in Section 4.1. In case of no collision the first half of the trajectory is kept while the second part is discarded to prevent the robot getting too close to any potential obstacle appearing immediately in front of the generated trajectory. However, in case of an obstacle the trajectory generated is discarded and an escape is found in four directions *i.e.* up, down, right and left. The closest escape is then chosen and the controller objective is changed to favor fast avoidance of the obstacle rather than saving energy. Similar process goes on with the required switching according to the scenario until the quadrotor escapes and reaches its final destination. The complete final trajectory commanded to the quadrotor to follow is shown in Figure 5.7.
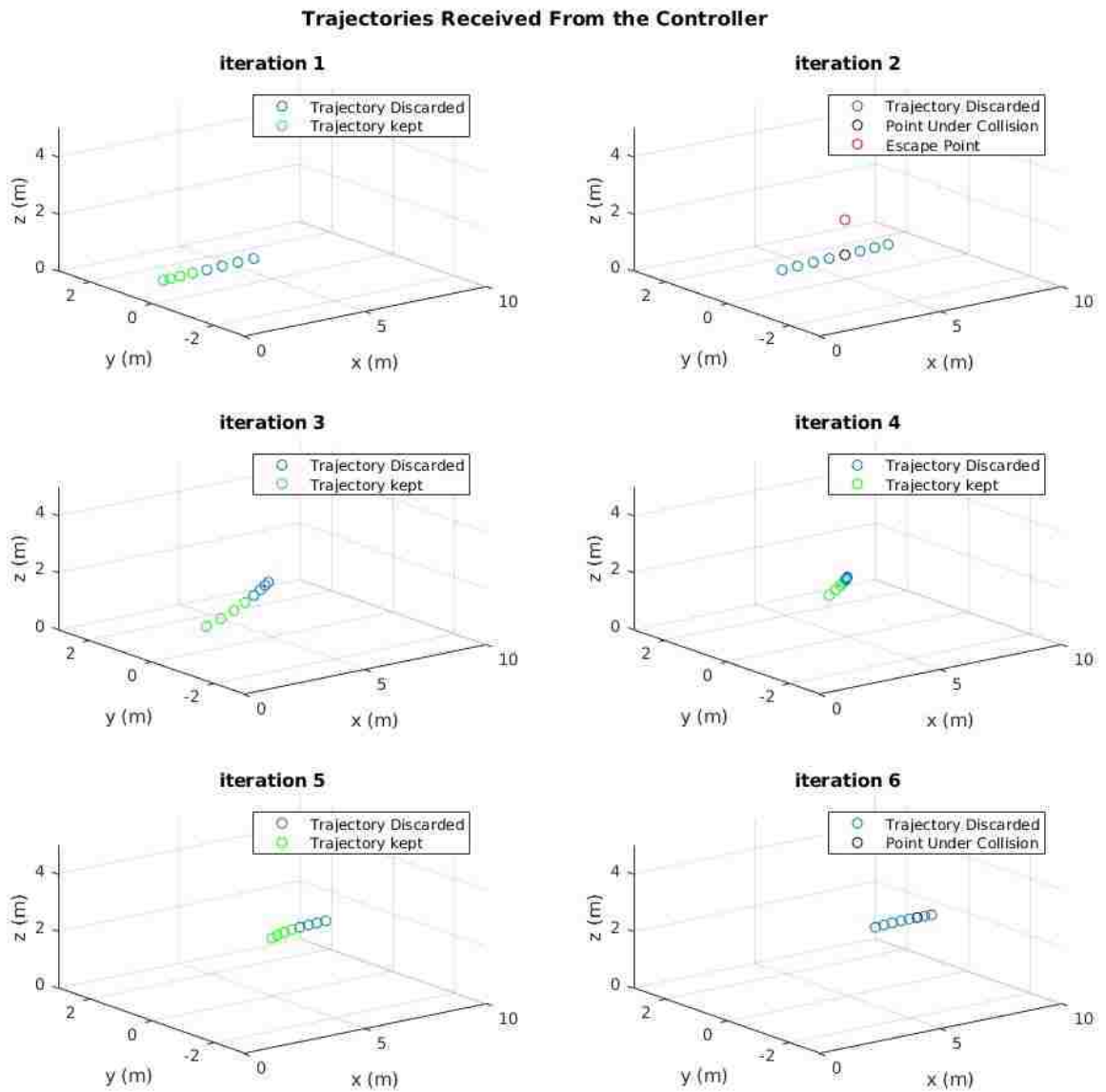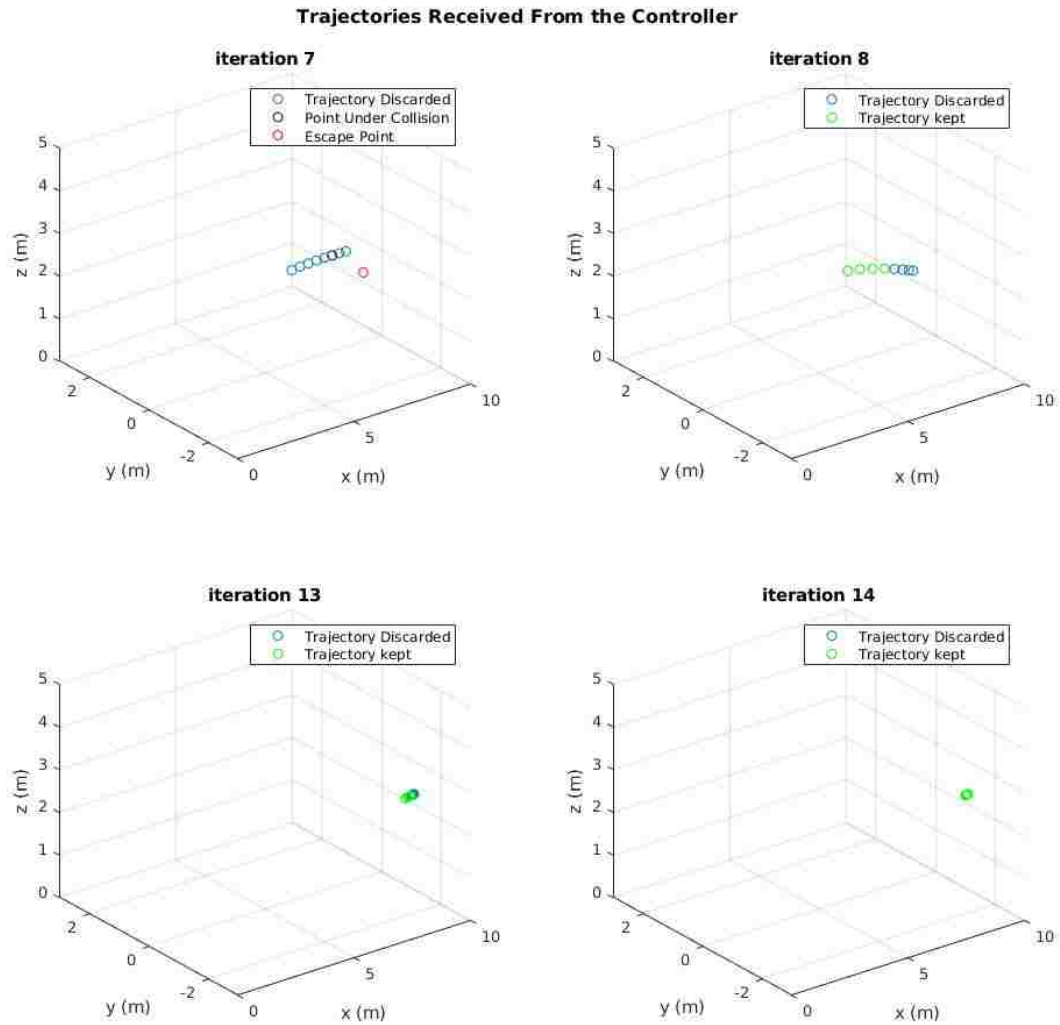
Figure 5.5: Trajectory generation (a).

