ENPM667: Project-01

A Technical Report on

A Novel Potential Field Controller for Use on Aerial Robots

(Alexander C. Woods and Hung M. La, Senior Member, IEEE)

Adarsh Malapaka
UID: 118119625

Sahruday Reddy Patti
UID: 118218366

Date of Submission: 25th November, 2021

A. James Clark School of Engineering (Robotics)
University of Maryland, College Park, MD - 20742

Table of Contents

Appendices

List of Tables

List of Figures

Abstract

This report focuses on the derivation, implementation, and the generation of the results and performance obtained from the Extended Potential Field Controller (extended PFC) in the context of aerial systems such as quadcopters to track dynamic targets without colliding with obstacles along the path. The attractive and repulsive potential field functions are set up using relative positions between the robot, target and obstacles in the traditional mode of control. Although this method does provide satisfactory results for wheeled mobile robots, in the context of drones, due to the limitations on speeds and ability to instantaneously brake or change directions, the field functions are augmented with relative velocity terms to overcome the shortcomings of the traditional controller. The original paper used MATLAB and SIMULINK to implement and verify the devised algorithm before implementing the same on an actual system in an indoor setting. An effort has been made to recreate these results on MATLAB using the AR Drone 2.0 Simulation Kit and then to subsequently port the same control law onto a more realistic physics simulator such as Gazebo using the sjtu_drone ROS package for AR Drone 2.0, for those who cannot afford to implement on physical hardware. The extended controller shows better way point tracking with more optimum trajectory following in addition to minimizing time and making the system more robust to dynamic targets.

Keywords: *Potential Field controller, drones, aerial robots, dynamic target tracking, obstacle avoidance, quadcopter dynamics, Unmanned Autonomous Vehicles (UAV)*

Chapter 1

Introduction

The last two decades have witnessed a growing interest toward unmanned aerial vehicles (UAVs). Some of the most relevant applications relate to contribution to rescue missions, aerial inspection of structures, precision agriculture, aerial imaging/sensing, package delivery, etc. As a result, there is an increasing need of guidance, navigation, and control strategies for this category of aerial vehicles. UAVs fall into two main categories: fixed wing vehicles and rotary wing vehicles. The latter category has, in general, between one to eight rotors depending on design criteria [2]

During the past few years, potential field methods (PFM) for obstacle avoidance have gained increased popularity among researchers in the field of robots and mobile robots. The idea of imaginary forces acting on a robot has been suggested by Andrews and Hogan [1983] and Khatib [1985]. In these approaches obstacles exert repulsive forces onto the robot, while the target applies an attractive force to the robot [4]. The sum of all forces, the resultant force R, determines the subsequent direction and speed of travel. One of the reasons for the popularity of this method is its simplicity and elegance. Simple PFMs can be implemented quickly and initially provide acceptable results without requiring many refinements. Thorpe [1985] has applied the potential field method to off-line path planning and Krogh and Thorpe [1986] suggest a generalized potential field method that combines global and local path planning.

The application of such potential field functions can often be less effective for aerial robots as opposed to the ground robot counterpart. With the race for making sizes of such aerial robots smaller yet much more energy efficient, the potential field method is computationally non-intensive as compared to other industry standard algorithms for drones to follow objects or avoid obstacles. The applications of such systems can be utilized in situations which may be hazardous to human operators, such as assisting wild land fire fighters [11], search and rescue in hazardous conditions or locations [3], and disaster remediation [12–14]. Additionally, UAVs can be used in repetitive or tedious work where a human operator may lose focus such as infrastructure inspection, agricultural inspections, and environmental sensing. However,

often practical applications are hindered by hardware and computational challenges pertaining to the on-board capabilities of the drone. These aerial systems also often use a global positioning system (GPS) for self localization which works well in many situations, but severely limits their use in GPS-denied environments. In order to overcome these challenges, it is desired to explore the potential field controller further so as to develop an algorithm that is reliable in tracking operations yet is not very taxing on the available resources.

Chapter 2

Dynamic Modeling

In this chapter, the governing mathematical model of the quad-copter dynamical system shall be developed using concepts from Newton-Euler methods and the free body diagram of the system. This is an important step prior to establishing the control algorithm as the controller can then be tested on this developed model to check for efficacy before actually implementing on a real quad-copter system.

Before proceeding to set up the mathematical tools to describe the quad-copter system and its motion, the following assumptions were made:

- All the links on the quadcopter are assumed to be rigid bodies.
- The curvature of the Earth is neglected and density of air is assumed to remain constant while performing the computation of the dynamic equations of the robot.
- The kinematic and dynamic parameters of the robot such as mass, lengths, inertia matrices, etc are time-invariant.
- The drone flies with relatively small 3 degree of freedom Euler angles to prevent the system from reaching a singularity

2.1 Co-ordinate Frames and Transformations

Since the quad-copter shall be commanded to move with a relative velocity with respect to an observer on the ground (assumed to be standing on the point from which the quad-copter takes off), it is necessary to define co-ordinate reference frames on the ground and on the drone to be able to describe the relative positions and velocities between the two frames. Since the observer on the ground (taken to be the origin) is always stationary, this reference frame is an *inertial frame of reference*. If the drone is accelerating with respect to the ground frame, the frame attached to the drone shall be a *non-inertial frame*. For the sake of simplicity, the ground frame shall be

referred to as the 'inertial frame' and the frame fixed on the quad-copter shall be known as the 'body frame'. The frame on the drone is affixed to it's center of mass which is the point of intersection of both the arms of the drone.

Let $(\hat{e}_x, \hat{e}_y, \hat{e}_z) \in \mathbb{R}^3$ be the set of orthogonal unit vectors spanning the dimensions of the ground inertial frame E and $(\hat{b}_x, \hat{b}_y, \hat{b}_z) \in \mathbb{R}^3$ be the set of orthogonal unit vectors representing the body reference frame B, as shown in [Fig 2.1]

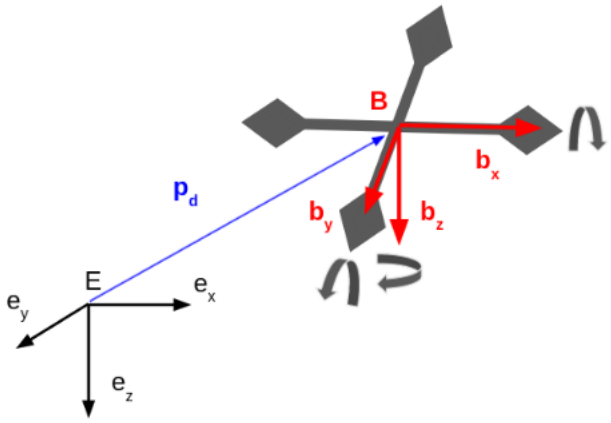

Figure 2.1: Ground and Body Reference Frames

2.1.1 Euler Angles

In this paper, the author had defined the orientation of the drone using the Euler angle conventions where rotation about the $\hat{b}_x$ axis is called '*roll*', rotation about the $\hat{b}_y$ axis is called '*pitch*' and the rotation about the $\hat{b}_z$ axis is called '*yaw*' (where, ϕ - Roll angle; θ - Pitch angle; ψ - Yaw angle).

Rotation Matrices

To be able to transform vectors (such as position and velocity) expressed in one reference frame (say J) with respect to another reference frame (say K), we use rotation matrices. The following rotation matrix describes frame $\{K\}$ with respect to $\{J\}$.

$$ {}^{J}_{K}R = \begin{bmatrix} \hat{X}_K.\hat{X}_J & \hat{Y}_K.\hat{X}_J & \hat{Z}_K.\hat{X}_J \\ \hat{X}_K.\hat{Y}_J & \hat{Y}_K.\hat{Y}_J & \hat{Z}_K.\hat{Y}_J \\ \hat{X}_K.\hat{Z}_J & \hat{Y}_K.\hat{Z}_J & \hat{Z}_K.\hat{Z}_J \end{bmatrix} \tag{2.1} $$

where each term inside the matrix, such as $\hat{X}_K.\hat{Z}_J$, is called a direction cosine. Using ${}^{J}_{K}R$, the inverse rotation transformation, that is, describing the rotation of frame $\{J\}$ with respect to $\{K\}$ can be done as follows:

$$ {}^{K}_{J}R = {}^{J}_{K}R^{-1} = {}^{J}_{K}R^{T} $$

For the drone and inertial ground frame system, the rotation matrices of the body frame rotated by the Euler angles (ϕ, θ and ψ) with respect to the ground inertial frame are given as follows [10],

Rotation about X axis (roll) is :

$$ {}^{E}_{B}R_{\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{2.2} $$

Rotation about Y axis (pitch) is :

$$ {}^{E}_{B}R_{\theta} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{2.3} $$

Rotation about Z axis (yaw) is :

$$ {}^{E}_{B}R_{\psi} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4} $$

Therefore, the total rotation matrix combining all the three Euler angles is found as follows,

$$ R_{\phi,\theta,\psi} = \begin{bmatrix} c_\psi c_\theta & c_\psi s_\phi s_\theta - s_\psi c_\phi & c_\psi c_\phi s_\theta + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\phi s_\theta + c_\psi c_\phi & s_\psi c_\phi s_\theta + c_\psi s_\phi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} \tag{2.5} $$

where, $c_x = \cos x$ and $s_x = \sin x$

Singularities

From equation 2.5, when $\cos(\theta) = 0$, the system loses one degree of freedom and hence results in a singularity. However, since only small angles are assumed in this paper, this is taken care of automatically.

Although the three Euler angles are extensively used for physically understanding the orientation of aircraft systems, in reality, the Euler angle representation can lead to singularities which result in the loss of one degree of freedom of the system. This is called Gimbal Lock. This is addressed by the usage of the quaternion representation for orientation.

2.2 Newton-Euler Formulation

Newton's Second Law: *The sum of the external forces acting upon an object equals the time rate of change of its linear momentum.*

$$\sum \vec{F} = \frac{\mathrm{d}(\vec{L})}{\mathrm{d}t} = \frac{\mathrm{d}(m\,\vec{v})}{\mathrm{d}t} \tag{2.6}$$

Euler's Second Law: *The sum of the external torques acting upon an object equals the time rate of change of its angular momentum.*

$$\sum \vec{\tau} = \frac{\mathrm{d}(\vec{H})}{\mathrm{d}t} = \frac{\mathrm{d}I_{cm}\vec{\omega}}{\mathrm{d}t} \tag{2.7}$$

To derive the fundamental expressions for the translational and rotational dynamics of the quad-copter, the Newton-Euler formulation for rigid bodies is used. In this method, Newton's second law of translational motion and it's rotational equivalent, Euler's second law are used to set up the dynamical equations [12].

Let, ${}^{E}_{B}\vec{p}$ be the position vector of the center of mass frame $\{B\}$ on the drone with respect to the ground inertial frame $\{E\}$ (as shown in [Fig 2.1]). The velocity of $\{B\}$ with respect to $\{E\}$, ${}^{E}_{B}\vec{v}$ is defined as,

$${}^{E}_{B}\vec{v} = \frac{\mathrm{d}({}^{E}_{B}\vec{p})}{\mathrm{d}t}$$

Given the mass of the robot 'm', the linear momentum of $\{B\}$ is,

$$\vec{L}_B = m({}^{E}_{B}\vec{v})$$

Using Newton's second law of motion with respect to the $\{E\}$ frame, where, $\overrightarrow{a_E}$ is the linear acceleration in the inertial $\{E\}$ frame.

$$\sum \overrightarrow{F_E} = \frac{\mathrm{d}(m\,\vec{v})}{\mathrm{d}t_E} = m\,\vec{a}_E \tag{2.8}$$

Representing the same with respect to the body $\{B\}$ frame,

$$\sum \vec{F}_B = m\left(\frac{\mathrm{d}\,\vec{v}}{\mathrm{d}t_B} + \vec{\omega}_B \times \vec{v}_B\right) = m\left(\vec{a}_B + \vec{\omega}_B \times \vec{v}_B\right) = m\,\vec{a}_B \tag{2.9}$$

Here, $\overrightarrow{\omega_B} = 0$, as $\{B\}$ is on the center of mass of the quad-copter.

If I_{cm} is the moment of inertia about the drone's center of mass, the angular momentum of the drone in the $\{B\}$ frame is given by,

$$\vec{H}_B = I_{cm}\vec{\omega}_B$$

Using Euler's second law of motion with respect to the $\{B\}$ frame, and assuming the inertia matrix to be time-invariant,

$$\sum \vec{\tau}_B = I_{cm}\frac{\mathrm{d}\vec{\omega}_B}{\mathrm{d}t_B} + \vec{\omega}_B \times I_{cm}\vec{\omega}_B = I_{cm}\vec{\alpha}_B + \vec{\omega}_B \times I_{cm}\vec{\omega}_B \tag{2.10}$$

where $\vec{\alpha}_B$ is the angular acceleration of the drone.

