ABSTRACT

Integration of Potential Field Theory and Proportional Navigation Theory to Autonomously Guide an Unmanned Aerial Vehicle

Patrick L. Friudenberg, M.S.E.C.E.

Mentor: Scott Koziol, Ph.D.

Industrial robotics, military, surveying, and delivery applications have laid a foundation for research into full autonomy of machines, including Unmanned Aerial Vehicles (UAV). This thesis supports this research by surveying the methods used to guide UAVs, and developing a new method by combining potential fields, typically used for obstacle avoidance, and proportional navigation, a popular missile guidance algorithm. The new algorithm modifies the old algorithms to allow a UAV to track an optimal path to, and rendezvous with, a moving target while avoiding obstacles in its path. A model for a quad rotor style UAV is developed and controlled using feedback linearization. A simulator is built for deploying a number of environments and taking performance measurements. Aspects of the hardware implementation are introduced.

Integration of Potential Field Theory and Proportional Navigation Theory to Autonomously Guide an Unmanned Aerial Vehicle

by

Patrick L. Friudenberg, B.S.E.C.E.

A Thesis

Approved by the Department of Electrical and Computer Engineering

_____

Kwang Lee, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of

Master of Science in Electrical and Computer Engineering

Approved by the Thesis Committee

_____

Scott Koziol, Ph.D., Chairperson

_____

Kwang Lee, Ph.D.

_____

Brian Garner, Ph.D.

Accepted by the Graduate School

December 2015

_____

J. Larry Lyon, Ph.D., Dean

Page bearing signatures is kept on file in the Graduate School.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I owe a thank you to the team at the Robotics and Embedded Systems Lab for their help in research and development of the hardware. Thank you to my advisor, Dr. Scott Koziol, for the opportunity to do research in a field I am passionate about, and to Dr. Brian Garner for help in understanding the mechanics of a quadrotor. I would also like to thank Dr. John Davis in the mathematics department for help in structuring the formulation of the problem, and for reviewing some of the mathematics required for completion of this thesis. Most importantly, thank you to my family for their patience and understanding during this endeavor.

CHAPTER ONE

Introduction

An unmanned aerial system (UAS) is an unmanned aircraft and all of the associated support equipment, control station, data links, telemetry, communications, and navigation equipment, etc. necessary to operate an unmanned aircraft. A subsystem within the UAS is the aerial robot itself, referred to as an Unmanned Aerial Vehicle (UAV). Controlling the UAV may be done by remote human control, or it may be by algorithms that give it full or partial autonomy. As of July, 2015, there were approximately 50 organizations developing and producing approximately 155 unmanned aircraft designs for use in military, commercial, and recreational applications. The most common uses are military applications for surveillance and weapon delivery in Iraq and Afghanistan, and on the Mexico/US Border [1]. Commercial use is somewhat limited due to FAA regulations concerning shared airspace with commercial airliners, but some exceptions are given when the need for a UAS is apparent. Such is the case with BP Oil for use in surveying the vast landscape of Alaska for drilling oil [1]. Recreational and educational aircraft are supplied by companies such as 3D Robotics, from whom the test aircraft for our project was acquired.

The International Aerial Robotics Competition (IARC) is a self-proclaimed "technology sport" whose motivation is best represented by the competition's stated purpose:

> The primary purpose of the International Aerial Robotics Competition (IARC) has been to "move the state-of-the-art in aerial robotics forward" through the creation of significant and useful mission challenges that are 'impossible' at the time they are proposed, with the idea that when the aerial robotic behaviors called for in the mission are eventually demonstrated, the technology will have been advanced for the benefit of the world. [2]

The competition is international with venues held in the Unites States and China. Adhering to their decree of only proposing challenges that are impossible at the time of proposal, the IARC presents mission challenges that require the use of new technologies to meet difficult specifications such as requiring the vehicle to act completely autonomously, without any human interference. On average a challenge requires over three years' time to be completed, meaning that in twenty two years' time, only six missions have been successfully completed. The first round of mission seven was completed in August of 2014, a picture of which can be seen in Figure 1.1. To provide incentive for institutions to participate, the organization offers significant cash rewards along with compelling accolades such as journal publications.

Early in the life of the competition, GPS technology was fairly new to the marketplace, and therefore, missions one and two consisted of using GPS technology without the use of inertial systems and differential GPS technology, respectively, to perform object retrieval and deposit. Mission three advanced the field of computer vision by having a UAV perform a search and rescue mission that required the UAV to differentiate between injured survivors and the dead while avoiding damaging obstacles such as fire and water geysers in a cluttered, smoke obscured environment. The ability to fly long distances and find a specific building, in a specific village, and deposit a sub-robot was demonstrated in mission four. Mission five and six extended and refined mission four by having the sub-robot use simultaneous localization and mapping (SLAM)

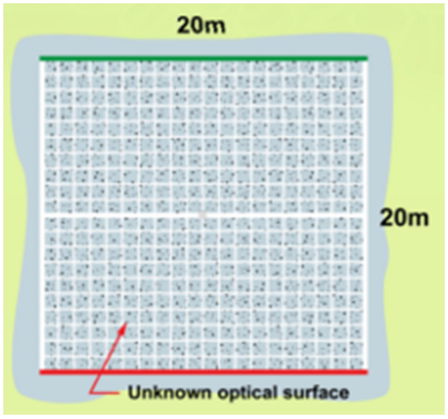
techniques to map the unknown interior of the building, and demonstrate the ability to read and interpret printed directions on the walls to locate a specific room, retrieve a specified object from the room, and carry it from the building.

Figure 1.1 IARC Venues

Mission 7 proposes a challenge which moves away from stationary scenarios to a dynamic field for both obstacles that the UAV must avoid, and friendly robots with which the UAV must interact. The three main objectives of the mission are to demonstrate interaction between UAV and moving objects, navigate in an environment without the use of GPS or SLAM techniques, and interaction between competing autonomous aerial robots[2].

As depicted in Figure 1.2(a), mission seven is to be held in a twenty meter by twenty meter arena with no major static objects available for assistance in spatial awareness for the aerial robot via a common SLAM algorithm. At the heart of the mission, there are a total of ten iRobots that are positioned in a circle facing outwards, as

depicted in Figure 1.2(c). The idea is to get as many of the iRobots to one end of the arena as possible within the allotted ten minute time frame. Once time starts, the iRobots will move in a path that can change up to twenty degrees clockwise every five seconds, and they will reverse direction every twenty seconds.



(a) IARC arena

(b) IARC obstacles

(c) IARC iRobots

Figure 1.2 Elements of IARC Mission Seven

The ability of the UAV to control one of the iRobots is obtained through the use of magnets. A magnet is placed on the top of the iRobot that triggers it to change its path by forty five degrees clockwise whenever the magnet is touched by a second magnet fixed to the aerial robot. Control of the iRobot can also be obtained by bumping its front bumper which will cause it to reverse directions. This must be done all while insuring no

contact is made with "obstacle robots", Figure 1.2(b), moving in a circular path around the arena. For each iRobot that makes it to the appointed side of the arena, the team will get two thousand points, while losing one thousand points for every iRobot that crosses any of the three other arena boundaries. These points, combined with other points for things such as workmanship, a journal paper, and team t-shirts, are totaled to determine the winning teams. The winning teams will receive a thirty thousand dollar cash reward and a chance to enter the second part of mission 7, mission 7b. The judges will determine who the "best of the best" is in 7a and grant an invitation to them to participate in 7b. 7b is the same as 7a except for two aerial robots will go head to head in the same run to try and get as many of the iRobots to their respective ends of the arena. The invitation to 7b is not based solely on points obtained in 7a, but also on the judge's opinions of the UAV's obstacle avoidance capabilities. This requirement is incorporated to keep the two team's aerial robots from colliding while running the mission at the same time. Even if a team is not invited to 7b, any team's robot that demonstrates the skills required to make it to 7b is awarded a cash prize of a thousand dollars. An award amount for winning 7b is not specified, however the competition does state that it will be larger than the amount awarded for 7a, making the total a team can win over sixty thousand dollars [2].

A quad rotor style of UAV has been chosen to perform at the IARC due to its ability to hover, and the ease of housing the physical apparatuses to prevent damage caused by a collision in a cluttered environment. In the next section the general structure of a UAS system is covered with respect to a quad rotor. The particular UAV chosen, 3D Robotics' X8, is covered on page 69 in the section 3D Robotic's X8. On page 24, the section Potential Fields-Proportional Navigation Guidance Law develops an algorithm

for making navigation decisions in real time, which is the focus of this thesis. Simulating the Algorithm on page 56 describes techniques used to simulate the environment for testing the algorithm, and the results are discussed on page 79.

UAS Subsystem Structure

General paradigms describing the chain of processes an autonomous robot experiences or performs have been developed. The inception of pick-and-place industrial robotics fostered the development of the "sense-think-act" paradigm for autonomous robotic movements. With the development of multi-robot tasks and superior telemetry systems, the appendage of communication to the "sense-think-act" paradigm has de facto been added according to [3]. In this thesis, the think part of the paradigm is split into two categories, mission level decisions and navigation decisions. This paradigm is related to nature in the sense that biological creatures tend to develop strategy in order to accomplish a particular task, but basic navigation from one point to another through most environments comes rather quickly without much thought. In the case of the IARC, the UAS's decision algorithm to determine which iRobot will be the optimal agent to control to complete the mission is separate from the algorithm used to cautiously navigate to that intended iRobot through an obstacle filled environment. The acting portion of the paradigm is encapsulated in the control structure of the UAS. These four subsystems, Sensors, Mission Level Decisions, Navigation Decisions, and the Control Structure, makeup the architect of the UAV. A diagram of the subsystems is shown in Figure 1.3. The World block represents everything in the physical area that the aerial robot may encounter during a competition including the arena itself, the ten iRobot robots, four iRobots with tall apparatuses fixed to them to serve as obstacles, and any markings on the

arena floor, again as depicted in Figure 1.2. As we will see later, these items will have to be modeled, simulated, and analyzed before actually placing the algorithms onto the hardware of the aerial robot.