Figure 5.6: Trajectory generation (b).

5.4   Implementation Results

The experiments are performed on a real quadrotor made out of Luminier QAV 250 frame, labeled as LoboDrone v2.0 as shown in Figure 2.6. The first set of experiments are performed in a 5x5 feet area with a single obstacle each time. Two different types
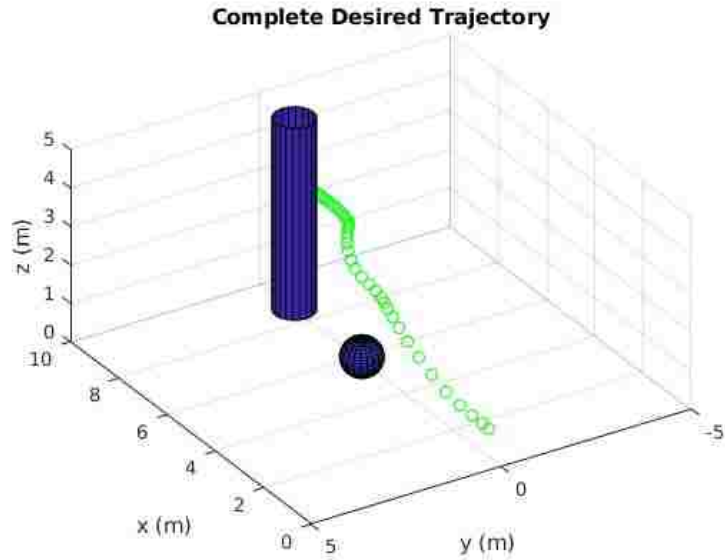
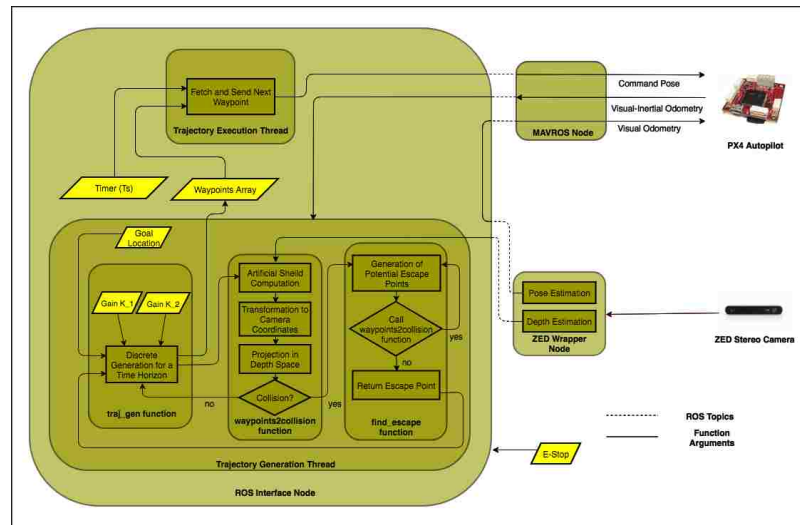Figure 5.7: Final trajectory setpoints (Simulation).