Expressing Eq. 2.9 and Eq. 2.10 in matrix form,

$$\begin{bmatrix}\sum \vec{F}_B \\ \sum \vec{\tau}_B\end{bmatrix} = \begin{bmatrix} mI_3 & 0 \\ 0 & I_{cm}\end{bmatrix}\begin{bmatrix}\vec{a}_B \\ \vec{\alpha}_B\end{bmatrix} + \begin{bmatrix} 0 \\ \vec{\omega} \times I_{cm}\vec{\omega}_B\end{bmatrix} \tag{2.11}$$

where, I_3 is a 3 x 3 identity matrix

2.3 Quadcopter Dynamics

Let I_{xx}, I_{yy} and I_{zz} be the moments of inertia about the center of mass along the $\hat{b}_x$, $\hat{b}_y$ and $\hat{b}_z$ axes respectively. For the quadcopter as seen in [Fig. 2.2], the two arms of the system are symmetrical and hence the moments of inertia about the $\hat{b}_x$ and $\hat{b}_y$ axes are equal. So, the moment of inertia matrix about the center of mass is given as,

$$I_{cm} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz}\end{bmatrix} \tag{2.12}$$

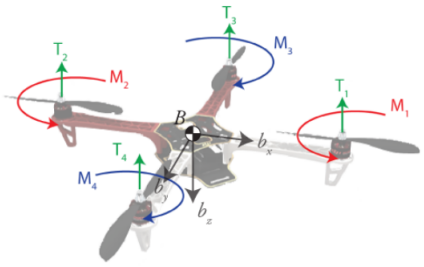

Figure 2.2: Free Body Diagram of the drone

Now, each of the four propellers on the drone that is connected to a motor generates a thrust force pointing upwards to lift the drone up. This force for each motor is given as follows where the negative sign appears as it is opposite to the defined positive sense of the z-axis,

$$\vec{T}_i = -k_T \Omega_i^2 \hat{b}_z \tag{2.13}$$

where, $\vec{T}_i$ is the thrust force and Ω_i is the angular velocity for i^{th} motor.

Similarly, the rotation of each propeller causes an associated moment to develop which can be expressed as below,

$$\vec{M}_i = -k_M \Omega_i^2 \hat{b_z} \tag{2.14}$$

Writing the equation for net force and torque on the system from the free body diagram with l taken to be the length of the drone's arms and m to be the drone's mass,

$$\sum \vec{F} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & mg - k_T \sum_{i=1}^{4} \Omega_i^2 \end{bmatrix} \begin{bmatrix} \hat{b_x} \\ \hat{b_y} \\ \hat{b_z} \end{bmatrix} \tag{2.15}$$

$$\sum \vec{\tau} = \begin{bmatrix} k_T(\Omega_3^2 - \Omega_4^2)l & 0 & 0 \\ 0 & k_T(\Omega_1^2 - \Omega_2^2)l & 0 \\ 0 & 0 & -k_M(\Omega_1^2 + \Omega_2^2 - \Omega_3^2 - \Omega_4^2)l \end{bmatrix} \begin{bmatrix} \hat{b_x} \\ \hat{b_y} \\ \hat{b_z} \end{bmatrix} \tag{2.16}$$

Substituting the above net force and torque equations in Eq. 2.11, we get the following equation that fully describes the motion of the quadcopter

system assuming small Euler angles (ϕ, θ, ψ).

$$\begin{bmatrix} \vec{a}_x \\ \vec{a}_y \\ \vec{a}_z \\ \overrightarrow{\alpha_x} \\ \overrightarrow{\alpha_y} \\ \overrightarrow{\alpha_z} \end{bmatrix} = \begin{bmatrix} \frac{1}{m} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{I_{xx}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \times \left(\begin{bmatrix} 0 \\ 0 \\ mg - k_T \sum_{i=1}^{4} \Omega^2 \\ k_T(\Omega_3^2 - \Omega_4^2)l \\ k_T(\Omega_1^2 - \Omega_2^2)l \\ -k_M(\Omega_1^2 + \Omega_2^2 - \Omega_3^2 - \Omega_4^2)l \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ \dot{\theta}\dot{\psi}(I_{zz} - I_{yy}) \\ \dot{\phi}\dot{\psi}(I_{xx} - I_{zz}) \\ \dot{\theta}\dot{\phi}(I_{yy} - I_{xx}) \end{bmatrix} \right) \tag{2.17}$$

$$= \begin{bmatrix} 0 \\ 0 \\ g - \frac{k_T}{m} \sum_{i=1}^{4} \Omega^2 \\ \frac{1}{I_{xx}} k_T(\Omega_3^2 - \Omega_4^2)l - \dot{\theta}\dot{\psi}(I_{zz} - I_{yy})) \\ \frac{1}{I_{yy}} k_T(\Omega_1^2 - \Omega_2^2)l - \dot{\theta}\dot{\psi}(I_{xx} - I_{zz})) \\ \frac{1}{I_{zz}}(-k_M(\Omega_1^2 + \Omega_2^2 - \Omega_3^2 - \Omega_4^2)l - \dot{\theta}\dot{\phi}(I_{yy} - I_{xx})) \end{bmatrix} \tag{2.18}$$

Chapter 3

Potential Field Controllers

In this chapter, the concept of potential field controllers is visited for target tracking by the drone. Using the concepts of a traditional potential field controller, an extended version of this controller is built to augment the performance of the traditional controller. The developed controller is put through the Lyapunov stability test for analysis purposes.

3.1 Background

A potential function is a differentiable real-valued function U : $\mathbb{R}^m \mapsto \mathbb{R}$ that is typically viewed as energy and hence the gradient of the potential is force. The gradient is a vector $\nabla U(q) = [\frac{\partial U}{\partial q_1}(q), ..., \frac{\partial U}{\partial q_m}(q)]^T$ which points in the direction that increases U, as a local maximum. Gradients can be intuitively viewed as forces acting on a positively charged particle robot which is attracted to the negatively charged goal [6]. Obstacles also have a positive charge which forms a repulsive force directing the robot away from obstacles. While dealing with first-order systems, the gradients can be viewed as velocity vectors instead of force vectors [7].

In general, the potential function is constructed as the sum of attractive and repulsive potentials which attracts the robot towards the target while deflecting it from obstacles.

$$U(q) = U_{att}(q) + U_{rep}(q)$$

3.2 Traditional Potential Field Controller

The classic form of a simple attractive potential field controller, in the absence of obstacles and a stationary target:

Let $\vec{p}_d = [x_d, y_d, z_d]^T$ and $\vec{p}_t = [x_t, y_t, z_t]^T$ be the position vectors of the drone and target from the ground inertial reference frame, respectively. The relative displacement between the drone and the target is given as,

$$\vec{p}_{dt} = \left[(x_d - x_t), (y_d - y_t), (z_d - z_t)\right]^T \tag{3.1}$$

3.2.1 Velocity due to the Attractive Potential:

The attractive potential function is defined as follows, where λ_1 is a positive scalar and $\|\vec{p}_{dt}\|$ is the magnitude of the relative distance between the robot and the target.

$$U_{att}(\vec{p}_d, \vec{p}_t) = \frac{1}{2}\lambda_1 \|\vec{p}_{dt}\|^2 \tag{3.2}$$

$$\|\vec{p}_{dt}\| = \sqrt{(x_d - x_t)^2 + (y_d - y_t)^2 + (z_d - z_t)^2} \tag{3.3}$$

In a potential field, the target corresponds to a minimum point of energy. So the velocity of the drone due to the attractive field, $\vec{v_d}^{att}$, is found from the negative gradient of this field as given below,

$$\vec{v_d}^{att}(\vec{p}_d, \vec{p}_t) = -\nabla U_{att}(\vec{p}_d, \vec{p}_t) \tag{3.4}$$

$$\tag{3.5}$$

$$= -\frac{\partial U_{att}}{\partial x}\hat{\imath} - \frac{\partial U_{att}}{\partial y}\hat{\jmath} - \frac{\partial U_{att}}{\partial z}\hat{k} \tag{3.6}$$

$$= -\nabla\left(\frac{1}{2}\lambda_1 \|\vec{p_{dt}}\|^2\right) \tag{3.7}$$

$$= -\frac{1}{2}\lambda_1 \nabla\left(\vec{x}_{dt}^2 + \vec{y}_{dt}^2 + \vec{z}_{dt}^2\right) \tag{3.8}$$

$$= -\lambda_1\left(\vec{x}_{dt} + \vec{y}_{dt} + \vec{z}_{dt}\right) \tag{3.9}$$

$$= -\lambda_1(\vec{p_d} - \vec{p_t}) \tag{3.10}$$

3.2.2 Velocity due to the Repulsive Potential:

In order to account for the presence of sources of repulsive potential such as obstacles, the repulsive potential is given as follows where $\vec{p}_o = [x_o, y_o, z_o]^T$ is the position vector of the obstacle with respect to the inertial frame of reference. Here, η_1 is a positive scalar.

$$U_{rep}(\vec{p}_d, \vec{p}_o) = \frac{1}{2}\eta_1 \frac{1}{\|\vec{p}_{do}\|^2} \tag{3.11}$$

$$\vec{v}_d^{rep}(\vec{p}_d, \vec{p}_o) = -\nabla U_{rep}(\vec{p}_d, \vec{p}_o) \tag{3.12}$$

$$= -\frac{\partial U_{rep}}{\partial x}\hat{\imath} - \frac{\partial U_{rep}}{\partial y}\hat{\jmath} - \frac{\partial U_{rep}}{\partial z}\hat{k} \tag{3.13}$$

$$= -\nabla\left(\frac{1}{2}\eta_1 \frac{1}{\|\vec{p}_{do}\|^2}\right) \tag{3.14}$$

$$= -\frac{1}{2}\eta_1 \nabla\left[\frac{1}{\vec{x}_{do}^2 + \vec{y}_{do}^2 + \vec{z}_{do}^2}\right] \tag{3.15}$$

$$= \eta_1 \frac{\vec{x}_{do} + \vec{y}_{do} + \vec{z}_{do}}{\left(\vec{x}_{do}^2 + \vec{y}_{do}^2 + \vec{z}_{do}^2\right)^2} \tag{3.16}$$

$$= \eta_1 \frac{\vec{p}_{do}}{\|\vec{p}_{do}\|^4} \tag{3.17}$$

Let P^* be the avoidance range distance from the obstacle beyond which there is no need for controlling the repulsive field velocity of the drone, $\vec{v}_d^{rep}$. Therefore, this can be written as,

$$\vec{v}_d^{rep}(\vec{p}_d, \vec{p}_o) = \begin{cases} \eta_1 \frac{\vec{p}_{do}}{\|\vec{p}_{do}\|^4}, & \|\vec{p}_{do}\| \leq P^* \\ 0, & \|\vec{p}_{do}\| > P^* \end{cases} \tag{3.18}$$

So, the total controller is obtained by the summation of both the attractive and repulsive velocities accounting for multiple obstacles (n is the number of obstacles in the area)

$$\vec{v}_d^{PFC}(\vec{p}_d, \vec{p}_t, \vec{p}_o) = \begin{cases} -\lambda_1(\vec{p}_d - \vec{p}_t) + \sum_{i=0}^{n} \eta_1 \frac{\vec{p}_{do}^i}{\|\vec{p}_{do}^i\|^4}, & \|\vec{p}_{do}^i\| \leq P^* \\ -\lambda_1(\vec{p}_d - \vec{p}_t), & \|\vec{p}_{do}^i\| \geq P^* \end{cases} \tag{3.19}$$

Observations:

- It is to be noted that the attractive velocity part in the overall traditional PFC velocity expression in the original paper contains a typo, printing it as $(\vec{p}_t - \vec{p}_d)$ instead of $(\vec{p}_d - \vec{p}_t)$.
- The positive constants λ_1 and η_1 are found manually by the trial and error method.
- The target and obstacles are both assumed to be stationary.

3.3 Extended Potential Field Controller

The traditional potential field controller developed previously works satisfactorily for wheeled mobile robots. However, in the case of aerial robots which cannot abruptly stop at a location and need to make continuous fine adjustments to the position and velocity, the potential field controller needs to account for velocity as well. Also, the previous controller works only for stationary targets and needs to be modified for dynamic targets.

The extended potential field controller addresses these issues by augmenting the traditional controller with new repulsive and attractive fields arising due to the relative velocity between the drone and the target, $\vec{v}_{dt}$. The intention of this controller is to make the drone be able to track the dynamic target's velocity.