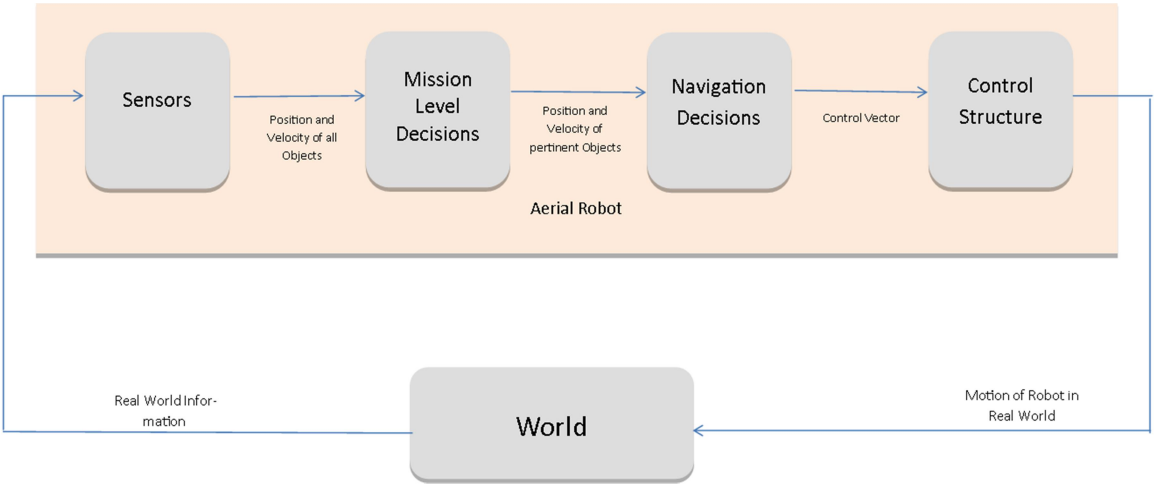


Figure 1.3 Planning Structure for Aerial Robot

Sensors

Robotic sensors are typically classified as either contact or noncontact sensors. Contact sensors include devices that measure touch, proximity, and slip. Noncontact sensors are comprised of optical, magnetic, capacitive, resistive, ultrasound and air pressure [4]. The capabilities of sensors have reached a scope far beyond that of the typical senses described in biological creatures: touch, sight, smell, hearing, and taste. For example, robotic sensors can detect electromagnetic waves that are outside the frequency range that humans are able to detect with eyes or ears. To further accentuate the capabilities of robotic sensors, the idea of sight can be extended beyond just a typical optical sensor that works like an eye, to sensors such as radar, which is capable of seeing through walls, or infrared, which is capable of seeing temperature.

To give a robot full autonomy the sensor system may be thought of as a system of three subsystems: sensing for the purpose of the mission, sensing to survive in the mission environment, and sensing one's own configuration [5]. The aerial robot can approximate its own configuration, location and orientation, through the use of an Inertial Measurement Unit (IMU). An IMU typically consists of three components: a 3 axis gyroscope to measure rotational acceleration, a 3 axis accelerometer for measuring linear acceleration, and a magnetometer to assist with calibration of the unit. These components use vibrations caused by acceleration of the robot to determine the magnitude and direction of the acceleration [6]. Taking into account the initial position and velocity of the robot and integrating the measured accelerations gives the robot a since of where it is and how fast it is going. This method, known as 'dead reckoning', has inherent issues due to the error of integral calculations. Any error in the UAV's position grows linearly relative to the initial velocity error estimate, quadratically with uncorrected bias, and at a cubic rate with attitude error [7].

To give an accurate position and velocity, the use of a device that can give a measurement based on features that are external to the UAV is usually required. Commonly sonar will be used to determine the robot's altitude. Sonar, short for Sound Navigation and Ranging, uses sound waves to determine the altitude of the aerial robot by transmitting an acoustic signal toward the earth and then measuring the time it takes to receive a reflected "echo"[8]. Special algorithms are required to overcome the inherent limitation of the sonar that manifests itself when the aerial robot passes over a large object that may trigger the sonar to give an altitude reading that is smaller than the robot's actual altitude. Typically an aerial robot is equipped with a device that provides it with

some sense of sight such as a typical video camera, or radar. By detecting objects around the robot in successive time frames, the robot can accurately determine its position and speed relative to its surroundings.

The combination of the IMU and external sensing apparatuses give the robot a single measurement vector that it can rely on to know its position, velocity, and acceleration. However, combining these measurements into a single measurement vector isn't easy either. Typically the IMU measurements are recorded at higher frequencies than the rate at which a video camera or radar can record measurements. On top of this, every sensor has a margin of error inherent in its measuring capabilities to begin with. To overcome errors the use of stochastic processes and estimation theory are used typically through either a Kalman Filter or Extended Kalman Filter. Typically noise in the sensor can be thought of as distributed normally and therefore, has a variance inherent in it. The basic idea of the Kalman Filter in this situation is to continually reduce the variance over successive measurements until the margin of error goes to zero[9]. The estimate from the Kalman Filter is then used as a more accurate measurement for the robot to base mission level decisions.

Mission Level Decisions

Mission level decisions are largely application based, although work has been done to present some general formulation of the area. For instance, Team-Soars is an experiment geared toward modeling human team decision making[10]. Multi-UAV missions are modeled using Petri nets and Markov models in [11]. Modeling a mission through learning and goal-directed decision making is commonly used as seen in [12-14]. For successful completion of the IARC, the UAV will need to use collected sensor data to

determine the optimal iRobot to control in order to maximize the total number of iRobots herded to one end of the arena. For this thesis, it will be assumed that the UAV has made this decision and is ready to decide an optimal path to the iRobot that avoids any obstacles in the environment.

Navigation Level Decisions

Studies for navigating a UAV through an environment are in large part based on localization, mapping, and path planning. The ability for a UAV to localize itself with its environment using visual sensors can be found in [15] and [16]. These methods use cameras to gather information about the environment which the UAV is in. When dealing with an environment that geometrically has curvature, sometimes a video camera may not be the best solution. [17] presents a technique for localization in a tunnel using RF signal fading. Coupled with localization and mapping comes the actual navigation of the UAV through the mapped environment. Dijkstra's shortest path algorithm is used in [18] to facilitate navigation within a search and rescue scheme. Reinforcement learning coupled with a heuristic search algorithm is used in [19] to plan an optimal path in a construction environment. Limit cycles are extended beyond the limited two dimensional method into three dimensions in order to plan a path for a UAV in [20]. To navigate in a cluttered environment, an artificial potential field algorithm is used in most of literature for obstacle avoidance, and will be used for the obstacle avoidance component of the algorithm presented in this thesis. A literature review and discussion is in the introduction to Potential Fields-Proportional Navigation Guidance Law on page 24.

Control Structures

The control structure encompasses the algorithms required to actuate the UAV to a desired position or velocity determined by the Navigation Decision system. The complexities of the nonlinear quad rotor model and the methods to control it have been studied extensively. A comparison of the techniques can be found in [21]. Feedback linearization of the system for PID control is covered extensively in [22], [23], and [24]. Optimal Control techniques, including LQR and LPV, can be found in [25] and [26]. Adaptive control methods are applied in [27]. By far the most documented nonlinear control strategy used is a sliding mode approach found in [28-32]. In our simulation, a simple feedback linearization control scheme will be used. In this particular scheme, the UAV model is linearized about an operating point using a truncated Taylor series expansion. This linear model is used to estimate what the UAV performance will be to a particular set of inputs when it is close to the operating point. Then a basic PID controller is put in place to give actuation commands to the motors.

CHAPTER TWO

Quad Rotor Control Structure

The dynamics and control of the quad rotor model were modified from the presentation in [23]. Their approach is based on a Y configuration quadrotor, whereas our quadrotor is an X configuration. This simply means that the body frame reference has been rotated $pi/4$ radians about the body frame's z axis as depicted in Figure 2.1. This will change the rigid body dynamics of the quad rotor which will affect the mathematical models used for feedback linearization. Another deviation from [23] is in our control structure itself. They use an attitude control inner loop with a position control outer loop. We will be modifying this structure to a simpler velocity control outer loop to match the units of our reference input from the guidance algorithm developed in Potential Fields-Proportional Navigation Guidance Law on page 24.

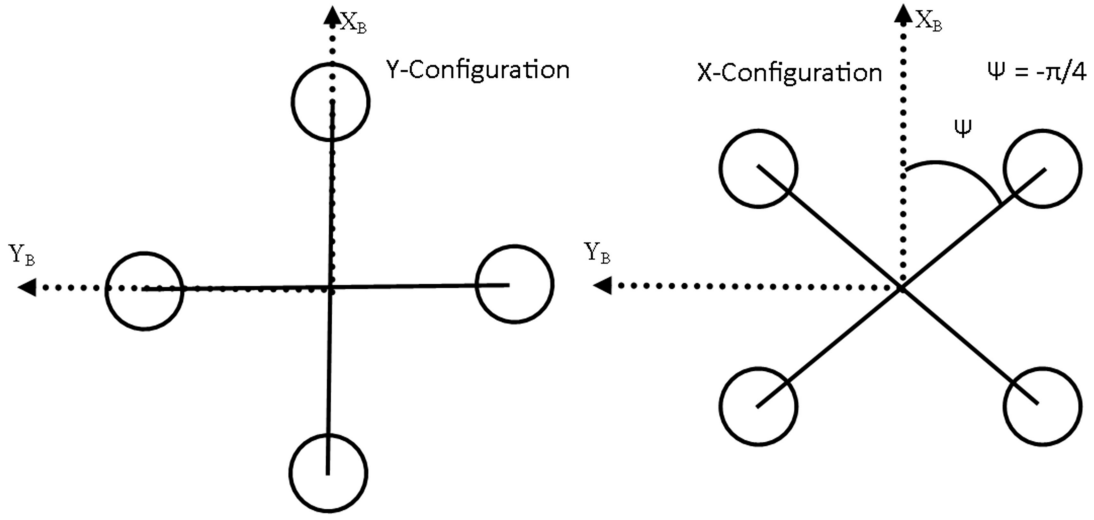


Figure 2.1 Rotating from Y Configuration to X Configuration

Two frames of reference are used in the model, x_b, y_b, z_b is the body frame of reference where x_b is pointed in the direction that the front of the UAV is pointed, y_b is pointed to the left of the UAV, and z_b is pointed normal to the top plane of the UAV. x_E, y_E, z_E is the earth frame of reference. We will use a state space model with a 12 dimensional state vector.