Figure 5.8: Simplified flow chart of the algorithm (Hybrid trajectory control approach).

of obstacles are placed in its way to observe the maneuvers in different scenarios. Figure 5.9 shows the plots for the two scenarios. In the first scenario the quadrotor

is expected to pass around the obstacle due to its thin structure while in the second one it is supposed to pass over an obstacle because of its low height. The setup was then tested for two consecutive obstacles in a larger workspace. Figure 5.10 shows the desired trajectory similar to what is presented for the Gazebo simulations (Figure 5.7). The algorithm worked well in the practical implementation following the expectations from the simulations.
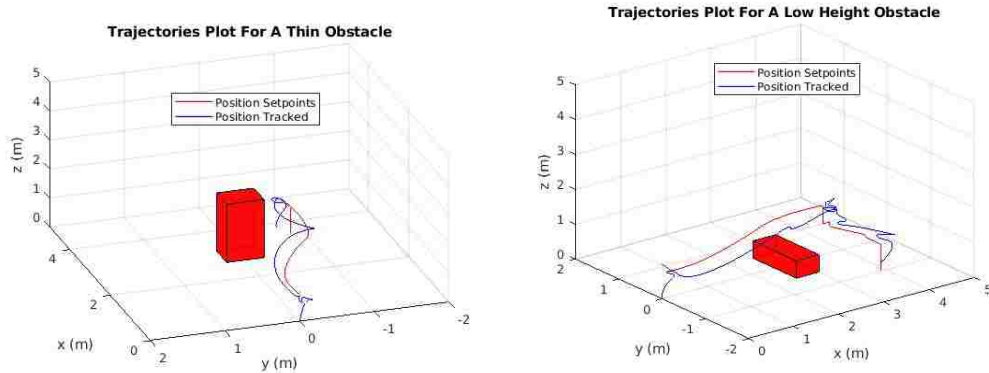

Figure 5.9: Trajectory plot for low height and tall obstacles (Implementation).

5.5   Artificial Potential Fields (APF) Approach

The experiments are also performed for obstacle avoidance using similar hardware to LoboDrone v1.0 to demonstrate the strengths and weaknesses of the proposed approaches. These experiments are done as a part of an undergraduate senior design project [42] at the MARHES Laboratory. Artificial potential fields method serve as a 'reactive' approach to avoid obstacles. The obstacle is simulated as a collection of point charges adding a repulsive force on the robot. Moreover, the final goal adds a positive vanishing potential to the robot. As a result the quadrotor is able to fly with the flow of potentials. It is treated as a point particle under the influence of the net charge. The main goal is to go towards the region of lowest net charge which causes the quadrotor to maneuver towards the goal. The attractive force vanishes close to
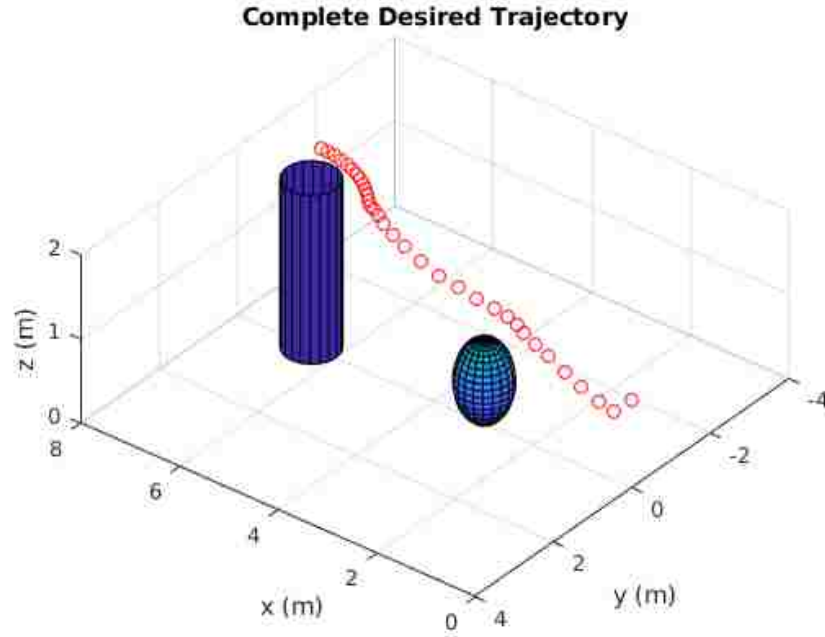
Figure 5.10: Final trajectory setpoints (Implementation).