3.3.1 Velocity due to the Attractive Potential:

The attractive potential can be written as,

$$U_{att}(\vec{v}_d, \vec{v}_t) = \frac{1}{2}\lambda_2 \|\vec{v}_d - \vec{v}_t\|^2 \|\overrightarrow{v_{dt}}\| = \|\vec{v}_d - \vec{v}_t\| = \sqrt{\dot{x}_{dt}^2 + \dot{y}_{dt}^2 + \dot{z}_{dt}^2} \quad (3.20)$$

The negative gradient of the attractive potential minimizes the relative velocity between the drone and the target ensuring proper target tracking, So, the desired velocity of the drone, due to the attractive potential field is computed as,

$$\vec{v}_d^{att}(\vec{v}_d, \vec{v}_t) = -\nabla U_{att}(\vec{v}_d, \vec{v}_t) \quad (3.21)$$

$$= -\frac{\partial U_{att}}{\partial \dot{x}}\hat{\imath} - \frac{\partial U_{att}}{\partial \dot{y}}\hat{\jmath} - \frac{\partial U_{att}}{\partial \dot{z}}\hat{k} \quad (3.22)$$

$$= -\nabla\left(\frac{1}{2}\lambda_2 \|\overrightarrow{v_{dt}}\|^2\right) \quad (3.23)$$

$$= -\frac{1}{2}\lambda_2 \nabla\left(\dot{\vec{x}}_{dt}^2 + \dot{\vec{y}}_{dt}^2 + \dot{\vec{z}}_{dt}^2\right) \quad (3.24)$$

$$= -\lambda_2\left(\dot{\vec{x}}_{dt} + \dot{\vec{y}}_{dt} + \dot{\vec{z}}_{dt}\right) \quad (3.25)$$

$$= -\lambda_2(\vec{v}_d - \vec{v}_t) = -\lambda_2(\vec{v}_{dt}) \quad (3.26)$$

where, λ_2 is a positive scalar.

3.3.2 Velocity due to the Repulsive Potentials:

$$U_{rep_1}(\vec{v}_d, \vec{v}_o) = \frac{1}{2}\eta_2 \frac{1}{\|\vec{v}_{do}\|^2} \tag{3.27}$$

$$\vec{v}_d^{rep_1}(\vec{v}_d, \vec{v}_o) = -\nabla U_{rep_1}(\vec{v}_d, \vec{v}_o) \tag{3.28}$$

$$= -\frac{\partial U_{rep_1}}{\partial \dot{x}}\hat{\imath} - \frac{\partial U_{rep_1}}{\partial \dot{y}}\hat{\jmath} - \frac{\partial U_{rep_1}}{\partial \dot{z}}\hat{k} \tag{3.29}$$

$$= -\nabla\left(\frac{1}{2}\eta_2 \frac{1}{\|\vec{v}_{do}\|^2}\right) \tag{3.30}$$

$$= -\frac{1}{2}\eta_2 \nabla\left(\frac{1}{\dot{\vec{x}}_{do}^2 + \dot{\vec{y}}_{do}^2 + \dot{\vec{z}}_{do}^2}\right) \tag{3.31}$$

$$= \eta_2 \frac{\vec{v}_d - \vec{v}_o}{(\dot{\vec{x}}_{do}^2 + \dot{\vec{y}}_{do}^2 + \dot{\vec{z}}_{do}^2)^2} \tag{3.32}$$

$$= \eta_2 \frac{\vec{v}_{do}}{\|\vec{v}_{do}\|^4} \tag{3.33}$$

The above repulsive potential velocity is zero when the obstacle's velocity is zero and hence it would not have any effect on the drone's motion.

$$\vec{v_d}^{rep_1}(\vec{v}_d, \vec{v}_o) = \begin{cases} \eta_2 \frac{\vec{v}_d - \vec{v}_o}{\|\vec{v}_{do}\|^4}, & \|\vec{v}_o\| < 0 \\ 0, & \|\vec{v}_o\| = 0 \end{cases} \tag{3.34}$$

Although the above attractive and repulsive velocities in conjunction with the traditional potential field velocity would be sufficient to setup the velocity of the drone from the extended controller, the time required to reach the target is further minimized by accounting the time rate of change of $\|\vec{p}_{do}\|$. If the drone is already moving away from the obstacle, then $\|\dot{\vec{p}}_{do}\| \geq 0$ in which case the repulsive action need not be taken, even if the drone is within the avoidance range (P^*) of the obstacle as it will eventually move away from the obstacle towards the target. In other words, when the distance between the drone and the obstacle ($\|\dot{\vec{p}}_{do}\| < 0$) is decreasing, the drone needs to avoid the obstacle.

$$\vec{v}_d^{rep_2}(\vec{p}_d, \vec{p}_o) = \begin{cases} -\eta_3 \|\dot{\vec{p}}_{do}\| \frac{\vec{p}_{do}}{\|\vec{p}_{do}\|}, & \|\dot{\vec{p}}_{do}\| \leq 0 \\ 0, & \|\dot{\vec{p}}_{do}\| \geq 0 \end{cases} \tag{3.35}$$

where, η_3 is a positive scalar.

Therefore, the effective extended potential field controller velocity is given as,

$$\vec{v}_d^{ePFC} = \vec{v}_d^{PFC} - \lambda_2(\vec{v}_d - \vec{v}_t) + \sum_{i=0}^{n} \eta_2 \frac{\vec{v}_d - \vec{v}_0^i}{\|\vec{v}_{do}^i\|^4} - \sum_{i=0}^{n} \eta_3 \|\dot{p}_{do}^i\| \frac{p_{do}^i}{\|p_{do}^i\|} \quad (3.36)$$

where, $\|\vec{v}_{do}^i\| \neq 0$ and $\|\dot{\vec{p}}_{do}^i\| < 0$ and n is the number of obstacles in the area.

Observations:

- The extended potential field controller velocity adds additional terms to the traditional expression to account for the target and obstacle velocities and to minimize the time taken to reach the target.
- Since, the velocities in the above expressions were found from the position vectors with respect to the inertial ground frame, the resulting velocity should be transformed into the body $\{B\}$ co-ordinate system of the drone before giving it as an input to the system using the following rotation matrix,

$$\vec{v}_B^{ePFC} = {}_B^E R^T * \vec{v}_E^{ePFC} = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \vec{v}_{d,E}^{ePFC} \quad (3.37)$$

where, ψ is the yaw angle about the drone's body z-axis.

Note: There is a typo in the rotation matrix printed in the original paper used for transforming ground velocities to body velocities.

3.4 Controller Stability

Lyapunov Function: *For a Linear-Time-Invariant (LTI) system, a function $V : \mathbb{R}^n \mapsto \mathbb{R}_+$ is a Lyapunov function if it satisfies the following conditions:-*

- $V(\vec{X}) > 0$ for $\vec{X}(t) \neq 0$
- $V(\vec{X}) = 0$ for $\vec{X}(t) = 0$
- In the absence of external input, $\frac{\mathrm{d}V(\vec{X})}{\mathrm{d}t} < 0$ when $\vec{X}(t) \neq 0$

Lyapunov Stability: *A Linear-Time-Invariant (LTI) system is stable if and only if it has a Lyapunov function, V.*

To analyze the stability of the velocity controller, the following artificial potential function of the controller is chosen as a potential candidate for the Lyapunov function,

$$V = U_{att} = \frac{1}{2}\lambda_1 \|\overrightarrow{p_{dt}}\|^2 + \frac{1}{2}\lambda_2 \|\overrightarrow{v_{dt}}\|^2 \tag{3.38}$$

Observations:

- For $\|\vec{p_d} - \vec{p_t}\|, \|\vec{v_d} - \vec{v_t}\| = 0, V = 0$
- For $\|\vec{p_d} - \vec{p_t}\|, \|\vec{v_d} - \vec{v_t}\| \neq 0, V > 0$

Hence, the chosen function is positive definite and satisfies the first two conditions of a Lyapunov function as seen above.

The Lie derivative of the function, V is given by,

$$V^* = \frac{\partial V}{\partial \vec{p}_{\,dt}} \vec{v}_{\,dt} + \frac{\partial V}{\partial \vec{v}_{\,dt}} \vec{p}_{\,dt} = \lambda_1 \|\overrightarrow{p_{dt}}\| \vec{v}_{\,dt} + \lambda_2 \|\overrightarrow{v_{dt}}\| \vec{a}_{\,dt}$$

where, $\vec{a}_{\,dt}$ is the relative acceleration between the drone and the target.

$$\overrightarrow{a_{dt}} = \dot{\overrightarrow{v_{dt}}} = \frac{d}{dt}\left(\frac{1}{2}\lambda_2 \nabla \left(\vec{\dot{x}}_{\,dt}^{\,2} + \vec{\dot{y}}_{\,dt}^{\,2} + \vec{\dot{z}}_{\,dt}^{\,2}\right)\right) \tag{3.39}$$

$$= \frac{d}{dt}\left(-\lambda_2 \|\vec{v}_{\,dt}\|\right) \tag{3.40}$$

$$= -\lambda_2 \frac{\|\vec{v}_{\,dt}(t) - \vec{v}_{\,dt}(t-1)\|}{\|\Delta t\|} \tag{3.41}$$

Substituting the values of v_{dt} and a_{dt},

$$V^* = -({\lambda_1}^2 \|\overrightarrow{p_{dt}}\|^2 + {\lambda_2}^2 \|\overrightarrow{v_{dt}}\| \frac{\|\vec{v}_{\,dt}(t) - \vec{v}_{\,dt}(t-1)\|}{\|\Delta t\|})$$

Observations:

- Here, $\|\overrightarrow{p_{dt}}\|, \|\overrightarrow{v_{dt}}$ and $\frac{\|\vec{v}_{\,dt}(t) - \vec{v}_{\,dt}(t-1)\|}{\|\Delta t\|}$ are always positive, $V^* < 0$.

- Since $V^* < 0$, V is a Lyapunov function for the system and the proposed velocity based potential field controller is stable while tracking dynamic targets.
- Only the attractive potentials have been considered in the Lyapunov function and stability as the repulsive potentials are designed to be inherently unstable.

Chapter 4

Simulation and Results

In the previous chapters, the mathematical model of the quadcopter was set up and the extended version of the potential field controller was developed. With these equations, the system is simulated using a MATLAB/Simulink model and further in the physics engine Gazebo. The AR Drone 2.0 quadcopter platform is used as the drone in the testing environment. The state space representation of the AR Drone's platform dynamics are taken from the AR Drone Simulink Development Kit [8]. An important note is that the velocity of the target and obstacles have been assumed to be zero unlike the equations derived earlier and the results simulated in the paper. This is because the paper made use of actual laboratory experiments to validate the controller using a 'follow the target' approach.

AR Drone 2.0

The AR Drone 2.0 quadcopter is a vehicle with substantial complexity and versatility; it has been used by several research groups from various universities. Despite being a relatively low-priced robot it contains several measuring elements found in more expensive vehicles. Some of these features are [5]:

- Front facing 720p HD Camera
- 3 axis gyroscope 2000 deg/second precision
- 3 axis accelerometer $\pm$50mg precision
- 3 axis magnetometer 6 deg precision
- Ultrasound sensors for ground altitude measurements
- 60 FPS vertical QVGA camera for ground speed measurements

This robot is also equipped with a 32-bit 1 GHz ARM Cortex A8 processor, 1 GB DDR2 RAM, and runs Linux. This means that the developed controller can be implemented on board the drone in future work.

4.1 MATLAB and Simulink

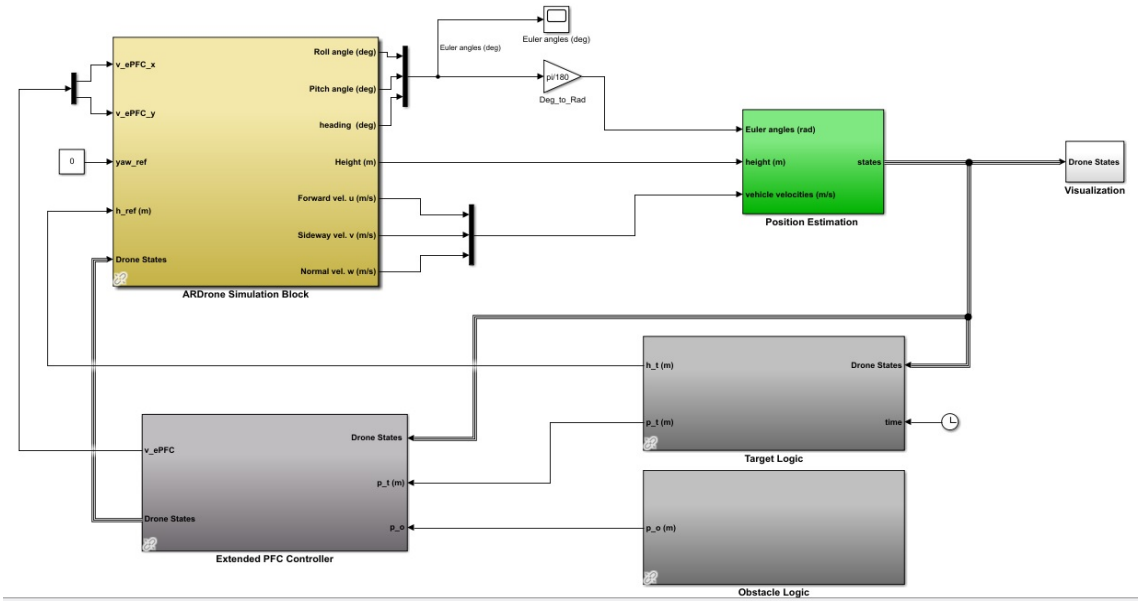

Figure 4.1: Implemented overall Simulink model

[Fig. 4.1] shows the entire SIMULINK structure for the simulation. The AR Drone Simulation and Position Estimation blocks have been taken from the MathWorks library while the other blocks have been custom defined. The internals of each of these blocks are presented in the subsequent sections. Also, codes wherever used in these blocks have been included in the end in Appendix B.