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{r} \\ \boldsymbol{v} \\ \boldsymbol{\Omega} \\ \boldsymbol{\Theta} \end{bmatrix}, \vec{\mathrm{r}} = \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ \mathrm{z} \end{bmatrix}, \dot{\vec{\boldsymbol{r}}} = \begin{bmatrix} \dot{\mathrm{x}} \\ \dot{y} \\ \dot{z} \end{bmatrix}, \boldsymbol{\Omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \boldsymbol{\Theta} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \tag{2.1}$$

The state vector is composed of four 3 dimensional column vectors as expressed in Equation (2.1). The position of the UAV expressed in the world frame is noted by $\vec{r}$, the velocity expressed in the world frame by $\dot{\vec{r}}$, the angular velocity expressed in the body frame by $\boldsymbol{\Omega}$, and the Euler angles by $\boldsymbol{\Theta}$.

Equations of Motion

We will develop the rigid body dynamics via Newtonian mechanics. As depicted in Figure 2.2, all forces created by the rotors are directed in the positive z_b direction, and all moments created by the rotors are in the x_b, y_b plane. The only other forces considered in this model is the force caused by gravity which will always be in the $-z_E$ direction. The equations of motion are already developed for a Y configured UAV in [23]. By utilizing a rotation matrix, we can adjust the equations to suit our X configuration.

We know a moment about an axis is simply the cross product of the force and the position vector of the force from the axis of rotation. Seen from the Y configured UAV in Figure 2.1, each force lay on an axis and is pointed normal to the x_b, y_b plane, which

reduces the cross product to a simple product of each force and its respective x_B or y_B component.

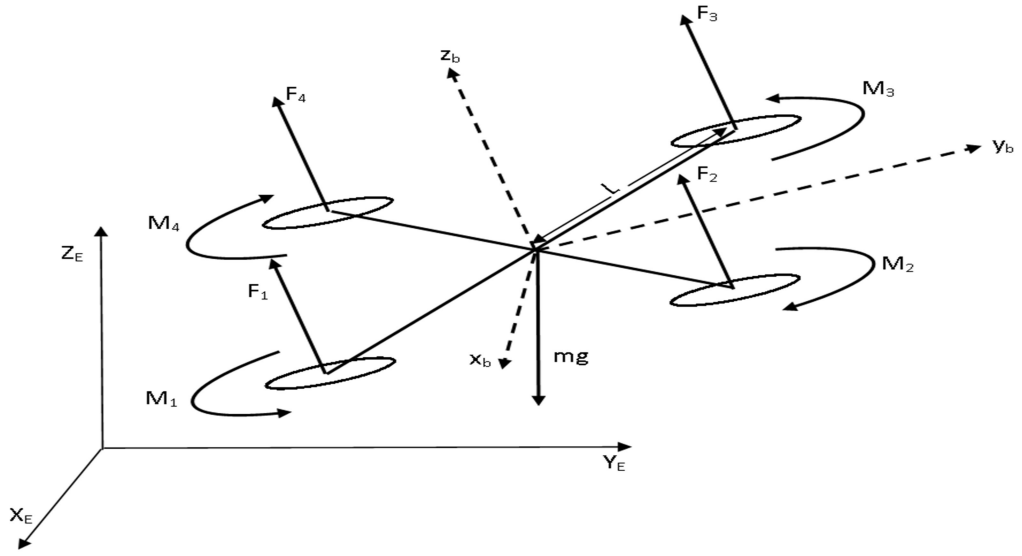


Figure 2.2 Forces and Moments of the Quadrotor

The moment about the z_b axis is the sum of moments of the motors, M_1–M_4 in Figure 2.2. The angular acceleration is then described by

$$J\dot{\boldsymbol{\Omega}} = \begin{bmatrix} L(\mathrm{F}_2 - \mathrm{F}_4) \\ L(\mathrm{F}_3 - \mathrm{F}_1) \\ (M_1 - M_2 + M_3 - M_4) \end{bmatrix} \tag{2.2}$$

[23]
To modify the equations to represent our X configured UAV we will use a rotation matrix about the z axis by $\frac{\pi}{4}$. The rotation matrix to fix the model is

$$R_M = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To see a brief overview of rotation matrices refer to the appendix on Rotations on page 94. Applying the rotation matrix leads to

$$J\dot{\Omega} = R_M \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ (M_1 - M_2 + M_3 - M_4) \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} L[(F_2 + F_3) - (F_1 + F_4)] \\ \frac{1}{\sqrt{2}} L[(F_3 + F_4) - (F_1 + F_2)] \\ \big((M_1 + M_3) - (M_2 + M_4)\big) \end{bmatrix} \quad (2.3)$$

The linear forces acting on the UAV due to the rotors are always in the z_b direction, therefore the rotation from the Y configuration to X configuration has no effect as to how the forces act on the UAV. The gravity is always with respect to the earth frame, therefore, the rotation has no effect on this force either.

$$m\ddot{\vec{r}} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + R_E \begin{bmatrix} 0 \\ 0 \\ \sum_i F_i \end{bmatrix} \quad (2.4)$$

[23]
R_E is a rotation matrix that transforms the accelerations in the body to accelerations in the earth frame. It is dependent on 3 angles, ϕ, θ, ψ, which represent angular rotations about the 3 axes in 3 successive frames of reference. These angles are called Euler angles. For more see the appendix Representing Frames with Euler Angles on page 96.

The relationship between the applied force and the angular rotor speed, ω, is found to be proportional to the square of ω.

$$F_i = K_F \omega_i^2 \quad (2.5)$$

[23]
Likewise the moment produced by each rotor is found to be proportional to the square of the angular speed.

$$M_i = K_M \omega_i^2 \quad (2.6)$$

[23]
The dynamics of the motor speed are modeled by a first order differential equation.

$$\dot{\omega}_i = k_{mo}\left(\omega_i^{ref} - \omega_i\right) \tag{2.7}$$

[23]

StateModel

Substituting equations(2.5) and (2.6) into (2.3) and (2.4), and solving for the accelerations give us the of the pieces of our state model.

$$\dot{\mathbf{\Omega}} = J^{-1}\begin{bmatrix} \frac{1}{\sqrt{2}}LK_F[(\omega_2^2+\omega_3^2)-(\omega_1^2+\omega_4^2)] \\ \frac{1}{\sqrt{2}}LK_F[(\omega_3^2+\omega_4^2)-(\omega_1^2+\omega_2^2)] \\ K_M\left((\omega_1^2+\omega_3^2)-(\omega_2^2+\omega_4^2)\right) \end{bmatrix} \tag{2.8}$$

$$\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \frac{1}{m}R_E\begin{bmatrix} 0 \\ 0 \\ K_F\sum_i \omega_i^2 \end{bmatrix} \tag{2.9}$$

To quantify the change in the Euler angles we use a relationship between the Euler angle rates and the body frame angular rates.

$$\dot{\mathbf{\Theta}} = \begin{bmatrix} 1 & \frac{s_\phi s_\theta}{c_\theta} & \frac{c_\phi s_\theta}{c_\theta} \\ 0 & c_\phi & -s_\phi \\ 0 & \frac{s_\phi}{c_\theta} & \frac{c_\phi}{c_\theta} \end{bmatrix}\mathbf{\Omega} \tag{2.10}$$

This relationship is developed in the appendix section Body Frame Angular Rates and Euler Angle Rates on page 97. Intuitively it is simple to see that if the UAV were rotated by 90 degrees about either x or y axis, the forces would be directed parallel to the face of the earth. The orientation would not provide any lift for the UAV, therefore, it would descend and crash. This is represented in equation (2.10) by the cosines in the denominator of some of the terms. If the rotation about the y axis is 90 degrees then the matrix will become singular. If a different order Euler angle transformations was chosen

to transform the body frame points to the earth frame, then the rotation about the x axis, ϕ, would be in the denominator causing singularity when the UAV is tilted 90 degrees about the x axis. Less intuitive may be the fact that the Euler rates are not dependent on the Euler angle position ψ.

Feedback Linearization Method

Now that an accurate state model has been developed a control technique called feedback linearization will be used to create a control structure that will allow the UAV to accurately track a reference velocity as an input. To do this first an inner feedback loop will be developed to handle the attitude control of the UAV. Then an outer feedback loop will be developed to control the velocity of the UAV. A high level development will be covered in this section. If detailed steps of the algebraic calculations are desired, refer to the appendix Feedback Linearization on page 99.

Inherently a quad rotor UAV is unstable. Without constant input, and constant feedback control it may drift at best, or at worst completely lose control. One equilibrium point in the model is when $\phi = \theta = 0, \dot{p} = \dot{q} = \dot{r} = 0$, and $\omega_i = \omega_h$. ω_h is the rotor speed with which each rotor must turn in order to keep the UAV hovering in place, $\ddot{z} = 0$. In equation (2.9) $R_E = I$, the identity matrix, when all angles are zero like our hovering point. Therefore, we can then conclude the following.

$$\omega_h = \sqrt{\frac{mg}{4K_F}} \tag{2.11}$$

We will linearize the rigid body dynamics about this point using a truncated Taylor series truncated at the first order terms. Then the linearized model will be coupled with a basic PID style of control to provide stability and controllability to the UAV. The control

structure contains an inner control loop that controls the attitude of the UAV and an outer control structure that controls the linear velocity of the UAV. For a diagram of the structure refer to Figure 2.3. For a detailed presentation of the truncated Taylor series expansion refer to the appendix Feedback Linearization on page 99.