the goal. This forms the parabolic well as an attractor. This attractive potential is given as :

$$U_{att} = \frac{1}{2}\zeta||\boldsymbol{q} - \boldsymbol{q}_{goal}||.$$

The force exerted on the quadrotor is represented as a negative gradient of potential as follows:

$$\boldsymbol{F}_{att}(\boldsymbol{q}) = -\Delta U(\boldsymbol{q}),$$

$$\boldsymbol{F}_{att}(\boldsymbol{q}) = -\zeta(\boldsymbol{q} - \boldsymbol{q}_{goal}).$$

The obstacle, however exerts a similar but repulsive force on the point particle. This repulsive potential is only assumed to exert a force when the quadrotor is within its neighborhood. The force goes to ∞ at the edges of the obstacle. This can be

summed up as:

$$
U_{rep}(\boldsymbol{q}) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{\|\boldsymbol{q}-\boldsymbol{q}_{obs}\|} - \frac{1}{\rho_0}\right)^2, & \text{if } \|\boldsymbol{q} - \boldsymbol{q}_{obs}\| < \rho_0. \\ 0, & \text{otherwise.} \end{cases}
$$

And,

$$
\boldsymbol{F}_{rep}(\boldsymbol{q}) = \begin{cases} \eta\left(\frac{1}{\|\boldsymbol{q}-\boldsymbol{q}_{obs}\|} - \frac{1}{\rho_0}\right)\left(\frac{1}{\|\boldsymbol{q}-\boldsymbol{q}_{obs}\|}\right)^2 \frac{\boldsymbol{q}-\boldsymbol{q}_{obs}}{\|\boldsymbol{q}-\boldsymbol{q}_{obs}\|}, & \text{if } \|\boldsymbol{q} - \boldsymbol{q}_{obs}\| < \rho_0. \\ 0, & \text{otherwise.} \end{cases}
$$

While having the same limited environment knowledge as with the previous experiments, artificial potential field method is implemented in real-time. The quadrotor only knows the goal position prior to flight. During flight the position estimate is provided through a ZED stereo camera and an IMU. The waypoints are updated at 2Hz to be sent to the low level controller to follow. In other words, given a resultant potential at a particular position, the waypoints are generated in a particular direction until they are updated again after 0.5 seconds. This process is shown in Figure 5.11. Small trajectory sections show the path anticipated from a particular position. Updating them at 2 Hz forms a complete desired trajectory. Figure 5.12 shows the desired and followed trajectories for the quadrotor.
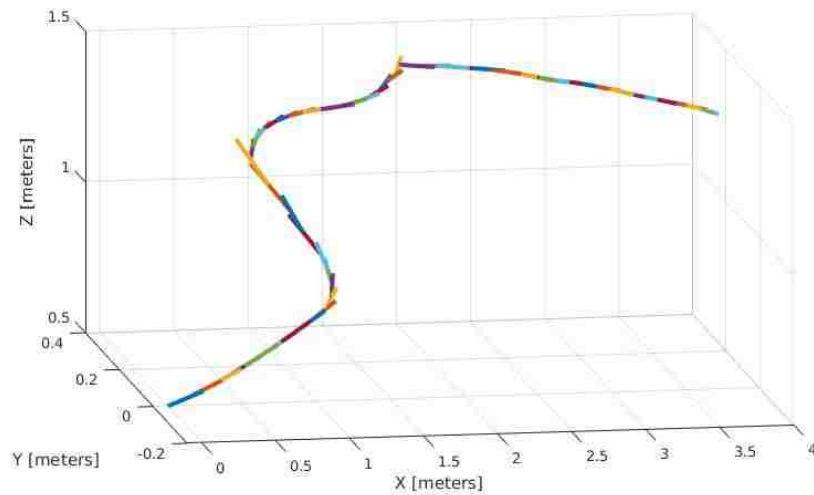

Figure 5.11: Formation of desired 3D trajectory (Artificial potential fields approach).
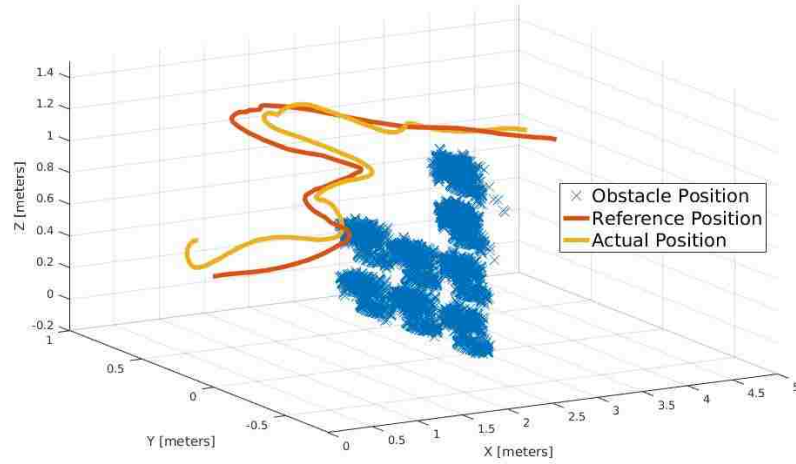
Figure 5.12: Complete 3D trajectory (Artificial potential fields approach).