The ePFC controller uses feedback information from the ARDrone simulation and position estimator blocks. The output of the ARDrone simulation block is simply the velocity of the drone, and the position estimator uses an integrator with zero initial conditions to calculate position. [9]

4.1.1 Simulink Blocks

The Extended PFC controller block in Fig. 4.2 has an ePFC logic block to which the states of the drone and the target and obstacle positions are given as inputs. Using these values and assuming the velocities of the targets and obstacles are zero, it outputs the X and Y velocity output of the controller

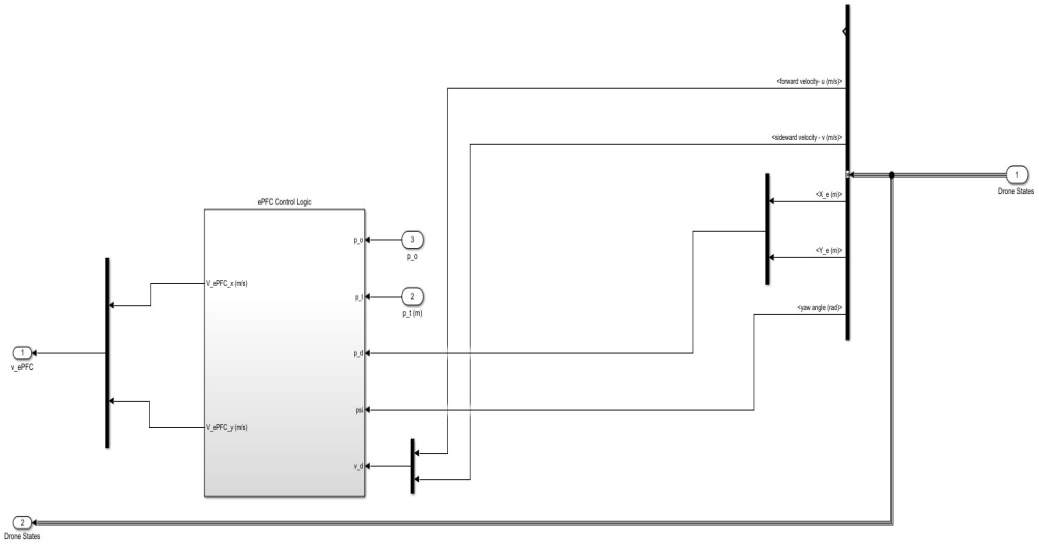

Figure 4.2: Extended PFC Controller Block

which is later saturated to [-1,1]. Fig. 4.3 shows the interior of the ePFC logic block which consists of three Matlab function blocks. The first function block computes the PFC velocities which is then cascaded to the second block where the total ePFC velocities are found. The final block transforms these inertial velocities into body velocities before giving them as input to the AR simulation block.

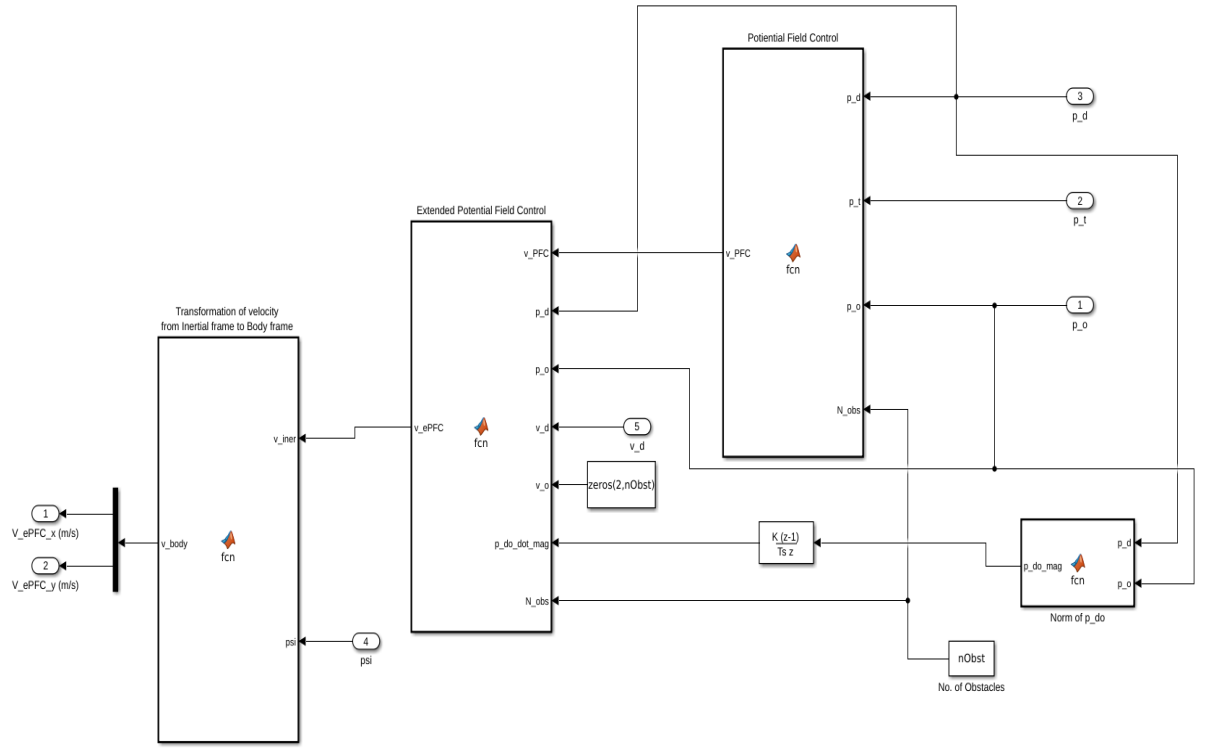

Figure 4.3: Extended PFC Control Logic

The Target Logic block [Fig. 4.4] uses the generated desired waypoints (using getWaypoints.m) to compute the desired target coordinates, heading angle and height. The states of the drone are given as input to this block apart from time block to dynamically change the next target once the drone reaches the present target. Similarly, the Obstacle Logic block takes the position details of the obstacles using the 'getObstaclePoints.m' script file and passes that as input to the ePFC control block. These obstacle coordinate points are then passed through a Matlab function to compute the rate of change of the norm of the relative distance between the drone and the obstacle. This is achieved using a Discrete Time Derivative block in Simulink.

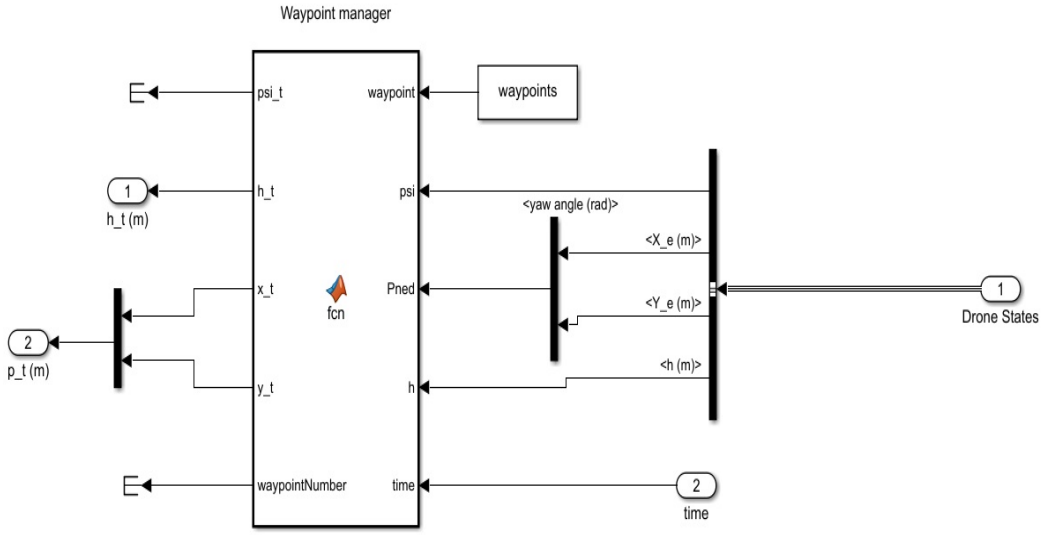

Figure 4.4: Target Logic Block

The Visualization block [Fig. 4.5] takes the states of the drone as input to correspondingly plot the data on the respective scopes. Here, an XY plotter block is used to plot the X and Y positions of the drone in real time, as it is moving.

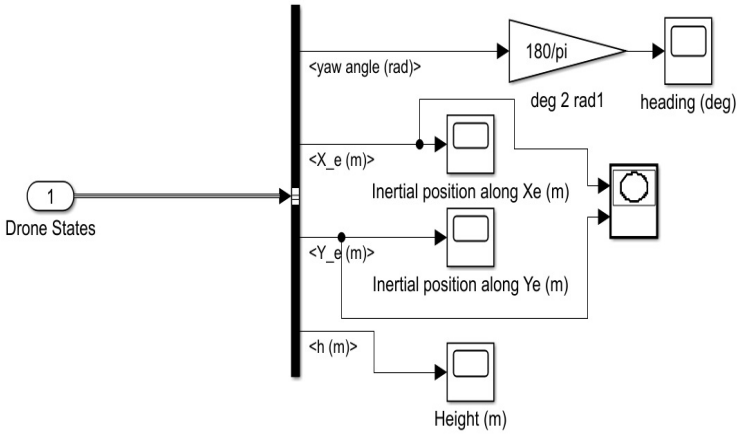

Figure 4.5: Visualization Block

4.2 ROS and Gazebo

The following sections elucidate the structure of the ROS environment used to perform way point tracking and obstacle avoidance in ROS + Gazebo. To simulate the AR Drone 2.0, the *sjtu_drone* [13] ROS package, which is a modified version of the *tum_simulator* [14] ROS package such that it is compatible with ROS Melodic distributions, has been used. It is to be noted that the ROS packages for simulating the AR drone 2.0 have last been compatible with ROS Kinetic and hence one may expect to end in installation/missing header files errors while trying to install for ROS versions Melodic and higher. The codes and files for this ROS structure are given in Appendix A.

4.2.1 Computation Graph

ROS Node

- **/ePFC:** This node defines publishers to initiate takeoff of the drone and to publish the desired velocity of the drone to the gazebo simulation node. It also defines subscribers to receive the current pose and velocity data of the drone. Using this information, the node computes the necessary commanded velocity based on the extended potential field equations defined previously.
- **/gazebo:** This node is run automatically when Gazebo is launched and is a built-in node which can be used to set/modify Gazebo's simulation parameters.

ROS Topics

- **/drone/takeoff:** This topic is used to make the drone to takeoff from the origin landing point just after the gazebo simulation is run.
- **/drone/gt_pose:** This topic provides the present position (x,y,z) and orientation (quaternion) of the drone with respect to the ground reference frame (p_d). By default, the robot always starts from (0,0,0) in the simulation.
- **/drone/gt_vel:** This topic provides the present linear velocity (x,y,z) and angular velocity (x,y,z) of the drone with respect to the ground reference frame (v_d). By default, the robot always starts from (0,0,0) in the simulation.

- **/cmd_vel:** This topic allows the user to set a desired velocity (both linear and angular) of the drone with respect to the ground reference frame (v_d). It is to noted that the input velocity values must be saturated in the range [-1,1] (m/s).

ROS Messages

- **geometry_msgs::Twist:** The contents of this message express velocity in free space broken into its linear and angular parts.
- **geometry_msgs::Pose:** The contents of this message are a representation of pose in free space, composed of position and orientation.
- **std_msgs::Empty:** This message type defines empty messages that can be used when there is no actual data to be sent to the subscriber.

RQT Graph

The below figure displays the ROS computation graph of the PFC waypoint tracking system containing all the nodes, topics and direction of communication. The *rqt_graph* tool in ROS provides a GUI plugin to visualise this graph. Alternatively, the *rosgraph* command provides a command line version of rqt_graph displaying periodic graph information in text format [15].