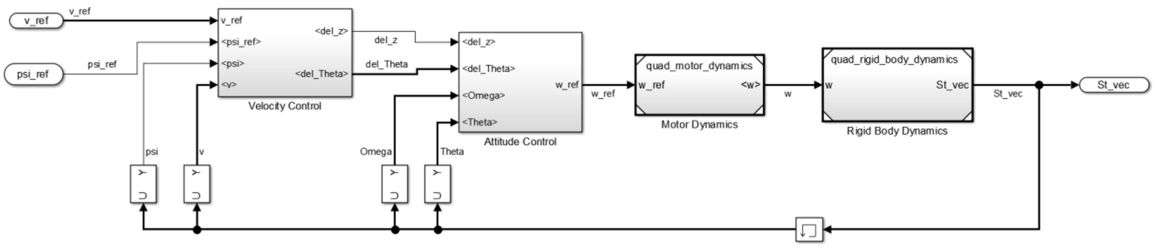


Figure 2.3 Control Structure for the UAV

PD Attitude Control

For the attitude controller we want a model that uses the desired orientation of the UAV to send rotor speed commands to the motor dynamics of the UAV model. Linearizing equation (2.8) about the chosen operating point leads to the following linear equations.

$$\dot{p}(\omega_1, \omega_2, \omega_3, \omega_4) \approx \frac{2\omega_h L K_F}{\sqrt{2}\, J_{xx}} \big((\omega_3 + \omega_2) - (\omega_1 + \omega_4)\big) \tag{2.12}$$

$$\dot{q}(\omega_1, \omega_2, \omega_3, \omega_4) \approx \frac{2\omega_h L K_F}{\sqrt{2} J_{yy}} \big((\omega_4 + \omega_3) - (\omega_1 + \omega_2)\big) \tag{2.13}$$

$$\dot{r}(\omega_1, \omega_2, \omega_3, \omega_4) \approx \frac{2\omega_h K_M}{J_{zz}} \big((\omega_1 + \omega_3) - (\omega_2 + \omega_4)\big) \tag{2.14}$$

These equations will be used to control the rate of change of our Euler rates. To determine a linear relationship between the Euler angles and the body angle rates we apply a truncated Taylor series expansion to equation (2.10).

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \approx \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.15}$$

To use equations (2.12)-(2.14) consider Figure 2.4. F_1-F_4 are forces from the rotors in the z direction (coming out of the page), and M_1-M_4 are the moments from each rotor that cause a total moment about the z axis.

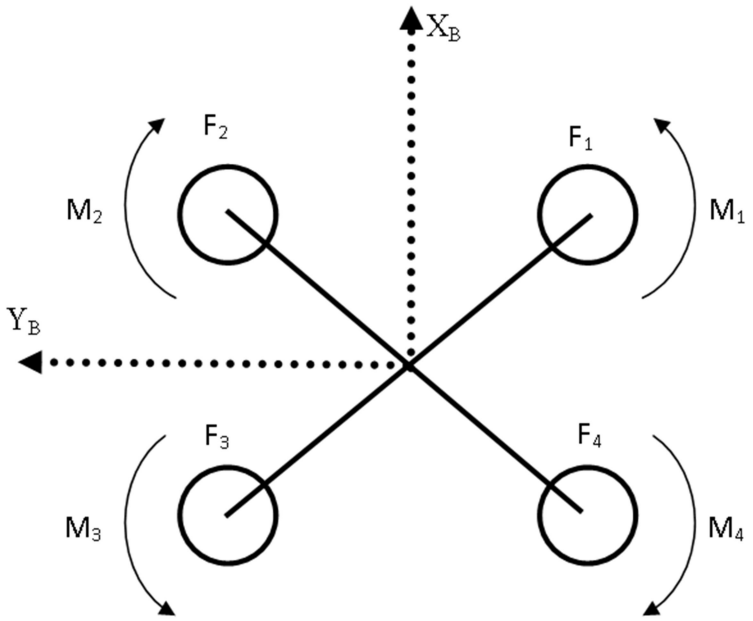


Figure 2.4 Top View of Forces and Moments

Now we consider that the sum of the motor speeds will cause a change in orientation of the UAV in the following manner.

$$\Delta\omega_{\phi} = -\omega_1 + \omega_2 + \omega_3 - \omega_4$$

$$\Delta\omega_{\theta} = -\omega_1 - \omega_2 + \omega_3 + \omega_4$$

$$\Delta\omega_{\psi} = \omega_1 - \omega_2 + \omega_3 - \omega_4$$

$\Delta\omega_{\phi}, \Delta\omega_{\theta}, \Delta\omega_{\psi}$ should be read as the total motor speed causing a perturbation of the respective subscripted angle. Likewise the sum of all four motor speeds should be equal

to 4 times the rotor speed required to make the rotor hover, ω_h, plus the motor speed causing a perturbation in the z position.

$$4\omega_h + \Delta\omega_z = \omega_1 + \omega_2 + \omega_3 + \omega_4$$

Putting these together we get the following.

$$\begin{bmatrix} 4\omega_h + \Delta\omega_z \\ \Delta\omega_\phi \\ \Delta\omega_\theta \\ \Delta\omega_\psi \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \tag{2.16}$$

By inversion we are now able to get a basis for our reference inputs to the motor dynamics.

$$\begin{bmatrix} \omega_1^r \\ \omega_2^r \\ \omega_3^r \\ \omega_4^r \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 4\omega_h + \Delta\omega_z \\ \Delta\omega_\phi \\ \Delta\omega_\theta \\ \Delta\omega_\psi \end{bmatrix} \tag{2.17}$$

By substituting equation (2.17) into equations (2.12)-(2.14) we get the following.

$$\dot{p}_{\text{ref}} \approx \frac{2\omega_h L K_F}{\sqrt{2} J_{xx}} \Delta\omega_\phi \tag{2.18}$$

$$\dot{q}_{ref} \approx \frac{2\omega_h L K_F}{\sqrt{2} J_{yy}} \Delta\omega_\theta \tag{2.19}$$

$$\dot{r}_{\text{ref}} \approx \frac{2\omega_h K_M}{J_{zz}} \Delta\omega_\psi \tag{2.20}$$

Considering equation (2.17) and equations (2.18)-(2.20) we are ready to implement a PD controller for the position of our Euler angles. We implement a PD controller to make the UAV track a reference orientation via Euler angles.

$$\begin{aligned} \Delta\omega_\phi &= k_{p,\phi}(\phi_{ref} - \phi) + k_{d,\phi}(p_{ref} - p) \\ \Delta\omega_\theta &= k_{p,\theta}(\theta_{ref} - \theta) + k_{d,\theta}(q_{ref} - q) \\ \Delta\omega_\psi &= k_{p,\psi}(\psi_{ref} - \psi) + k_{d,\psi}(r_{ref} - r) \end{aligned} \tag{2.21}$$

$p_{ref}, q_{ref}, r_{ref}$ are determined by integrating equations (2.18)-(2.20). By substituting our PD controller in into equation we have developed the full attitude control that takes reference angles, $\phi_{ref}, \theta_{ref}, \psi_{ref}$, and outputs desired motor speeds $\omega_1^r - \omega_4^r$. A block diagram of the control structure can be viewed in Figure 2.5. Notice with this controller alone the altitude of the UAV will decrease if the orientation has any magnitude about the x or y axes. This is handled with the velocity controller and simply passed through the attitude controller to the mixer that supplies the motor references.

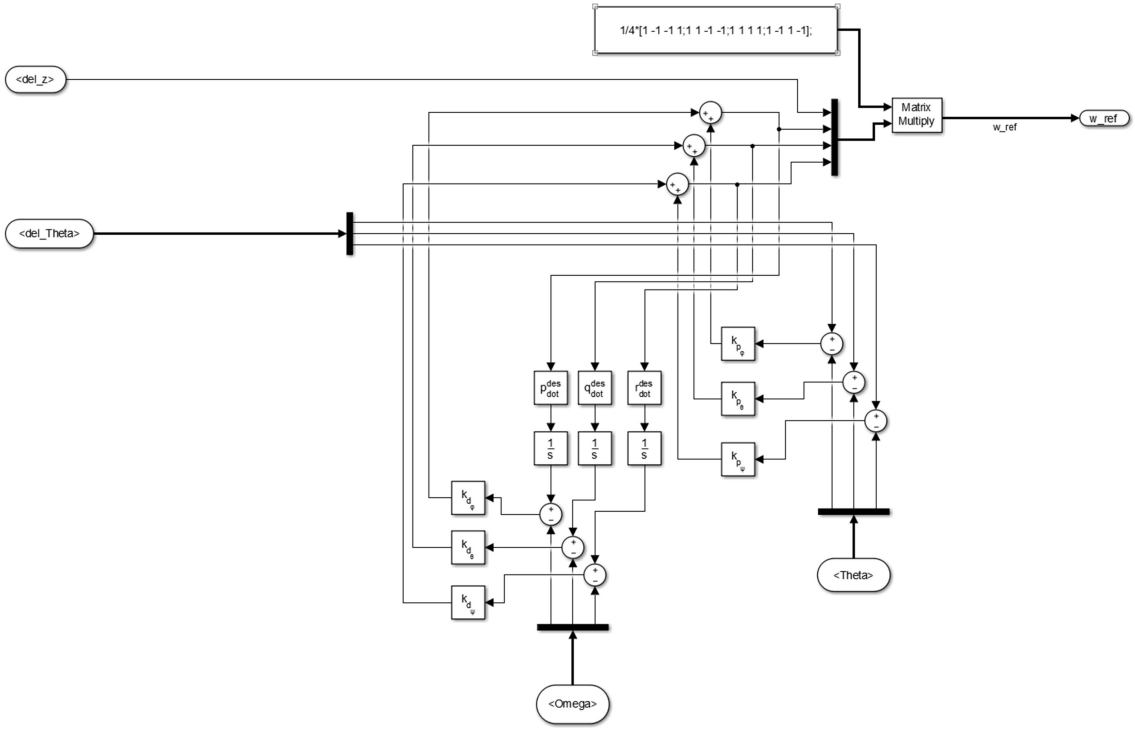


Figure 2.5 Attitude Controller Block Diagram

PID Velocity Control

For the outer loop we want a structure that takes a reference velocity vector and outputs desired Euler rates to the attitude control. The velocity controller should also output desired motor speeds to the motor input mixer for velocity in the z direction. By

referring back to Figure 2.4 it is obvious that the velocity components are dependent on the ψ component of the UAV orientation, and the model will be unstable if this angle is not fully taken into consideration. For this reason we will have a semi linear system involving two trigonometric equations with ψ as the argument. All other components will be linearized. From here on we will use the word linearize with this understanding implied.