5.6 Comparison Between APF and Hybrid Trajectory Generation Approaches

Artificial potential fields is a very old method but is established for many robotics path planning applications. This method outperforms many methods and is still considered a competitive method depending on the nature of applications. However, every method have some pros and cons associated with them. They are summarized in Table 5.6. Both the methods in comparison are not complete. Artificial potential fields have inherent drawback of the robot getting stuck in local minima if it encounters one. In that case a global solution may exists but the algorithm is unable to identify it. A similar problem is experienced by the proposed strategy. However, it is because of two reasons *i.e.* limited field of view and limited number of motion primitives. The quadrotor gets stuck if it gets so close to the obstacle that it is unable to find a collision free trajectory within its field of view. This problem with the limited field of view is quite straightforward to relate but it cannot be solved with both the techniques because of the hardware resource limitations. Figure 5.13 shows

how the limited number of motion primitives affect the maneuver. The quadrotor might encounter a situation where none of the motion primitives is able to form a collision free trajectory to the escape point. In other words speed of the algorithm is compromised with completeness. This is referred to as restricted reachability space in the table because the reachability space depends on the allowed maneuvers at a particular configuration. Rapidly-exploring random trees (RRT) is another famous motion planning algorithm. It guarantees resolution completeness but it might take longer time to find collision free paths in a depth map because it explores in random directions. Moreover, it also involves more collision checks because the search for escape is completely random. This makes it computationally more expensive. As described above, the presence of a forward facing stereo camera pose a limited field of view problem to RRT method as well.

Another advantage common to both the hybrid control and artificial potential fields methods is that the trajectories are dynamically feasible. Most of these differences can be inferred by comparing Figures 5.12 and 5.9. While the hybrid controller tends to form trajectories that are closer to the obstacle, the potential fields favors trajectories far away from the obstacle depending upon the attractor and repellor gains. It also takes more time to reach the goal than the hybrid control approach in which we have full control over the speed of individual maneuvers. Moreover, one of the main drawbacks of using potential field approach is that it requires to estimate the obstacle location and distance to quantify the charge on the robot. The sampling based methods and the hybrid trajectory control method however, solves this problem by performing collision checks in depth space and re-planning the trajectories in future when necessary. Also, the potential fields method requires to take into consideration all the workspace obstacles or atleast the obstacles in the field of view to compute the net charges on the robot. In contrast the hybrid trajectory approach can perform checks on a limited set of workspace and does not need to know the location of all the obstacles unless they intercept the projected trajectory.
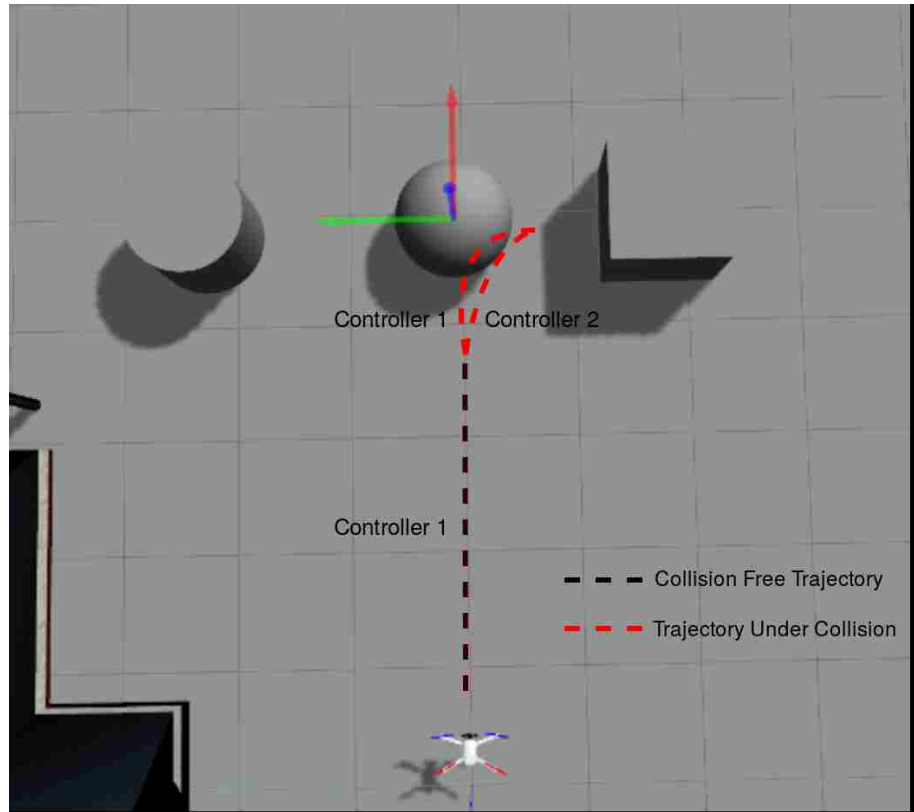
Figure 5.13: One instance of quadrotor getting stuck due to limited motion primitives (Hybrid trajectory control approach)

	Potential fields method	Hybrid control approach
Dynamically feasible trajectories	✓	✓
Unrestricted reachability space	✓	✗
Completeness	✗	✗
Fast	✗	✓
Does not require estimating the obstacles geometries	✗	✓
Can work in limited environment knowledge (Does not need to know every obstacle around)	✗	✓

Table 5.1: Comparison.