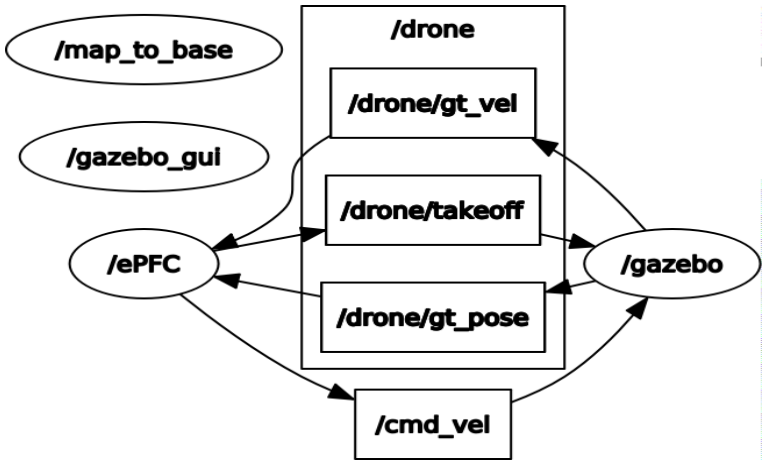

Figure 4.6: RQT Graph showing all Nodes (ovals) and Topics (rectangles)

4.2.2 Gazebo World

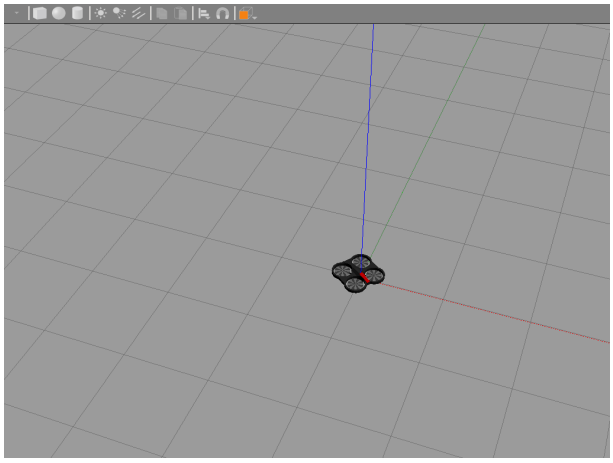

Figure 4.7: Default empty Gazebo world

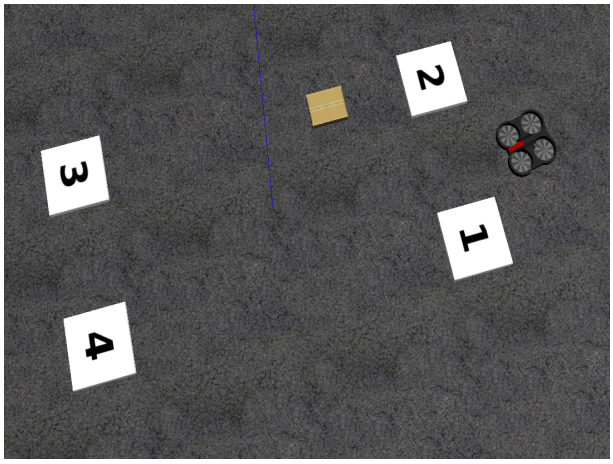

Figure 4.8: Gazebo world to mimic Fig 8 in the paper (obstacle: box)

Upon launching Gazebo with the drone's launch file, the AR drone is spawned in an empty world (as shown in [Fig. 4.7]). Although this world can be used to observe the motion of the drone while it is trying to reach the waypoint targets while avoiding obstacles, another custom created Gazebo world [Fig. 4.8] has been used to recreate the waypoint tracking corresponding to Fig. 8 and 9 in the original paper. The numbered boxes represent the

waypoints and the cardboard box represents the obstacle. It is to be noted that this world file has been created using Gazebo objects already present in our environment directories and hence some of these may not appear if the corresponding SDF/texture files are missing.

4.3 Results and Plots

With the simulation environments of MATLAB and Gazebo established for testing the effectiveness of the controller, the various results obtained in the original paper [1] that include static targets and/or obstacles are simulated in the subsequent sections. The results in the paper obtained by using an actual drone and motion capture system have been replaced with a corresponding simulation in Gazebo.

4.3.1 For Fig. 8 and 9

Waypoint	X (m)	Y (m)	Obstacle	X (m)	Y (m)
1	2.5	-1.0	1	1.0	1.0
2	2.5	1.0	-	-	-
3	-2.5	1.0	-	-	-
4	-2.5	-1.0	-	-	-
5	2.5	-1.0	-	-	-

Table 4.1: Fig 8 and 9 waypoints and obstacle positions

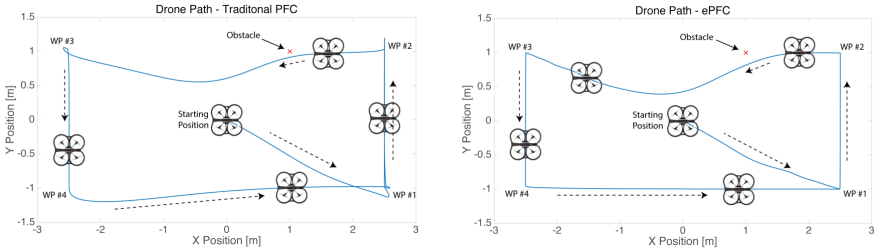

Figure 4.9: Traditional vs ePFC: Original Paper

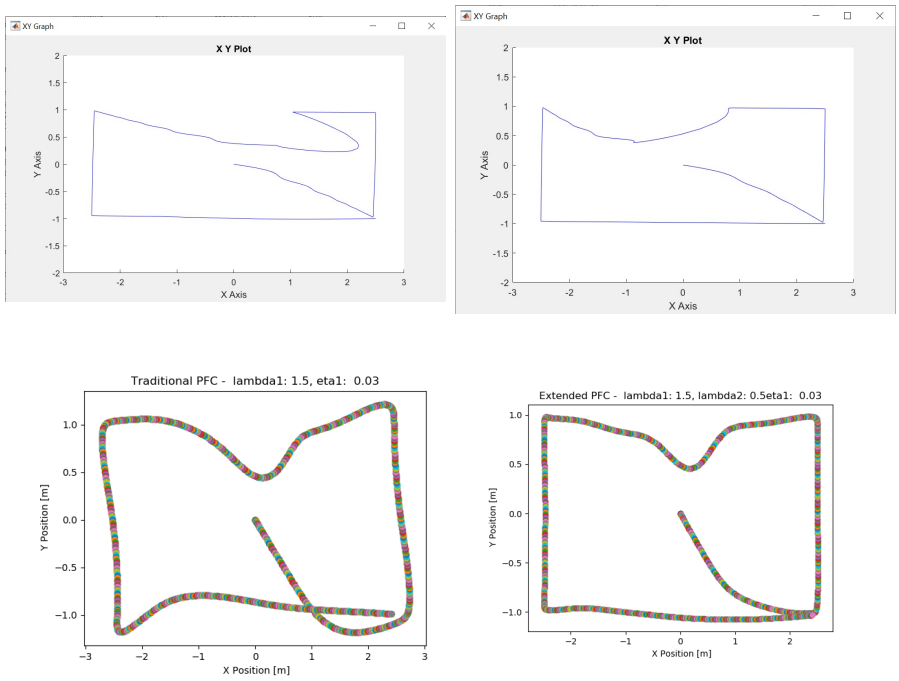

Figure 4.10: Traditional vs ePFC: Simulink (Top) and Gazebo (Bottom)

Observations:

- In Fig 4.10, the plots from Simulink clearly show the effectiveness of the ePFC to return an optimized path while avoiding the obstacle as opposed to the traditional control.
- There is no noticable overshoot in the system while using ePFC which validates the result given in the paper.
- In the next section onwards, only the corresponding Gazebo figures shall be shown for the results in the paper as we have seen satisfactory results from Simulink.

4.3.2 For Fig. 10

Waypoint	X (m)	Y (m)	Obstacle	X (m)	Y (m)
1	6.0	0.0	1	1.2	0.0
2	0.0	0.0	2	2.5	0.6
3	-	-	3	3.0	0.0
4	-	-	4	4.2	0.5
5	-	-	5	4.7	-0.5

Table 4.2: Fig 10 waypoints and obstacle positions
Note: The x-values given in the paper were offset by 3 to ensure the drone starts at (0,0) in simulation

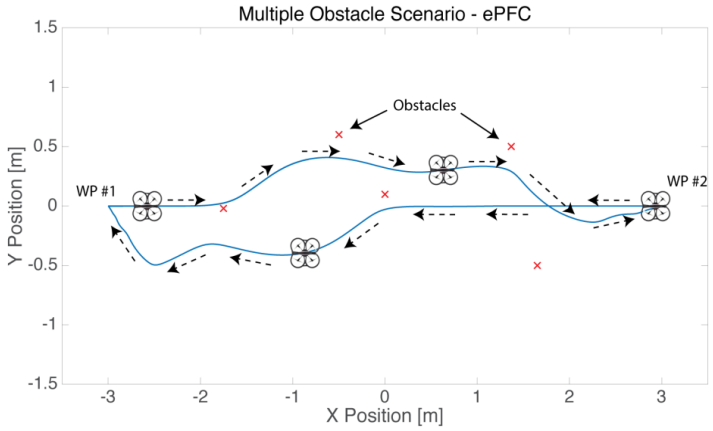

Figure 4.11: ePFC: Multiple Obstacles (Original Paper)

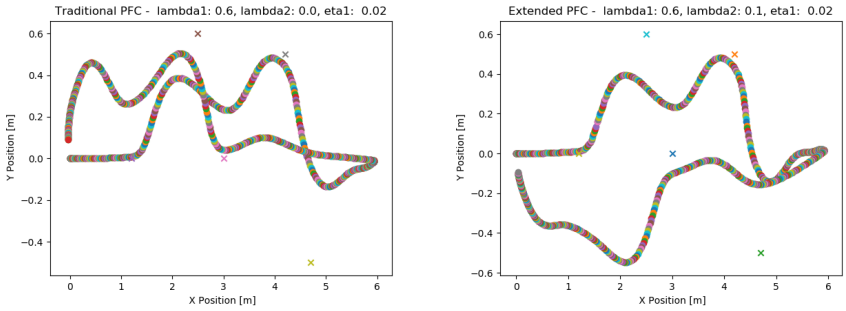

Figure 4.12: Traditional vs ePFC: Multiple Obstacles (Gazebo)

Observations:

- In the Gazebo plot, although the traditional PFC navigates the bot through all the waypoints, it takes a much longer route than that of the ePFC plot clearly showing the effectiveness of the algorithm.
- In the plots of subsequent sections, the traditional figure shall not be shown as it has already been known to perform poorly than ePFC.

4.3.3 For Fig. 14

Waypoint	X (m)	Y (m)	Obstacle	X (m)	Y (m)
1	4.2	0.0	1	-	-
-	-	-	-	-	-

Table 4.3: Fig 14 waypoints and obstacle positions

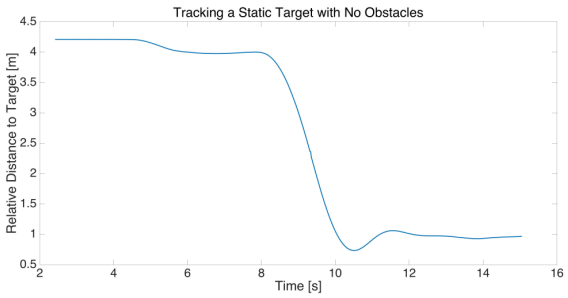

Figure 4.13: ePFC: Tracking static target - No Obstacle (Original Paper)

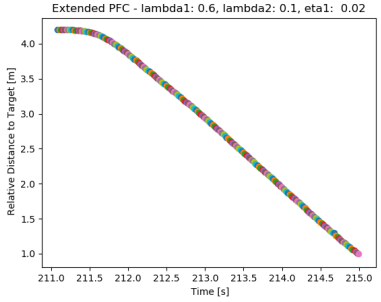

Figure 4.14: ePFC: Tracking static target - No Obstacle (Gazebo)

Observations:

- From the Gazebo plot, the drone takes 4.5 seconds (215.5-211.0) to settle at the target which is very close to the 5 seconds in the paper.
- There is 0% overshoot which coincides with the simulated value presented in the paper.

4.3.4 For Fig. 15

Waypoint	X (m)	Y (m)	Obstacle	X (m)	Y (m)
1	5.2	0.0	1	2.5	0.0
-	-	-	-	-	-

Table 4.4: Fig 15 waypoints and obstacle positions

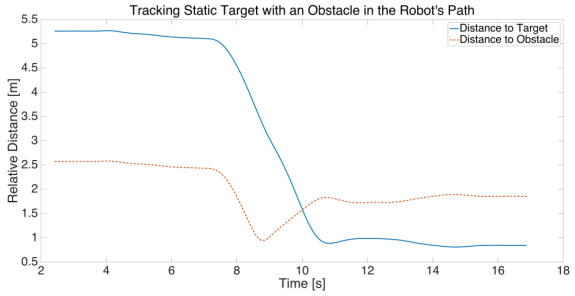

Figure 4.15: ePFC: Tracking static target - With Obstacle (Original Paper)

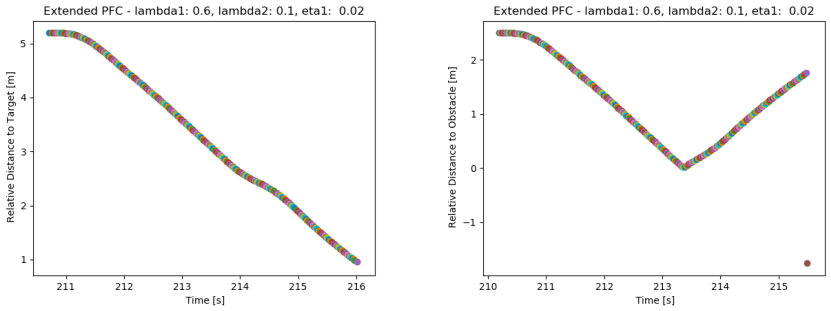

Figure 4.16: ePFC: Tracking static target - With Obstacle (Gazebo)

Observations:

- From the Gazebo plots, both the relative distance graphs are very close to the one obtained by the author.