We start by linearizing equation.(2.9). Multiplying through by the R_E term renders the following form.

$$\dot{\boldsymbol{v}} = \frac{K_F \sum_i \omega_i^2}{m} \begin{bmatrix} s_\psi s_\phi + c_\phi s_\theta c_\psi \\ s_\psi s_\theta c_\phi - c_\psi s_\phi \\ c_\phi c_\theta \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} \tag{2.22}$$

Going through the process of linearization covered in detail on page 101, the linearized model of equation (2.22) is as follows.

$$\ddot{x}_{des}(\omega, \phi, \theta, \psi) \approx g[\phi_{\text{des}} \sin(\psi) + \theta_{\text{des}} \cos(\psi)] \tag{2.23}$$

$$\ddot{y}_{des}(\omega, \phi, \theta, \psi) \approx g[\theta_{des} \sin(\psi) - \phi_{des} \cos(\psi)] \tag{2.24}$$

$$\ddot{z}_{des}(\omega, \phi, \theta, \psi) \approx 2\frac{K_F}{m}[\omega_1 + \omega_2 + \omega_3 + \omega_4 - 4\omega_h] \tag{2.25}$$

Referring back to equation (2.16) and substituting the relative elements in equation (2.25) we obtain a relationship for the motor mixer with respect to the z component of the UAV's position.

$$(4\omega_h + \Delta\omega_z)_{des} = \frac{m}{2 * K_F} \ddot{z}_{des} + 4\omega_h$$

Now we can invert equations (2.23) and (2.24) to form a relationship between our desired input into the attitude controller and their respective accelerations.

$$\phi_{des} = g(\sin(\psi)\, \ddot{x}_{des} - \cos(\psi)\, \ddot{y}_{des} \tag{2.26}$$

$$\theta_{des} = g(\cos(\psi)\ddot{x}_{des} + \sin(\psi)\,\ddot{y}_{des}) \tag{2.27}$$

To find the desired accelerations we use a simple proportional controller to convert reference velocity vector.

$$\dot{v}_{des} = k_{p,v}(v_{des} - v) \tag{2.28}$$

Now we have a complete control structure that accepts a reference velocity and a desired yaw angle and outputs desired motor speeds to have the UAV track appropriately. A block diagram of the velocity control structure can be seen in Figure 2.6. Notice the reference yaw angle is simply passed straight through to the attitude controller as would be expected.

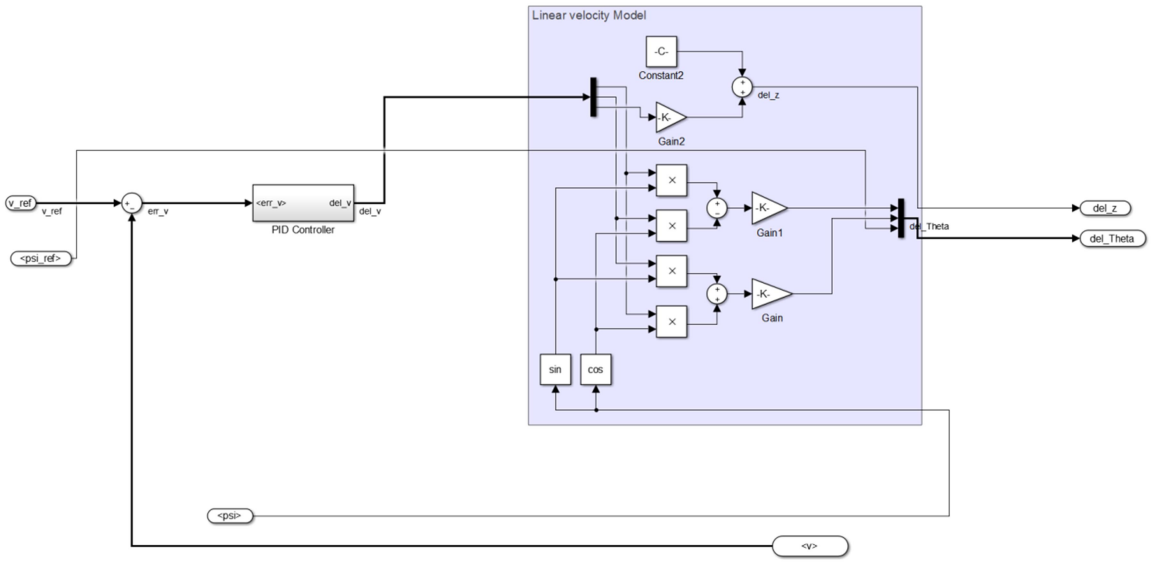


Figure 2.6 Velocity Control Block Diagram

CHAPTER THREE

Potential Fields-Proportional Navigation Guidance Law

The use of vectors and vector fields to control UAV's has been widely adopted due to the physical nature of a vector. The use of general vector fields to direct a UAV on both straight and circular paths were developed and proven to be stable via the Lyaponuv method in [36-38]. A method of generating a vector field is through the gradient of a potential field. This approach to developing the vector field is beneficial when obstacle avoidance, dynamic fields, or rendezvous with a target is desired. They have been applied to UAV's [39], nonholonomic wheeled robots [40, 41], and robotic arms [42]. To overcome limitations, such as local minima in the field, potential fields have been coupled with other algorithms such as fuzzy logic [43] and genetic algorithms. In our approach, we too will couple the potential field method with another method, proportional navigation, to overcome the inherent tail chase that ensues when using potential fields, and to add obstacle avoidance capabilities to proportional navigation.The potential field approach utilizes 2 sets of functions to produce a control vector. One set of functions are for developing the control vector component that guides the UAV to the intended target, while the 2^{nd} set of functions are for guiding the UAV away from any obstacles. We will show that the functions for guiding the UAV away from obstacles can be optimized by using vector correlation as a scaling function. Furthermore, we will show that the function set that guides the UAV to the intended target can be optimized by combining them with proportional navigation theory.

Proportional navigation is an interception guidance technique where control commands are meant to control of the rotational rate of the line of sight, LOS, vector [44]. The most widespread implementation is to control the robot's acceleration component that is normal to the line of sight between it and the target. In [45], the authors use a proportional navigation guidance technique to catch a ball with a robotic arm. They define their general interception task as "approaching a moving object while matching its location and velocity in the shortest possible time [45]". Some have even utilized the theory with two wheeled robots, [46, 47], although it isn't used prominently in robotics due to its trait to make contact with a target instead of rendezvous with the target. We will attempt to remedy this pitfall while adding the benefit of obstacle avoidance by coupling the method with potential field theory. Explicitly, a control vector will be constructed to control the UAV's velocity in two pieces. The direction will be determined using proportional navigation missile guidance theory, and the magnitude through a potential field obstacle avoidance theory. This will give the UAV the ability to rendezvous or land on a moving target, while effectively avoiding obstacles in the field. The idea of rendezvousing and landing on a target has been presented before. In [48] a helicopter follows the LOS, and bases the velocity magnitude on the distance from the target for effective landing. For moving targets, they simply add the target's velocity to the helicopter velocity magnitude. This method of simply following the LOS is known for creating a tail chase before arriving at a moving target, because the UAV does not take an angle to the target. Proportional navigation addresses this issue. Similar to our paper, in [49] and [50] the velocity of the robot is perturbed based on the motion of the target. However, rather than taking the targets velocity into account directly, the robot's

velocity is perturbed using a proportional control structure that relies on successive measurements of the LOS vector.

$$v_R = v_0 + \alpha * LOS$$

This would not be ideal if an evasive target was present. Our methodologies rely on the sensors of the UAV to obtain true velocity data of the target, and perturb the UAV velocity based on that information.

Certain assumptions were made to restrict the scope of this work:

1. The states of any obstacles in the field and of the target have been obtained through the sensors subsystem.
2. The environment variables have been analyzed at the Mission Decisions Level and an optimum target has been chosen.
3. The sampling time, T, may be chosen small enough that target and obstacle velocities may be considered constant and linear between samples without effecting performance.
4. The performance of the UAV is superior to that of the target in general.

In the next section potential fields will be described in detail as applied to our UAV. First it will be developed in two dimensions and then it will be shown how to extend the method to the third dimension. Following that, proportional navigation will be developed as applied to our UAV. Like the potential field method, it is easier to grasp the proportional navigation concept in two dimensions before showing how it is extended to three dimensions. A section on how to combine the two laws to give one velocity reference for input into the UAV control structure is covered. To end the chapter, an algorithm to fine tune the obstacle avoidance portion of the potential field method is presented.