Chapter 6

Conclusion

The thesis explained the development of a platform capable of vision-based autonomous navigation in cluttered environments. The development of this platform involves a lot of steps. There was a need for hardware and software platforms and algorithms to accomplish on-board processing, on-board sensing, agility and finally the autonomy of a quadrotor UAV platform. In the work supported under Army Research Lab's Micro Autonomous Systems and Technology Collaborative Alliance (ARL-MAST) and Sandia National Labs' Aerial Suppression of Airborne Platforms (ASAP), the MARHES testbed procured quadrotor UAVs capable of on-board sensing. In the MAST project, quadrotor aerial and bio-inspired crawling ground robots were used to interact with each other for exploration and mapping. The ground robots were capable of taking pictures from different position and orientations. Due to limited space on the computer attached to the crawling robots, the quadrotor UAV was used to hover over them turn-by-turn to fetch those pictures over a wireless link. Once collected, the pose-stamped pictures were dumped to a server to create a map. On-board processing was thoroughly used on all the robots. In the ASAP project ([43]) the concept of forward stochastic reachability was used in the famous pursuit-evasion problem. In both these instances the locations of all the concerned hardware were provided through VICON motion capture system.

After contributing and getting motivated by these projects, I worked towards real-time and on-board sensing techniques mainly for the obstacle avoidance problem. This problem typically requires the need for on-board sensing if the environment is partially or fully unknown. Major upgrades in hardware include the addition of a stereo camera (Stereo Labs ZED) and a better computer (NVIDIA Jetson) on-board. A quadrotor UAV was custom developed using carbon fiber plates to form a frame to carry a Jetson TK1 computer, a ZED stereo camera and other flight modules to accomplish the task of on-board sensing. The platform was tested in a project in which the quadrotor UAV was set-up to pass through the square targets when their positions were not fully known. It was capable of detecting the targets and getting its own odometery using its stereo camera on-board. However, the platform was heavy and less agile for many applications. Mainly due to its weight, its flight time was quite restricted. A more efficient platform was developed to accomplish agile obstacle-avoidance tasks by combining the good properties of the existing platforms and upgrading the computer and camera on-board with a Jetson TX2 and a ZED mini respectively. Through studying existing perception and control techniques the algorithm for agile obstacle avoidance was developed. The stereo camera model was studied to get insight about stereo vision. Inspired from some existing techniques ([12], [14]), the obstacle detection was performed in real-time by processing the depth images from the on-board stereo camera and projecting the UAV into the future. The projected UAV was then checked for collisions in the depth image frame. LQR controllers were used to generate dynamically feasible trajectories. A thorough study of hybrid controls was performed to analyze trajectories generated from different control laws. The hybrid trajectory generation technique was combined with the obstacle detection strategy in a receding horizon fashion. The simulation and experimental results were used to validate the theoretical concepts.

Moving forward, I am motivated to exploit artificial intelligence in the obstacle avoidance application. I am trying to combine the current obstacle detection

technique with deep q-learning under the ROS framework. This will guarantee safety while the robot performs exploration. The obstacle detection function (*way-points2collision*) is already written in C++. For the machine learning part the library 'Keras' for Python is used. ROS service-client protocol is used to wrap C++ functions in a Python script. While the Python node subscribes to the features extracted from the depth image and suggest an action under ϵ greedy policy, the C++ node provides the service to execute that action and return the reward. Collision checks are to be performed within the C++ node to assign a reward for each action. However, this is still a work to be done in future.

Figure 6.1: Current MARHES aerial testbed

References

[1] Dong, Wei, Guo-Ying Gu, Xiangyang Zhu, and Han Ding. "Development of a Quadrotor Test BedModelling, Parameter Identification, Controller Design and Trajectory Generation." *International Journal of Advanced Robotic Systems* 12, no. 2 (2015): 7.

[2] Mahony, Robert, Vijay Kumar, and Peter Corke. "Multirotor aerial vehicles." *IEEE Robotics and Automation magazine* 20, no. 32 (2012).

[3] Grzonka, Slawomir, Giorgio Grisetti, and Wolfram Burgard. "Towards a navigation system for autonomous indoor flying." In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 2878-2883. IEEE, 2009.

[4] Bachrach, Abraham, Samuel Prentice, Ruijie He, and Nicholas Roy. "RANGE Robust autonomous navigation in GPS-denied environments." *Journal of Field Robotics* 28, no. 5 (2011): 644-666.