4.3.5 For Fig. 17 and 19

Waypoint	X (m)	Y (m)	Obstacle	X (m)	Y (m)
1	-1.5	-0.5	1	0.0	0.4
2	1.5	-0.5	-	-	-
3	1.5	0.5	-	-	-
4	-1.5	0.5	-	-	-
5	-1.5	-0.5	-	-	-

Table 4.5: Fig 17 waypoints and obstacle positions

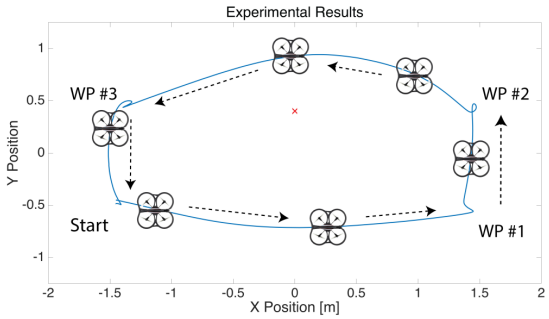

Figure 4.17: ePFC: Experimental Waypoints with Obstacle (Original Paper)

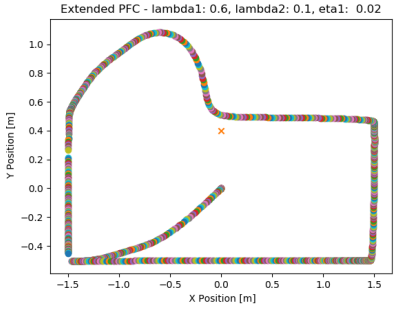

Figure 4.18: ePFC: Experimental Waypoints with Obstacle (Gazebo)

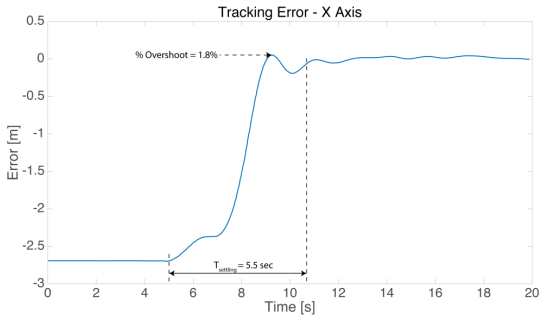

Figure 4.19: ePFC: X-axis Tracking Error (m) (Original Paper)

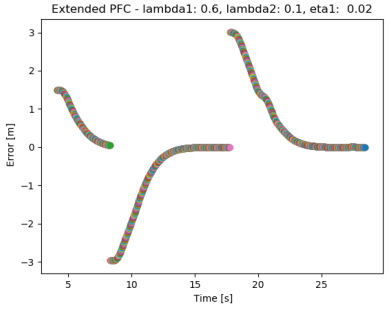

Figure 4.20: ePFC: X-axis Tracking Error (m) (Gazebo)

Observations:

- In [Fig 4.18], although the trajectpry doesn't very closely resemble that of the original plot, the controller still performs well in terms of overshoot and oscillations.
- In [Fig 4.20], the portion of the graph between 7 seconds until 12 seconds is the region of interest and it is seen that the settling time for this is 5 seconds which is close to what is in the paper.

4.3.6 Sample Demo

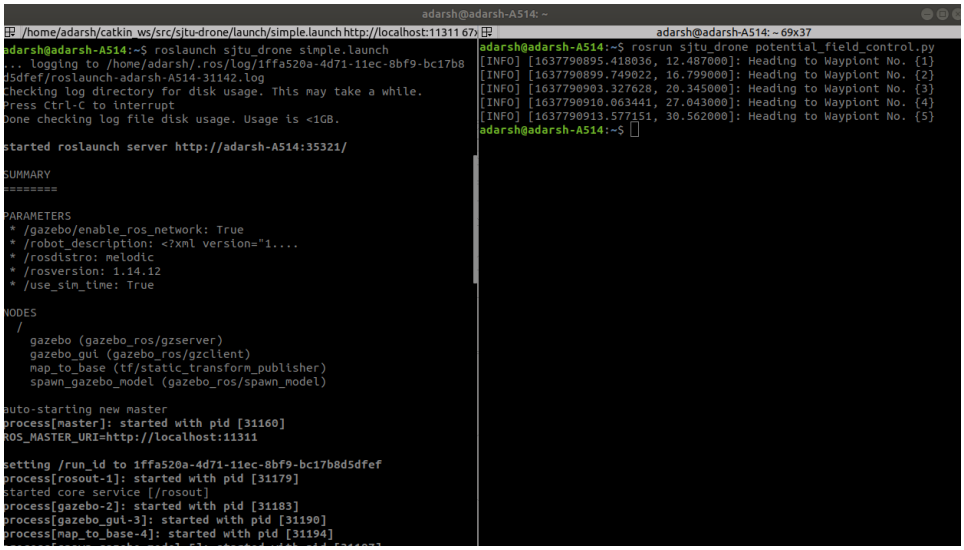

Figure 4.21: Running the Gazebo visualization of tracking Table 4.1 waypoints

Using the waypoints and obstacle data given in Table 4.1, the gazebo simulation of the AR drone with appropriate objects in the world to visualize the waypoints are given. This could've been achieved with AR tags but that is out of the scope of this report. Fig. 4.21 shows the commands run in the terminal to launch the simulation and start the ePFC node while Fig. 4.22 shows the drone tracking the waypoints at different instances of time.

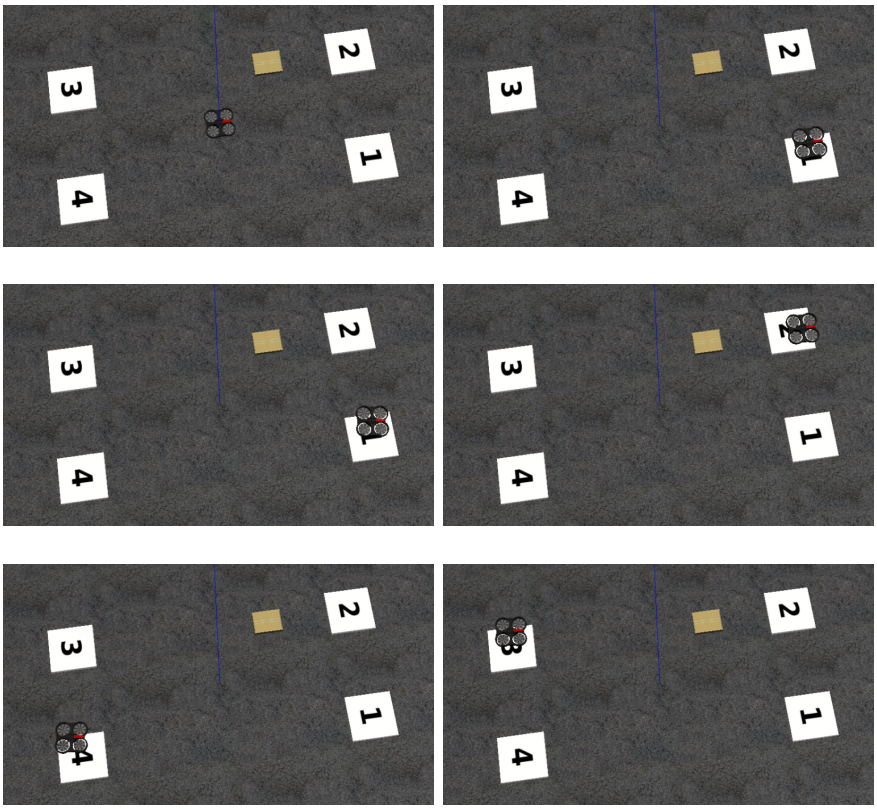

Figure 4.22: (Clock-wise from top-left) Gazebo visualization of Section 4.3.1 waypoints

Chapter 5

Discussion and Future Work

The paper introduced an extended potential field controller augmenting the capability of the traditional potential field controller with the inclusion of a feedback loop including relative velocities between the drone and the target or obstacles. Next, the stability of the ePFC was proven using Lyapunov methods. The developed controller was simulated and evaluated in a MATLAB environment after which actual experiments were conducted. Due to the unavailability of an actual drone and motiuon tracking system, we used the Robot Operating System (ROS) and the associated physics engine Gazebo to more accurately represent the dynamical effects of flying on the quadcopter while executing the developed control law. Due to the absence of moving obstacle or target data, our simulations were restricted to only stationary obstacles and targets. Despite this, the evaluation evidently shows that the extended PF controller performs significantly better than the traditional PF controller when navigating between way points without gravely affecting the power consumption of the system.

Future work may include using the experimental system with completely on board sensing capabilities and controller implementation of the AR Drone 2.0. The velocities of the obstacle and targets may be included in the simulation setup in Gazebo to better reflect all the results obtained by the original author. Since, the extended PF controller is not computationally intensive, it can be tested on other similar drone platforms on Gazebo to evaluate the performance over a wide-range of systems. Furthermore, the SIMULINK toolbox developed for the AR Drone can be made more efficient and bug-free for better deployment and testing of such systems.

Bibliography

[1] A. C. Woods and H. M. La, "A Novel Potential Field Controller for Use on Aerial Robots," in IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 49, no. 4, pp. 665-676, April 2019, doi: 10.1109/TSMC.2017.2702701.

[2] Fabio Morbidi, Roel Cano, David Lara. Minimum-Energy Path Generation for a Quadrotor UAV. IEEE International Conference on Robotics and Automation, May 2016, Stockholm, Sweden. ffhal01276199v2f

[3] R. Pitre, X. Li, and R. Delbalzo, "Uav route planning for joint search and track missions: An information-value approach," Aerospace and Electronic Systems, IEEE Transactions on, vol. 48, no. 3, pp. 2551–2565, July 2012.

[4] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," Proceedings. 1991 IEEE International Conference on Robotics and Automation, 1991, pp. 1398-1404 vol.2, doi: 10.1109/ROBOT.1991.131810.

[5] P. Vílez, N. Certad and E. Ruiz, "Trajectory Generation and Tracking Using the AR.Drone 2.0 Quadcopter UAV," 2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR), 2015, pp. 73-78, doi: 10.1109/LARS-SBR.2015.33.

[6] Howie Choset, Kevin Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki and Sebastian Thrun, "Principles of Robot Motion: Theory, Algorithms, and Implementations"; 2005

[7] Spong, M. W.; Hutchinson, S.; Vidyasagar, "M. Robot Modeling and Control"; John Wiley & Sons: Hoboken, NJ, 2006.

[8] `https://www.mathworks.com/matlabcentral/fileexchange/43719-ar-drone-simulink-development-kit-v1-1`

[9] S. Al Habsi, M. Shehada, M. Abdoon, A. Mashood and H. Noura, "Integration of a Vicon camera system for indoor flight of a Parrot AR Drone," 2015 10th International Symposium on Mechatronics and its Applications (ISMA), 2015, pp. 1-6, doi: 10.1109/ISMA.2015.7373476.

[10] R. Mahony, V. Kumar and P. Corke, "Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor," in IEEE Robotics & Automation Magazine, vol. 19, no. 3, pp. 20-32, Sept. 2012, doi: 10.1109/MRA.2012.2206474.

[11] L. Merino, F. Caballero, J. Martnez-de Dios, J. Ferruz, and A. Ollero, "A coop- erative perception system for multiple uavs: Application to automatic detection of forest fires," Journal of Field Robotics, vol. 23, no. 3-4, pp. 165–184, 2006.