Potential Field Guidance Law

Potential fields takes the concepts of potential energy fields and the gradient of that field, traditionally force in physics, and applies them to a control vector which allows a target to autonomously track a path to a designated target while avoiding any obstacles obstructing the path. The idea of using a potential field is a subset of control using an energy function. The goal is to develop a function that minimizes when a desired goal is reached. One way to think of it is in terms of electric potential. Consider the UAV having a charge, the intended target as having a charge with a sign opposite that of the UAV, and all obstacles as having a charge with the same sign as the UAV. In this case the UAV would be attracted by the target while at the same time, being repelled by any obstacles it comes close to. If the magnitude of attraction is proportional to the distance between the UAV and the target rather than inversely proportional as in electric potential, then the function created by this attraction, along with the repulsion from obstacles, would minimize at the target and maximize at a distance from the target or at the point of any obstacles. Oussama Khatib develops the appropriate equations in [42] which will be presented here. For appropriate notation throughout this section refer to Table 3.1.

It's important to distinguish a generalized potential energy function from well-known potential energy fields in physics. The restraints of units do not apply here; the energy function is not necessarily Joules and the gradient does not necessarily have to be Newtons. Rather than being restrained by physics, the potential energy field is purely artificial, denoted U_{art}, and is developed solely with idea of developing a velocity control vector for the UAV. The attractive component contributed by the target is itself an artificial energy function, denoted U_{att}, as well as the repulsive component contributed

by the obstacles collectively, denoted U_{rep}. Together they form the total artificial potential energy function.

$$U_{art} = U_{att} + U_{rep} \tag{3.1}$$

Table 3.1 Notation for Potential Field Algorithms

Symbol	Units	Description
$\vec{r}$	Meters	UAV position vector relative to the World Frame
$\dot{\vec{r}}$	Meters per second	UAV velocity vector relative to World Frame
$\dot{r}_{max}$	Meters per second	Maximum magnitude of UAV velocity vector
$\vec{t}$	Meters	Target position vector relative to the World Frame
$\dot{\vec{t}}$	Meters per second	Target velocity vector relative to the World Frame
$\vec{o}$	Meters	Obstacle position relative to the World Frame
$\dot{\vec{o}}$	Meters per second	Obstacle velocity vector relative to the World Frame
U_{art}		Artificial Potential Field
U_{att}		Attractive component of U_{art}
U_{rep}		Repulsive component of U_{art}
k_{rep}		Proportional gain for U_{rep}
k_{att}		Proportional gain for U_{att}
$\dot{\vec{r}}_{ref}$	Meters per second	UAV reference velocity vector

Defining the Attractive Field

First we develop the attractive component. A common method used to minimize an energy function is to take the negative gradient, ∇U_{att}, of the function and then step down it iteratively. This method is prone to stabilizing in a local minimum when the desired result is to completely minimize the function and stabilize at the global minimum. To try and minimize the local minima and saddles, it's ideal to have a function with a quadratic form, which also makes finding gradients easier in general. The idea is to develop an energy function that minimizes at the position of the target so that its use in the control of the UAV will result in directing the UAV to the target. Taking this into consideration U_{att} can be defined as:

$$\mathrm{U_{att}} = \frac{1}{2} k_{att} \left(\left| \vec{t} - \vec{r} \right| \right)^2 \tag{3.2}$$

where $\vec{t}$ is the position of the target, $\vec{r}$ is the position of the UAV, and k_{att} is a designer controlled gain. As expected, U_{att} reaches a minimum when the UAV reaches the target,$\vec{t} = \vec{r}$ and $U_{att} = 0$, and grows without bound as the distance from the target increases. From a gravitational potential field prospective, this artificial field can be thought of as tricking a UAV into thinking it is forced to climb a hill when moving away from the target, and moving downhill when approaching the target as shown in Figure 3.1. Here both the target and the UAV would be positioned in the x, y plane with the target located at $(-8, -8)$. Here the UAV can start anywhere in the plane and it will move toward the target just as a ball placed on the function itself would roll to the minimum. Furthermore, if gradient decent is used to minimize the function, the projection of the path of a free rolling ball onto the x, y plane would be the expected path of the UAV. Notice no matter where the UAV starts, it is guaranteed to reach the target through a gradient descent method without ending up stuck in a local minimum.

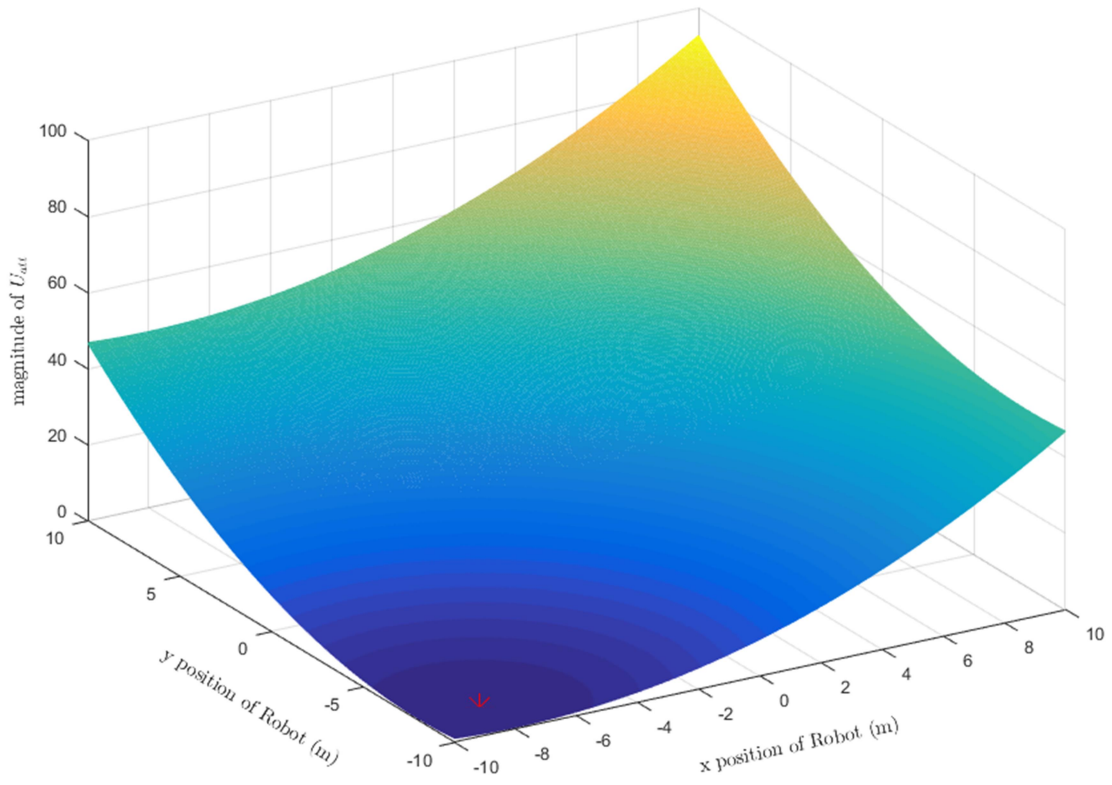


Figure 3.1 U_{att} Component of U_{art}

To allow the UAV to minimize Equation (3.2), the gradient of (3.2) will be used to create a vector field telling the UAV what direction to move.

$$\nabla U_{att} = k_{att}(\vec{t} - \vec{r}) \tag{3.3}$$

Now the UAV has more than just a scalar value that just tells it a general idea of how close it is. $\vec{t} - \vec{r}$ is a vector pointing directly at the target and scaled by design by k_{att}. The further away the UAV is, the larger the magnitude of the control vector and it goes to zero as the UAV approaches the target. This is the kind of behavior expected from the velocity of the UAV, so we'll define the gradient vector as the velocity control vector for the UAV.

$$\dot{\vec{r}}_{att} = \nabla U_{att} = k_{att}\,\vec{t} - \vec{r} \tag{3.4}$$

Developing the Repelling Field

For the repelling component of the artificial field, U_{rep}, the energy function is expected to minimize as the UAV moves away from the obstacle and grow without bound as the UAV approaches the obstacle. At some point the UAV will be far enough from the obstacle that the obstacle shouldn't contribute to the overall effect of the field. This distant should be set by the designer and will be denoted here as ρ. Staying with the idea of quadratic energy functions and gradient descent, U_{rep} is defined as

$$U_{rep} = \begin{cases} \frac{1}{2} k_{rep} \left(\frac{1}{|\vec{r} - \vec{o}|} - \frac{1}{\rho} \right)^2, & |\vec{r} - \vec{o}| \leq \rho \\ 0, & otherwise \end{cases} \tag{3.5}$$

[42]

where $\vec{o}$ represents the position of the obstacle. ρ denotes the radius of repulsion defined by the designer and makes $U_{rep} \to 0$ as$|\vec{r} - \vec{o}| \to \rho$, or as the UAV approaches the radius of no repulsion. Conversely, as $\vec{r} \to \vec{o}$, or as the UAV approaches the obstacle, U_{rep}, grows without bound. By definition after the UAV reaches a distance of ρ from the obstacle, $U_{rep} = 0$ to prevent any further manipulation from that particular obstacle. If all of the obstacles in the field have an artificial field attached to them, then the total repulsive factor of the field U_{art} is simply the sum of all the individual repulsive fields.

$$U_{rep} = \begin{cases} \sum_i \frac{1}{2} k_{rep} \left(\frac{1}{|\vec{r} - \vec{o}_i|} - \frac{1}{\rho} \right)^2, & |\vec{r} - \vec{t} \leq \rho \\ 0 & otherwise \end{cases} \tag{3.6}$$

From a gravitational potential field prospective, this produces a set of steep ‘hills’ that the UAV would have to climb in order to reach the obstacle as shown in Figure 3.2.