[5] Schmid, Korbinian, Teodor Tomic, Felix Ruess, Heiko Hirschmller, and Michael Suppa. "Stereo vision based indoor/outdoor navigation for flying robots." In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 3955-3962. IEEE, 2013.

[6] Richards, Arthur, and Jonathan P. How. "Aircraft trajectory planning with collision avoidance using mixed integer linear programming." In *American Control Conference, 2002. Proceedings of the 2002*, vol. 3, pp. 1936-1941. IEEE, 2002.

[7] Swingler, Ashleigh, and Silvia Ferrari. "On the duality of robot and sensor path planning." In *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, pp. 984-989. IEEE, 2013.

[8] Ferrari, Silvia, Chenghui Cai, Rafael Fierro, and Brent Perteet. "A geometric optimization approach to detecting and intercepting dynamic targets." In *American Control Conference, 2007. ACC'07*, pp. 5316-5321. IEEE, 2007.

References

[9] Eaton, Christopher M., Edwin KP Chong, and Anthony A. Maciejewski. "Robust UAV path planning using POMDP with limited FOV sensor." In *Control Technology and Applications (CCTA), 2017 IEEE Conference on*, pp. 1530-1535. IEEE, 2017.

[10] Chiang, Hao-Tien, Nick Malone, Kendra Lesser, Meeko Oishi, and Lydia Tapia. "Path-guided artificial potential fields with stochastic reachable sets for motion planning in highly dynamic environments." In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 2347-2354. IEEE, 2015.

[11] Malone, Nick, Hao-Tien Chiang, Kendra Lesser, Meeko Oishi, and Lydia Tapia. "Hybrid Dynamic Moving Obstacle Avoidance Using a Stochastic Reachable Set-Based Potential Field." *IEEE Transactions on Robotics* 33, no. 5 (2017): 1124-1138.

[12] Smith, Justin S., and Patricio Vela. "PiPS: Planning in perception space." In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 6204-6209. IEEE, 2017.

[13] Oleynikova, Helen, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart, and Enric Galceran. "Continuous-time trajectory optimization for online UAV replanning." In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pp. 5332-5339. IEEE, 2016.

[14] Matthies, Larry, Roland Brockers, Yoshiaki Kuwata, and Stephan Weiss. "Stereo vision-based obstacle avoidance for micro air vehicles using disparity space." In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 3242-3249. IEEE, 2014.

[15] Ratliff, Nathan, Matt Zucker, J. Andrew Bagnell, and Siddhartha Srinivasa. "CHOMP: Gradient optimization techniques for efficient motion planning." In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 489-494. IEEE, 2009.

[16] Scherer, Sebastian, Dave Ferguson, and Sanjiv Singh. "Efficient C-space and cost function updates in 3D for unmanned aerial vehicles." In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 2049-2054. IEEE, 2009.

[17] Ghosh, Sourish, and Joydeep Biswas. "Joint perception and planning for efficient obstacle avoidance using stereo vision." In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pp. 1026-1031. IEEE, 2017.

References

[18] Lamers, Kevin, Sjoerd Tijmons, Christophe De Wagter, and Guido de Croon. "Self-supervised monocular distance learning on a lightweight micro air vehicle." In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pp. 1779-1784. IEEE, 2016.

[19] Cigla, Cevahir, Roland Brockers, and Larry Matthies. "Image-based visual perception and representation for collision avoidance." In *IEEE International Conference on Computer Vision and Pattern Recognition, Embedded Vision Workshop*. 2017.

[20] Tedrake, Russ. "LQR-Trees: Feedback motion planning on sparse randomized trees." (2009).

[21] Mohta, Kartik, Michael Watterson, Yash Mulgaonkar, Sikang Liu, Chao Qu, Anurag Makineni, Kelsey Saulnier et al. "Fast, autonomous flight in GPS-denied and cluttered environments." *Journal of Field Robotics* 35, no. 1 (2018): 101-120.

[22] Barry, Andrew J., Peter R. Florence, and Russ Tedrake. "High-speed autonomous obstacle avoidance with pushbroom stereo." *Journal of Field Robotics* 35, no. 1 (2018): 52-68.

[23] Ding, Jerry, Jeremy H. Gillula, Haomiao Huang, Michael P. Vitus, Wei Zhang, and Claire J. Tomlin. "Hybrid systems in robotics." *IEEE Robotics & Automation Magazine* 18, no. 3 (2011): 33-43.

[24] Bezzo, Nicola, Rafael Fierro, Ashleigh Swingler, and Silvia Ferrari. "A disjunctive programming approach for motion planning of mobile router networks." *International Journal of Robotics and Automation* 26, no. 1 (2011): 13.

[25] Liu, Sikang, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. "Search-based motion planning for quadrotors using linear quadratic minimum time control." In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, IEEE, 2017.

[26] Dubey, Geetesh, Sankalp Arora, and Sebastian Scherer. "DROAN: Disparity space Representation for Obstacle Avoidance." In *IEEE Intl. Conf. on Intelligent Robots and Systems*. 2017.

[27] Richter, Charles, and Nicholas Roy. "Safe visual navigation via deep learning and novelty detection." In *Proc. of the Robotics: Science and Systems Conference*. 2017.