[12] `https://alliance.seas.upenn.edu/~meam535/cgi-bin/pmwiki/uploads/Main/Newton_Euler_11.pdf`

[13] `https://github.com/tahsinkose/sjtu-drone`

[14] `http://wiki.ros.org/tum_simulator`

[15] ROS Documentation - `http://wiki.ros.org/`

Appendix A

ROS and Gazebo Codes

A.1 ROS Publisher and Subscriber code to make AR Drone track waypoints

```
# potential_field_control.py
#!/usr/bin/env python
from numpy.core.fromnumeric import size
import rospy
from geometry_msgs.msg import Twist
from geometry_msgs.msg import Pose
from std_msgs.msg import Empty
from tf.transformations import euler_from_quaternion
import numpy as np
import matplotlib.pyplot as plt

p_d = np.array([0.0, 0.0])
v_d = np.array([0.0, 0.0])

#***** Setting the target and obstacle X-Y coordinates *******
# Uncomment for the corresponding X-Y plot of the Figures in the
    paper
# For Figures 8 & 9
p_t = np.matrix((np.array((2.5,-1.0)), np.array((2.5,1.0)),
    np.array((-2.5,1.0)), np.array((-2.5,-1.0)),
    np.array((2.5,-1.0)))).T
p_o = np.matrix((np.array([1.0, 1.0]))).T

# For Figure 10
# p_t = np.matrix((np.array([6.0,0.0]), np.array([0.0,0.0]))).T
# p_o = np.matrix((np.array([1.2,0.0]), np.array([2.5,0.6]),
    np.array([3.0,0.0]), np.array([4.2,0.5]),
    np.array([4.7,-0.5]))).T

# For Figure 14
# p_t = np.matrix((np.array((4.2,0.0)))).T
```

```
# p_o = np.matrix(np.empty).T

# For Figure 15
# p_t = np.matrix((np.array([5.2,0.0]))).T
# p_o = np.matrix((np.array([2.5,0.0]))).T

# For Figure 17 & 19
# p_t = np.matrix((np.array([-1.5,-0.5]), np.array([1.5,-0.5]),
    np.array([1.5,0.5]), np.array([-1.5,0.5]),
    np.array([-1.5,-0.5]))).T
# p_o = np.matrix(np.array([0.0,0.4])).T


num_tgt = np.shape(p_t)[1] # number of targets
num_obs = np.shape(p_o)[1] # number of obstacles
if (np.shape(p_o)[0] is 1) and (np.shape(p_o)[1] is 1):
    num_obs = 0


# Callback Function to receive the drone's pose data
def pose_callback(data):
    global p_d
    global roll, pitch, yaw

    # Array to save the received drone X and Y positions
    p_d = np.array([data.position.x, data.position.y])

    # Array to save the received drone quaternion orientations
    drone_angles = [data.orientation.x, data.orientation.y,
        data.orientation.z, data.orientation.w]

    # converting quaternion to Euler angles
    (roll, pitch, yaw) = euler_from_quaternion (drone_angles)


# Callback Function to receive the drone's velocity data
def vel_callback(data):
    global v_d
    # Array to save the received drone X and Y linear velocities
    v_d = np.array([data.linear.x, data.linear.y])


# Function to compute the extended potential field velocities
def ex_PFC():
```

```
global v_d, p_d

# Positive scalars of exPFC controller
lambda_1 = 0.6
lambda_2 = 0.1
eta_1 = 0.02

control_vel = Twist()

i_tgt = 0 # target index
p_avoid = 0.8 # obstacle avoiding range

rate = rospy.Rate(50) # 50 Hz

while ((not rospy.is_shutdown())):
    rospy.loginfo("Heading to Waypiont No. {"+str(i_tgt+1)+"}")
    if lambda_2 is not 0.0:
        plt.title('Extended PFC - lambda1: ' + str(lambda_1)+',
            lambda2: ' + str(lambda_2) + ', eta1: ' + str(eta_1))
    else:
        plt.title('Traditional PFC - lambda1: ' +
            str(lambda_1)+', lambda2: ' + str(lambda_2) + ',
            eta1: ' + str(eta_1))

    # Comment for Fig 14, 15 or 19 in paper
    plt.xlabel(' X Position [m]')
    plt.ylabel(' Y Position [m]')

    # Uncomment for Fig 19 in paper
    # plt.xlabel(' Time [s]')
    # plt.ylabel(' Error [m]')

    # Uncomment for Fig 14, 15
    # plt.ylabel(' Relative Distance to Target [m]')

    # Uncomment for Fig 15
    # plt.ylabel(' Relative Distance to Obstacle [m]')

    # drone is considered to reach target if it is <= 0.1m of
        the target,
    while(np.linalg.norm(p_d - (p_t[:,i_tgt]).T) > 0.1):

        v_pfc_att = -lambda_1*(p_d - p_t[:,i_tgt].T) #
            attractive potential velocity
```

```
v_pfc_rep = np.array([0,0]) # repulsive potential
    velocity

for i in range(0,num_obs):
    if (np.linalg.norm(p_d - (p_o[:,i]).T) <= p_avoid):
        v_pfc_rep = (eta_1/(np.linalg.norm((p_d -
            (p_o[:,i]).T)))**4) * (p_d - (p_o[:,i]).T)
    else:
        v_pfc_rep = v_pfc_rep + np.array([0,0])

v_pfc = v_pfc_att + v_pfc_rep
v_epfc_iner = v_pfc - lambda_2*(v_d)

# rotation matrix to transform from inertial frame to
    body frame
R = np.matrix([[np.cos(yaw), np.sin(yaw)],
    [-np.sin(yaw), np.cos(yaw)]])

v_epfc_body = (np.matmul(R, v_epfc_iner.T)).T # array
v_epfc_body = np.clip(v_epfc_body, -1, 1) # saturating
    the calculated velocity to [-1,1]

# publishing the calculated velocity to the drone
    simulation
control_vel.linear.x = v_epfc_body[0,0]
control_vel.linear.y = v_epfc_body[0,1]
control_vel.linear.z = 0
control_vel.angular.x = 0
control_vel.angular.y = 0
control_vel.angular.z = 0
pfc_pub.publish(control_vel)

# Comment for Fig 14, 15, or 19 in paper
plt.scatter(p_d[0], p_d[1])

# Uncomment for Fig 19 in paper
# plt.scatter(rospy.get_time(), p_d[0] -
    (p_t[0,i_tgt]).T)

# Uncomment for Fig 14, 15 (ploting target distance) in
    paper
# plt.scatter(rospy.get_time(), (p_t[0,i_tgt]).T -
    p_d[0])
```

```
            # Uncomment for Fig 15 (ploting obstacle distance) in
                paper
            # plt.scatter(rospy.get_time(),
                np.linalg.norm((p_o[:,0]).T - p_d))

            rate.sleep()
            # rospy.spin()

        control_vel.linear.x = 0.0
        control_vel.linear.y = 0.0
        control_vel.linear.z = 0.0
        control_vel.angular.x = 0.0
        control_vel.angular.y = 0.0
        control_vel.angular.z = 0.0
        pfc_pub.publish(control_vel)

        # Uncomment for Fig 19 in paper
        # plt.scatter(rospy.get_time(), p_d[0] - (p_t[0,i_tgt]).T)

        # Uncomment for Fig 14, 15 (ploting target distance) in paper
        # plt.scatter(rospy.get_time(), (p_t[0,i_tgt]).T - p_d[0])

        # Uncomment for Fig 15 (ploting obstacle distance) in paper
        # plt.scatter(rospy.get_time(), (p_o[0,0]).T - p_d[0])

        i_tgt = i_tgt + 1

        # Comment for Fig 14, 15 or 19 in paper
        plt.scatter(p_d[0], p_d[1])


        if(i_tgt >= num_tgt):
            # Comment the below 2 lines for Fig 14, 15 or 19 in paper
            for i in range(0,num_obs):
                plt.scatter(p_o[0,i], p_o[1,i], marker='x',
                    linewidths=5)
            plt.show()
            break

if __name__ == '__main__':
    try:
        rospy.init_node('ePFC', anonymous=False)

        # Publisher to initially make the drone to takeoff
```

```
        takeoff_pub = rospy.Publisher('/drone/takeoff', Empty,
            queue_size=10)

        # Loop until the takeoff topic has a connection
        while(takeoff_pub.get_num_connections() <= 0):
            pass

        takeoff_pub.publish(Empty())

        pose_sub = rospy.Subscriber("/drone/gt_pose", Pose,
            pose_callback)
        vel_sub = rospy.Subscriber("/drone/gt_vel", Twist,
            vel_callback)
        pfc_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)

        # Loop until there are publishers and subscribers to the
            above 3 topics
        while(pose_sub.get_num_connections() <= 0 or
            vel_sub.get_num_connections() <= 0 or
            pfc_pub.get_num_connections() <= 0):
            pass

        ex_PFC()

    except rospy.ROSInterruptException:
        pass
```

A.2 Launch file to open Gazebo simulation

```
<launch>
 <!-- ardrone_waypoint.launch -->
 <!-- these are the arguments you can pass this launch file, for
     example paused:=true -->
  <arg name="paused" value="false"/>
  <arg name="verbose" value="true"/>
  <arg name="world_name" default="$(find
      sjtu_drone)/worlds/waypoint_tracking.world"/>

  <!-- launch the custom world -->
   <include file="$(find gazebo_ros)/launch/empty_world.launch" >
       <arg name="paused" value="$(arg paused)"/>
       <arg name="world_name" value="$(arg world_name)"/>
```

```
      <arg name="verbose" value="$(arg verbose)"/>
  </include>

  <!-- send robot urdf to param server -->
  <param name="robot_description" command="cat '$(find
      sjtu_drone)/urdf/sjtu_drone.urdf'" />

  <!-- TF node that gives the static transformation between the
      world map and the base link of your robot. Change /base_link
      to the /root link name of your robot -->
  <node pkg="tf" type="static_transform_publisher"
      name="map_to_base" args="0 0 0 0 0 0 1 /map /base_link 10" />


  <!-- push robot_description to factory and spawn robot in gazebo
      at the origin, change x,y,z arguments to spawn in a different
      position -->
  <node name="spawn_gazebo_model" pkg="gazebo_ros"
      type="spawn_model" args="-urdf -param robot_description
      -model sjtu_drone -x 0 -y 0 -z 0"
    respawn="false" output="screen" />
</launch>
```