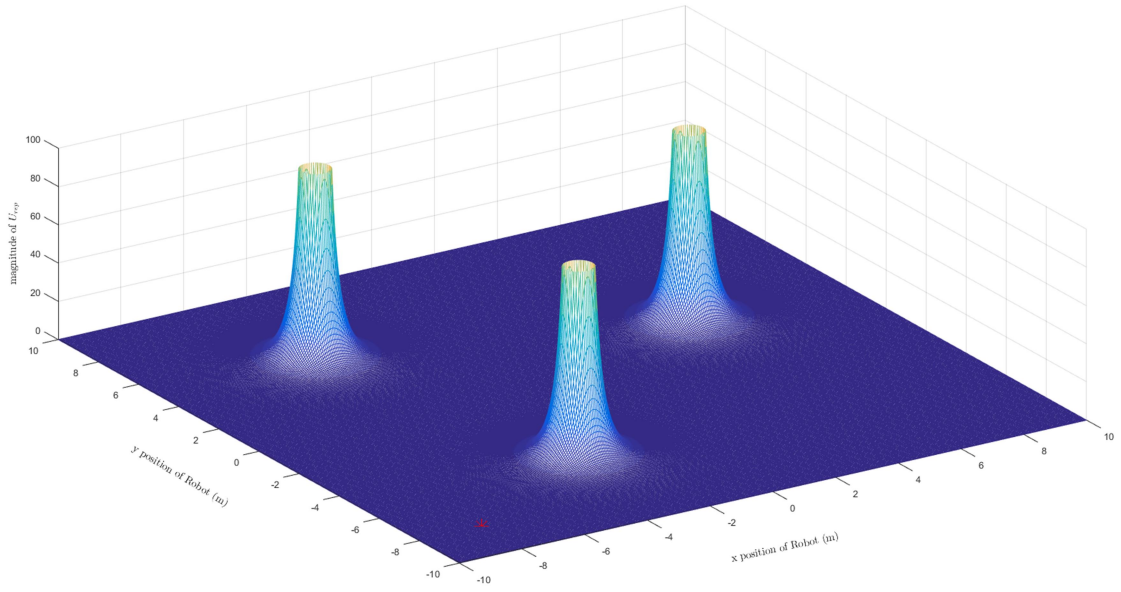


Figure 3.2 U_{rep} component of U_{art}

Now we want to provide a vector valued function for the UAV to have an idea of what direction to travel by taking the gradient of equation (3.6).

$$\nabla U_{rep} = \begin{cases} \sum_i k_{rep}\left(\frac{1}{|\vec{r}-\vec{o}_i|}-\frac{1}{\rho}\right)\left(\frac{1}{|\vec{r}-\vec{o}_i|^2}\right)\left(\frac{\vec{r}-\vec{o}_i}{|\vec{r}-\vec{o}_i|}\right), & |\vec{r}-\vec{o}_i| \le \rho \\ 0 & otherwise \end{cases} \quad (3.7)$$

$\vec{r}-\vec{o}_i$ in the numerator is a vector pointing away from obstacle in the direction of the UAV, and the $|\vec{r}-\vec{o}_i|$ terms in the denominator make the magnitude grow inversely proportional to the distance between the UAV and the obstacle. This again is the behavior expected from the velocity of the UAV, so it will be used as a velocity control vector.

$$\dot{\vec{r}}_{rep} = \nabla U_{rep} \quad (3.8)$$

Developing the Total Potential Field

By substituting equations (3.2) and (3.6) into equation (3.1), the total artificial potential field, U_{art}, effecting the UAV is determined. Figure 3.3 shows a plot of U_{art}

with a stationary target, $\vec{t} \equiv (-8, -8)^T$, and three stationary obstacles, $O = [\vec{o}_1, \vec{o}_2, \vec{o}_3]$. To get the UAV to the target while simultaneously avoiding obstacles we'll take the gradient of U_{art} to produce a vector field that points away from the objects and towards the target and can be used as a velocity control vector.

$$\nabla U_{art} = \nabla\left(U_{att} + U_{rep}\right) = \nabla U_{att} + \nabla U_{rep} \tag{3.9}$$

From Equations (3.3) and (3.7), equation (3.9) is solved, leading to a final velocity control vector, $\dot{\vec{r}}_r$, from equations (3.4) and (3.8).

$$\dot{\vec{r}}_{ref} = \dot{\vec{r}}_{att} + \dot{\vec{r}}_{rep} \tag{3.10}$$

To demonstrate to ability of equation (3.10) an environment with two obstacles positioned in the direct path between the UAV and the target was setup and the UAV was fed the velocity control vector at a rate of 10 hertz. The performance is captured in Figure 3.4..

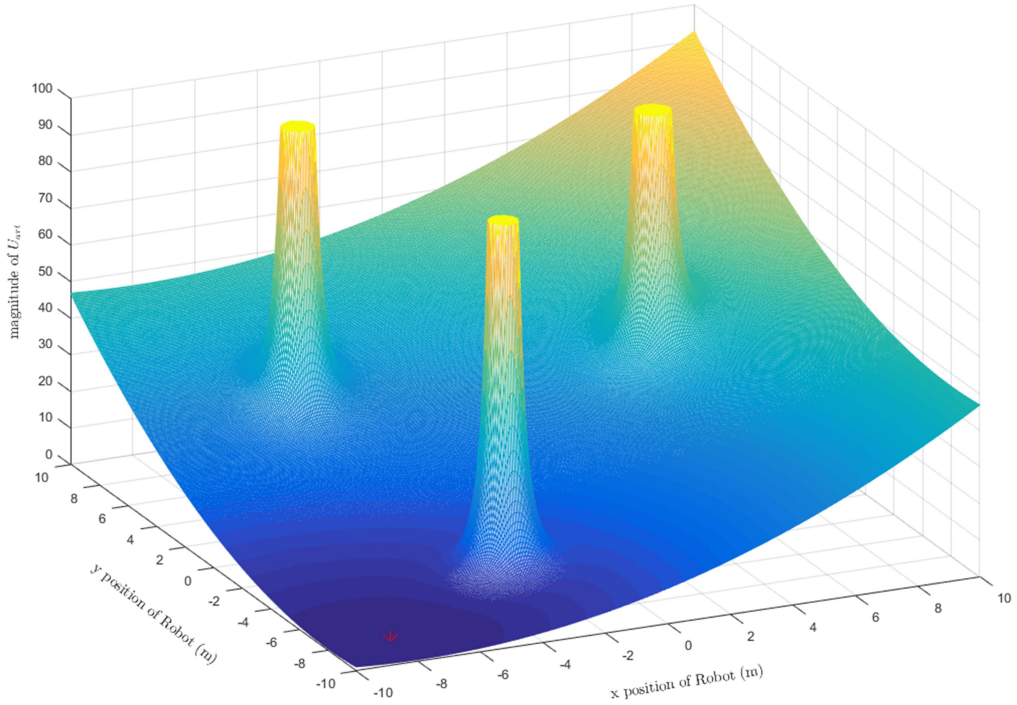


Figure 3.3 $U_{att} + U_{rep} = U_{art}$

The target in this scenario is held in position rather than moving. If the target were in motion then equation (3.3) shows that the UAV would not rendezvous with the target. Equation (3.3) goes to 0 when the UAV reaches the target, therefore, $\dot{\vec{r}}_{att} \rightarrow 0$ as the UAV approaches the target, so at the point of rendezvous, the UAV's velocity is 0, but the target's velocity is not. Inevitably the control vector to the UAV would simply reach the same magnitude and direction of the target's velocity at some point away from the target, and equilibrium would be reached where the UAV travels a fixed distance behind the target. In the following section we will look at popular guidance laws show how they are developed in a way that remedies this scenario. We'll also nuance the methods to see which provides the more optimal path toward a moving target.

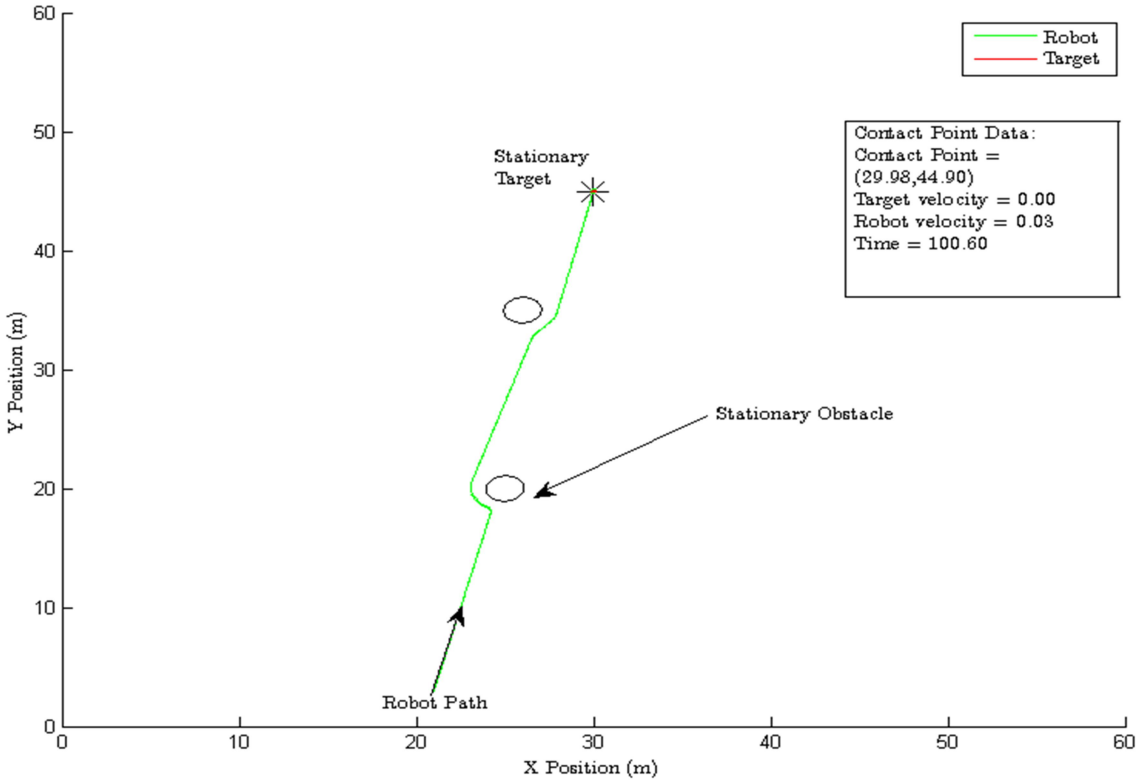


Figure 3.4 Plot of $\vec{r}$ trajectory using Potential Fields

Proportional Navigation Guidance Law

Typical guidance laws for impact with a target use the line of sight, LOS, between the target and the pursuer in one way or another. Guidance laws normally fall within two basic methods that have been researched extensively: pursuit and proportional navigation [51]. The angle of the LOS, ψ, is a common metric used to guide the pursuer to the target. In one of the most basic, though popularly used schemes, the heading of the pursuer is adjusted to bring the angle of the heading equal to ψ. This is called pursuit and amounts to constantly pointing the pursuer directly at the target without regard to the target's velocity. In many cases this results in the pursuer falling in behind the target and having to catch the target in a tail chase, or a race to catch up from behind. Proportional navigation was developed to enable missiles to better track an evasive target than the