[28] Moore, Joseph, and Russ Tedrake. "Control synthesis and verification for a perching UAV using LQR-Trees." In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pp. 3707-3714. IEEE, 2012.

References

[29] C. T. John Lygeros and S. Sastry, *Hybrid Systems: Modeling, Analysis and Control.* 2008.

[30] A. J. Van Der Schaft and J. M. Schumacher, *An introduction to hybrid dynamical systems*, vol. 251.

[31] LaValle, Steven M. *Planning algorithms.* Cambridge university press, 2006.

[32] Furrer, Fadri, Michael Burri, Markus Achtelik, and Roland Siegwart. "RotorsA modular gazebo mav simulator framework." In *Robot Operating System (ROS)*, pp. 595-625. Springer International Publishing, 2016.

[33] Vetrella, Amedeo Rodi, Al Savvaris, Giancarmine Fasano, and Domenico Accardo. "RGB-D camera-based quadrotor navigation in GPS-denied and low light environments using known 3D markers." *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pp. 185-192. IEEE, 2015.

[34] Liu, Chang, and Stephen D. Prior. "Design and implementation of a mini quadrotor control system in GPS denied environments." In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pp. 462-469. IEEE, 2015.

[35] Fu, Changhong, Adrian Carrio, and Pascual Campoy. "Efficient visual odometry and mapping for unmanned aerial vehicle using ARM-based stereo vision preprocessing system." In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pp. 957-962. IEEE, 2015.

[36] Giusti, Alessandro, Jrme Guzzi, Dan C. Cirean, Fang-Lin He, Juan P. Rodrguez, Flavio Fontana, Matthias Faessler et al. "A machine learning approach to visual perception of forest trails for mobile robots." *IEEE Robotics and Automation Letters* 1, no. 2 (2016): 661-667.

[37] Loianno, Giuseppe, Michael Watterson, and Vijay Kumar. "Visual inertial odometry for quadrotors on se (3)." In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 1544-1551. IEEE, 2016.

[38] Concha, Alejo, Giuseppe Loianno, Vijay Kumar, and Javier Civera. "Visual-inertial direct SLAM." In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 1331-1338. IEEE, 2016.

[39] LaValle, Steven M. *Planning algorithms.* Cambridge university press, 2006.

[40] Malone, Nick, Hao-Tien Chiang, Kendra Lesser, Meeko Oishi, and Lydia Tapia. "Hybrid Dynamic Moving Obstacle Avoidance Using a Stochastic Reachable Set-Based Potential Field." *IEEE Transactions on Robotics (2017).*

References

[41] Hintz, Christoph, Shakeeb Ahmad, Joseph Kloeppel, Rafael Fierro. "Robust hybrid control for swinging-up and balancing an inverted pendulum attached to a UAV." In *Control Technology and Applications (CCTA), 2017 IEEE Conference on.*

[42] Brunson, Gregory, Forrest Gabrys, Nadia Coleman, Shakeeb Ahmad, Rafael Fierro. "Artificial Potential Fields for Real-Time Navigation and Obstacle Avoidance in GPS-Denied Environments." In *University of New Mexico (2018).*

[43] Vinod, Abraham P., Baisravan HomChaudhuri, Christoph Hintz, Anup Parikh, Stephen P. Buerger, Meeko M. K. Oishi, Greg Brunson, Shakeeb Ahmad, and Rafael Fierro. "Multiple Pursuer-Based Intercept via Forward Stochastic Reachability." In *American Control Conference, 2018. ACC'18*

[44] EscobarAlvarez, Hector D., Neil Johnson, Tom Hebble, Karl Klingebiel, Steven AP Quintero, Jacob Regenstein, and N. Andrew Browning. "RADVANCE: Rapid Adaptive Prediction for Visionbased Autonomous Navigation, Control, and Evasion." *Journal of Field Robotics* 35, no. 1 (2018): 91-100.

[45] PerezGrau, Francisco J., Ricardo Ragel, Fernando Caballero, Antidio Viguria, and Anibal Ollero. "An architecture for robust UAV navigation in GPSdenied areas." *Journal of Field Robotics* 35, no. 1 (2018): 121-145.

[46] Jung, Sunggoo, Sungwook Cho, Dasol Lee, Hanseob Lee, and David Hyunchul Shim. "A direct visual servoingbased framework for the 2016 IROS Autonomous Drone Racing Challenge." *Journal of Field Robotics* 35, no. 1 (2018): 146-166.

[47] Fierro, Rafael, and Frank L. Lewis. "A framework for hybrid control design." *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 27, no. 6 (1997): 765-773.

[48] NVIDIA Jetson (2018). Retrieved July 05, 2018, from https://www.nvidia.com/en-us/autonomous-machines.

[49] Stereo Labs (2018). Retrieved July 05, 2018, from https://www.stereolabs.com.

[50] Luminier QAV250 (2018). Retrieved July 05, 2018, from http://www.lumenier.com/products/multirotors/qav250.

[51] Pixracer PX4 (2018). Retrieved July 05, 2018, from https://docs.px4.io/en/flight_controller/pixracer.html.