A.3 Custom generated Gazebo World to visualize way points

```
<!-- waypoint_tracking.world -->
<sdf version='1.6'>
  <world name='default'>
    <light name='sun' type='directional'>
      <cast_shadows>1</cast_shadows>
      <pose frame=''>0 0 10 0 -0 0</pose>
      <diffuse>0.8 0.8 0.8 1</diffuse>
      <specular>0.2 0.2 0.2 1</specular>
      <attenuation>
        <range>1000</range>
        <constant>0.9</constant>
        <linear>0.01</linear>
        <quadratic>0.001</quadratic>
      </attenuation>
      <direction>-0.5 0.1 -0.9</direction>
    </light>
```

```
<model name='ground_plane'>
  <static>1</static>
  <link name='link'>
    <collision name='collision'>
      <geometry>
        <plane>
          <normal>0 0 1</normal>
          <size>100 100</size>
        </plane>
      </geometry>
      <surface>
        <contact>
          <collide_bitmask>65535</collide_bitmask>
          <ode/>
        </contact>
        <friction>
          <ode>
            <mu>100</mu>
            <mu2>50</mu2>
          </ode>
          <torsional>
            <ode/>
          </torsional>
        </friction>
        <bounce/>
      </surface>
      <max_contacts>10</max_contacts>
    </collision>
    <visual name='visual'>
      <cast_shadows>0</cast_shadows>
      <geometry>
        <plane>
          <normal>0 0 1</normal>
          <size>100 100</size>
        </plane>
      </geometry>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Grey</name>
        </script>
      </material>
    </visual>
    <self_collide>0</self_collide>
    <enable_wind>0</enable_wind>
```

```
    <kinematic>0</kinematic>
  </link>
</model>
<gravity>0 0 -9.8</gravity>
<magnetic_field>6e-06 2.3e-05 -4.2e-05</magnetic_field>
<atmosphere type='adiabatic'/>
<physics name='default_physics' default='0' type='ode'>
  <max_step_size>0.001</max_step_size>
  <real_time_factor>1</real_time_factor>
  <real_time_update_rate>1000</real_time_update_rate>
</physics>
<scene>
  <ambient>0.4 0.4 0.4 1</ambient>
  <background>0.7 0.7 0.7 1</background>
  <shadows>1</shadows>
</scene>
<wind/>
<spherical_coordinates>
  <surface_model>EARTH_WGS84</surface_model>
  <latitude_deg>0</latitude_deg>
  <longitude_deg>0</longitude_deg>
  <elevation>0</elevation>
  <heading_deg>0</heading_deg>
</spherical_coordinates>
<model name='asphalt_plane'>
  <static>1</static>
  <link name='link'>
    <collision name='collision'>
      <geometry>
        <box>
          <size>20 20 0.1</size>
        </box>
      </geometry>
      <max_contacts>10</max_contacts>
      <surface>
        <contact>
          <ode/>
        </contact>
        <bounce/>
        <friction>
          <torsional>
            <ode/>
          </torsional>
          <ode/>
        </friction>
```

```
      </surface>
    </collision>
    <visual name='visual'>
      <cast_shadows>0</cast_shadows>
      <geometry>
        <box>
          <size>20 20 0.1</size>
        </box>
      </geometry>
      <material>
        <script>
          <uri>model://asphalt_plane/materials/scripts</uri>
          <uri>model://asphalt_plane/materials/textures</uri>
          <name>vrc/asphalt</name>
        </script>
      </material>
    </visual>
    <self_collide>0</self_collide>
    <enable_wind>0</enable_wind>
    <kinematic>0</kinematic>
  </link>
  <pose frame=''>-0.071622 0.034885 0 0 -0 0</pose>
</model>
<model name='number1'>
  <pose frame=''>0.847947 -5.36201 0.4 0 -0 0</pose>
  <static>1</static>
  <link name='link'>
    <visual name='visual'>
      <geometry>
        <mesh>
          <uri>model://number1/meshes/number.dae</uri>
        </mesh>
      </geometry>
      <material>
        <script>
          <uri>model://number1/materials/scripts</uri>
          <uri>model://number1/materials/textures</uri>
          <name>Number/One</name>
        </script>
      </material>
    </visual>
    <self_collide>0</self_collide>
    <enable_wind>0</enable_wind>
    <kinematic>0</kinematic>
  </link>
```

```
</model>
<model name='number2'>
  <pose frame=''>7.49357 -3.51758 0.4 0 -0 0</pose>
  <static>1</static>
  <link name='link'>
    <visual name='visual'>
      <geometry>
        <mesh>
          <uri>model://number1/meshes/number.dae</uri>
        </mesh>
      </geometry>
      <material>
        <script>
          <uri>model://number2/materials/scripts</uri>
          <uri>model://number2/materials/textures</uri>
          <name>Number/Two</name>
        </script>
      </material>
    </visual>
    <self_collide>0</self_collide>
    <enable_wind>0</enable_wind>
    <kinematic>0</kinematic>
  </link>
</model>
<model name='number3'>
  <pose frame=''>2.03543 1.01702 0.4 0 -0 0</pose>
  <static>1</static>
  <link name='link'>
    <visual name='visual'>
      <geometry>
        <mesh>
          <uri>model://number1/meshes/number.dae</uri>
        </mesh>
      </geometry>
      <material>
        <script>
          <uri>model://number3/materials/scripts</uri>
          <uri>model://number3/materials/textures</uri>
          <name>Number/Three</name>
        </script>
      </material>
    </visual>
    <self_collide>0</self_collide>
    <enable_wind>0</enable_wind>
    <kinematic>0</kinematic>
```

```
    </link>
  </model>
  <model name='number4'>
    <pose frame=''>4.85642 -5.66893 0.4 0 -0 0</pose>
    <static>1</static>
    <link name='link'>
      <visual name='visual'>
        <geometry>
          <mesh>
            <uri>model://number1/meshes/number.dae</uri>
          </mesh>
        </geometry>
        <material>
          <script>
            <uri>model://number4/materials/scripts</uri>
            <uri>model://number4/materials/textures</uri>
            <name>Number/Four</name>
          </script>
        </material>
      </visual>
      <self_collide>0</self_collide>
      <enable_wind>0</enable_wind>
      <kinematic>0</kinematic>
    </link>
  </model>
  <model name='cardboard_box'>
    <pose frame=''>0.215846 -0.194719 0.15 0 -0 0</pose>
    <link name='link'>
      <inertial>
        <mass>2</mass>
        <inertia>
          <ixx>0.0416667</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>0.0566667</iyy>
          <iyz>0</iyz>
          <izz>0.0683333</izz>
        </inertia>
        <pose frame=''>0 0 0 0 -0 0</pose>
      </inertial>
      <collision name='collision'>
        <geometry>
          <box>
            <size>0.5 0.4 0.3</size>
          </box>
```

```
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>1</mu>
            <mu2>1</mu2>
          </ode>
          <torsional>
            <ode/>
          </torsional>
        </friction>
        <contact>
          <ode>
            <kp>1e+07</kp>
            <kd>1</kd>
            <min_depth>0.001</min_depth>
            <max_vel>0.1</max_vel>
          </ode>
        </contact>
        <bounce/>
      </surface>
      <max_contacts>10</max_contacts>
    </collision>
    <visual name='visual'>
      <pose frame=''>0 0 -0.15 0 -0 0</pose>
      <geometry>
        <mesh>
          <uri>model://cardboard_box/meshes/cardboard_box.dae</uri>
          <scale>1.25932 1.00745 0.755591</scale>
        </mesh>
      </geometry>
    </visual>
    <self_collide>0</self_collide>
    <enable_wind>0</enable_wind>
    <kinematic>0</kinematic>
  </link>
</model>
<state world_name='default'>
  <sim_time>205 817000000</sim_time>
  <real_time>206 352901460</real_time>
  <wall_time>1637799143 371111227</wall_time>
  <iterations>205817</iterations>
  <model name='asphalt_plane'>
    <pose frame=''>-0.071622 0.034885 0 0 -0 0</pose>
    <scale>1 1 1</scale>
```

```
    <link name='link'>
      <pose frame=''>-0.071622 0.034885 0 0 -0 0</pose>
      <velocity>0 0 0 0 -0 0</velocity>
      <acceleration>0 0 0 0 -0 0</acceleration>
      <wrench>0 0 0 0 -0 0</wrench>
    </link>
  </model>
  <model name='cardboard_box'>
    <pose frame=''>0.999898 0.999936 -0.026247 -0.001209
        0.000967 3.13999</pose>
    <scale>1 1 1</scale>
    <link name='link'>
      <pose frame=''>0.999898 0.999936 -0.026247 -0.001209
          0.000967 3.13999</pose>
      <velocity>0 0 0 0 -0 0</velocity>
      <acceleration>-0 0 0.001652 0 -0 0</acceleration>
      <wrench>-0 0 0.003304 0 -0 0</wrench>
    </link>
  </model>
  <model name='ground_plane'>
    <pose frame=''>0 0 0 0 -0 0</pose>
    <scale>1 1 1</scale>
    <link name='link'>
      <pose frame=''>0 0 0 0 -0 0</pose>
      <velocity>0 0 0 0 -0 0</velocity>
      <acceleration>0 0 0 0 -0 0</acceleration>
      <wrench>0 0 0 0 -0 0</wrench>
    </link>
  </model>
  <model name='number1'>
    <pose frame=''>2.5 -1 -0.3 0 -0 3.14</pose>
    <scale>1 1 1</scale>
    <link name='link'>
      <pose frame=''>2.5 -1 -0.3 0 -0 3.14</pose>
      <velocity>0 0 0 0 -0 0</velocity>
      <acceleration>0 0 0 0 -0 0</acceleration>
      <wrench>0 0 0 0 -0 0</wrench>
    </link>
  </model>
  <model name='number2'>
    <pose frame=''>2.5 1 -0.3 0 -0 3.14</pose>
    <scale>1 1 1</scale>
    <link name='link'>
      <pose frame=''>2.5 1 -0.3 0 -0 3.14</pose>
      <velocity>0 0 0 0 -0 0</velocity>
```

```
        <acceleration>0 0 0 0 -0 0</acceleration>
        <wrench>0 0 0 0 -0 0</wrench>
      </link>
    </model>
    <model name='number3'>
      <pose frame=''>-2.5 1 -0.3 0 -0 3.14</pose>
      <scale>1 1 1</scale>
      <link name='link'>
        <pose frame=''>-2.5 1 -0.3 0 -0 3.14</pose>
        <velocity>0 0 0 0 -0 0</velocity>
        <acceleration>0 0 0 0 -0 0</acceleration>
        <wrench>0 0 0 0 -0 0</wrench>
      </link>
    </model>
    <model name='number4'>
      <pose frame=''>-2.5 -1 -0.3 0 -0 3.14</pose>
      <scale>1 1 1</scale>
      <link name='link'>
        <pose frame=''>-2.5 -1 -0.3 0 -0 3.14</pose>
        <velocity>0 0 0 0 -0 0</velocity>
        <acceleration>0 0 0 0 -0 0</acceleration>
        <wrench>0 0 0 0 -0 0</wrench>
      </link>
    </model>
    <light name='sun'>
      <pose frame=''>0 0 10 0 -0 0</pose>
    </light>
  </state>
  <gui fullscreen='0'>
    <camera name='user_camera'>
      <pose frame=''>0.111856 -2.66715 12.4309 0 1.31164
          1.4242</pose>
      <view_controller>orbit</view_controller>
      <projection_type>perspective</projection_type>
    </camera>
  </gui>
 </world>
</sdf>
```

Appendix B

MATLAB Function Codes

B.1 Script to setup simulation environment and variables

```
%  This script is based on the setupWPTrackingSim.m script in the
    toolbox

%%
%  Cleaning workspace
bdclose all;
clear all;
clc

%%
% Adding ARDrone library path
addpath ../;
%% Simulation parameters

% Flight management system sample time. This is the sample time at
    which
% the control law is executed.
FMS.Ts = 0.065;

% Time delay due to communication between drone and host computer
timeDelay = FMS.Ts*4;


%% Vehicle model based on linear dynamics

% Loading state space representation of vehicle dynamics
setupARModel;

%%
% Loading list of waypoints
waypoints = getWaypoints() ;
```

```
% Loading list of obstacles
obstacles = getObstaclepoints() ;
nObst = size(obstacles,2);

%%
% Simulation time
simDT = 0.005 ;

%%
% Loading Simulink model of Extended Potential Field
% controller for AR Drone 2.0
potential_field_controller ;
```

B.2 Function to compute traditional PFC velocity

```
function v_PFC = fcn(p_d , p_t, p_o, N_obs)
    %% Calculating PFC velocity
    % Inputs: drone position (X,Y), target positions (X,Y),
    %         obstacle positions (X,Y), Number of Obstacles
    % Output: Computed PFC velocity in Inertial frame(m/s)

    p_avoid = 0.8; % Obstacle avoidance range (m)

    % Positive Scalar constants
    lamda_1 = 0.4; % Tuned values: 0.3 & 0.01
    eta_1 = 0.03;

    v_att_PFC = -lamda_1 * (p_d - p_t); % attractive potential
        velocity
    v_rep_PFC = [0 ; 0]; % repulsive potential velocity

    for i = 1:N_obs
       if ((norm(p_d - p_o(:,i)) <= p_avoid) && (N_obs ~= 0))
            v_rep_PFC = v_rep_PFC + (eta_1/(norm(p_d-p_o(:,i)))^4) *
                (p_d - p_o(:,i));
       else
            v_rep_PFC = v_rep_PFC + [0; 0];
       end
    end
```

```
    % Total PFC control velocity
    v_PFC = v_att_PFC + v_rep_PFC;
end
```

B.3 Function to compute extended PFC velocity

```
function v_ePFC = fcn(v_PFC, p_d , p_o, v_d, v_o, p_do_dot_mag,
    N_obs)
    %% Calculating ePFC velocity
    % Inputs: PFC velocity (m/s), drone position (X,Y),
    %         obstacle positions (X,Y), drone velocities (X,Y)
    %         obstacle velocities (X,Y), |p_do|_dot, No. of Obstacles
    % Output: Computed extended PFC velocity in Inertial frame (m/s)

    % Positive scalar constants
    lamda_2 = 0.3;
    eta_2 = 0.06;
    eta_3 = 0.06;

    v_att_ePFC = -lamda_2 * (v_d); % Here, v_target = 0
    v_rep1_ePFC = [0 ; 0];
    v_rep2_ePFC = [0 ; 0];

    for i = 1:N_obs
       if (((norm(v_o(:,i)) ~= 0)) && (N_obs ~= 0))
            v_rep1_ePFC = v_rep1_ePFC +
                (eta_2/(norm(v_d-v_o(:,i)))^4) * (v_d - v_o(:,i));
       else
           v_rep1_ePFC = v_rep1_ePFC + [0;0]; % if v_obstacle = 0,
               no repulsive effect
       end

       if ((p_do_dot_mag(i) < 0) && (N_obs ~= 0))
           v_rep2_ePFC = v_rep2_ePFC + (eta_3 * p_do_dot_mag(i)) *
               ((p_d - p_o(:,i))/norm(p_d - p_o(:,i)));
       else
           v_rep2_ePFC = v_rep2_ePFC + [0;0]; % if |p_do_dot| >= 0,
               no repulsive effect
       end
    end

    % Total ePFC control velocity
```

```
    v_ePFC = v_PFC + v_att_ePFC + v_rep1_ePFC - v_rep2_ePFC;

end
```

B.4 Function to compute time rate of change of magnitude of p_do

```
function p_do_mag = fcn(p_d, p_o, nObst)

    p = zeros(1, nObst);

    if (nObst ~= 0)
        for i = 1:nObst
           p(i) = norm(p_d - p_o(:,i));
        end
    end

    p_do_mag = p;

end
```

B.5 Function to generate obstacle points

```
function [ obstacle ] = getObstaclepoints()
    %GETWAYPOINTS Generates a list of obstacle points for the
        ARDrone
    %   Each obstacle point is a column vector that contains the
        obstacle's X & Y positions and the obstacles X & Y
        velocities.
    %   The list of obstacles is created by combining the column
        vectors.



    % Edit the following entries for k =1,2,...,nObsPoints
    % obstaclesListARDrone(:,k) = [ x_o (m); y_o (m); vx_o (m/s);
        vy_o (m/s)]

    % For Figure 9
```

```
    nObsPoints = 1;
    obstaclesListARDrone = zeros(2,nObsPoints);
    obstaclesListARDrone(:,1) = [1.0; 1.0] ;

    obstacle = obstaclesListARDrone;

end
```