capabilities of the more simplistic algorithm used by pursuit. The algorithm gets its name from the fact that the rotation of the velocity vector of the pursuer is proportional to the rotation of the LOS, or $\dot{\psi}$ [52]. To better explain proportional navigation, first pursuit will be covered and then it will be shown that not only is the proportional navigation algorithm more robust in terms of the target trajectories that it can make contact with, but the algorithm also typically out performs a simple pursuit algorithm on trajectories that the pursuit algorithm can handle. Including the notation for potential fields, refer to Table 3.2 for added notation in this section.

Table 3.2 Added Notation for Proportional Navigation

Symbol	Units	Description
$\dot{\overrightarrow{r_N}}$	Meters per second	Desired velocity of UAV normal to LOS
$\dot{\overrightarrow{r_T}}$	Meters per second	Desired velocity of UAV tangent to LOS
P		Projection matrix onto $LOS^{\perp}$
LOS	Meters	Line of Sight vector from UAV to Target
ψ	Radians	Angle of LOS
$\dot{\psi}$	Radians per second	Rate of change of LOS angle
T	Seconds	Sampling Time of UAV

Here both algorithms will assume a constant linear velocity for the target, $\dot{\vec{t}}$, over a time sample. It is assumed that the sampling time is frequent enough that the pursuer will be able to adjust to any nonlinearity and still make contact. For this assumption to be true, it is also assumed that the pursuer has better performance capabilities than the target, including maximum velocity, and maximum acceleration.

Since the UAV accepts a velocity control input vector, a pursuit algorithm will now be developed to produce that velocity control vector. First the LOS needs to be defined in terms of the position of the pursuer and the position of the target,

$$LOS = \vec{t} - \vec{r} \tag{3.11}$$

By normalizing the LOS vector, a pure direction is obtained and then the magnitude can be set so that the pursuer is traveling with its velocity vector pointing in the appropriate direction.

$$\dot{\vec{r}}_r = k_r \left(\frac{LOS}{|LOS|} \right) \tag{3.12}$$

k_r may be a constant or a function of time to set the magnitude of the velocity, but regardless of how the magnitude of k_r is set, or how the target evades the pursuer, the velocity vector is always pointed directly at the target [51]. Though this algorithm is simple, it is pervasive in many tracking algorithms. This is closely related to the attractive portion of the potential field method described previously. If k_{att} were a time varying function then substituting equation (3.11) back into equation (3.12) yields equation (3.3) with $k_{att} = \frac{k_r}{|LOS|}$. However this would defeat the purpose of trying to catch a moving target as discussed earlier. Equation (3.12) is utilized more appropriately by letting the $\left(\frac{LOS}{|LOS|} \right)$ portion simply be a directional unit vector, and using k_r to adjust the speed as desired. Figure 3.5 shows a trajectory plot of a pursuer pursuing a moving target using the pursuit algorithm, where the pursuer's velocity is held at a constant maximum magnitude.

Notice the pursuer does not take an angle when chasing the target. Instead, the pursuer continually points at the target, creating a tail chase scenario before reaching the target. When reviewing the attractive element of Potential Fields, it's easily inferred that the potential field algorithm has the same debilitating trait.

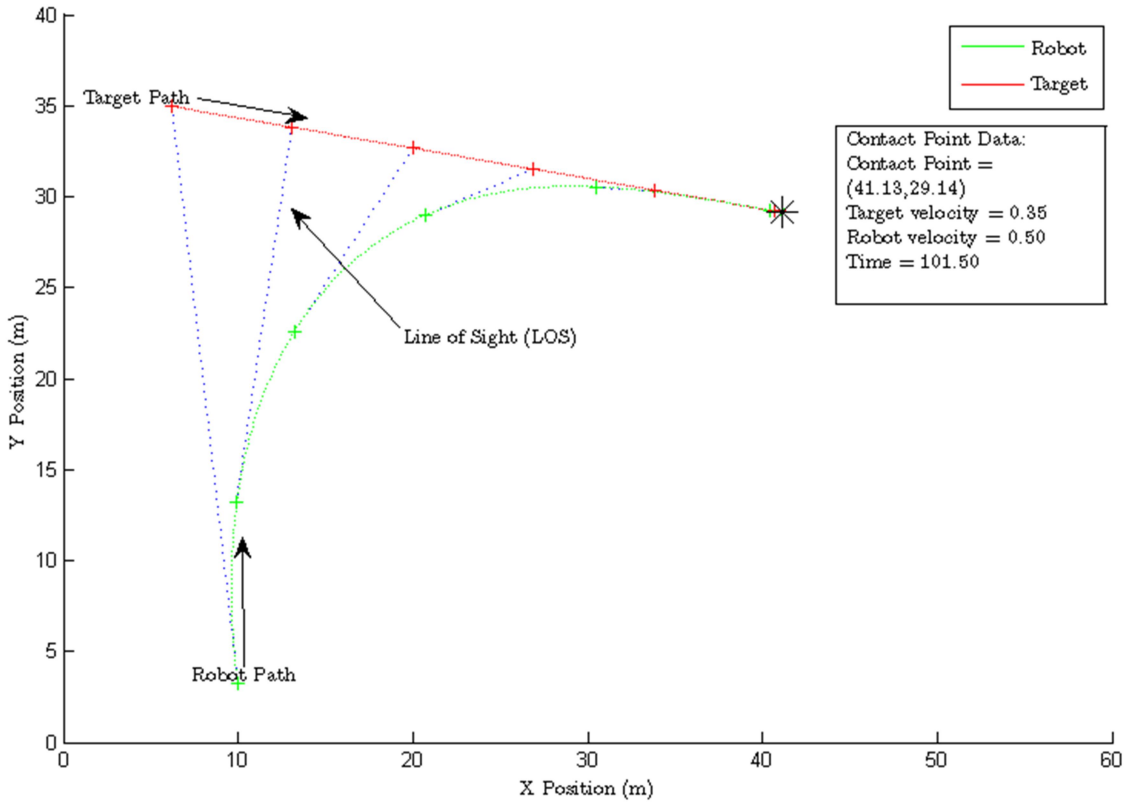


Figure 3.5 Path to Target using Pursuit guidance law

Proportional navigation tries to avoid this pitfall by setting a trajectory that indirectly takes the targets velocity into account. Rather than use the LOS as a heading reference, proportional navigation attempts to provide an optimal straight line path to the target by controlling the motion perpendicular to the LOS. In its basic form, proportional navigation is defined as:

$$\ddot{\vec{r}}_N = k\,\dot{\psi}\,\dot{\vec{r}}_T \tag{3.13}$$

[53]

$\ddot{\vec{r}}_N$ is the acceleration normal to the LOS, $\dot{\vec{r}}_T$ is the velocity tangent to the LOS, or the closing velocity, and $\dot{\psi}$ is the rate of change of the LOS angle. The main idea of the algorithm is that if a straight line path is taken such that there will be contact made with

the target, then there will be no rotation of the LOS. The acceleration normal to the LOS does indeed go to zero when the rotation of the LOS, $\dot{\psi}$, is zero. Furthermore, $\dot{r}_T$ in the algorithm accommodates the fact that the faster the pursuer is approaching the target, the faster it will have to adjust to ensure contact. To simplify Equation (3.13) the closing velocity is sometimes considered to be a constant and encapsulated with the control gain. This reduces the equation to

$$\ddot{\vec{r}}_N = k\,\dot{\psi} \tag{3.14}$$

Keeping with the idea of developing a velocity control vector from Section *Potential Field Guidance Law*, a velocity control vector will now be developed using proportional navigation.

Intuition implies that simply integrating equation (3.14) will get the desired normal component of the control vector.

$$\dot{\vec{r}}_N = \int k\,\dot{\psi}\,dt = k\int \dot{\psi}\;dt = k\left(\left(\psi_f - \psi_0\right)t + \dot{\vec{r}}_{N_0}\right) \tag{3.15}$$

This equation produces a velocity vector that is reactive to the change in ψ after the change has already occurred. If there isn't any change in ψ, equation (3.15) reduces to a constant, which shows that if ψ is constant, then the normal component of the velocity can be held constant and contact will be made. More importantly, if a scenario exists where there is an optimal straight line path to make contact with the target, there exists some LOS angle that can be used to set the normal component of the velocity control vector. If this angle can be determined then the velocity control vector can be set directly. To calculate the angle an estimated contact point would have to be calculated, which proves to be computationally intensive. The same idea of proportional navigation can be looked at as saying, if the normal component of the velocity vector is known such that

contact will be made with the target, then ψ will can be a constant proportional to that normal component of the velocity. From equation (3.14),

$$\psi = C\int \ddot{\vec{r}}_N \, dt = C\left(\left(\ddot{\vec{r}}_{N_f} - \ddot{\vec{r}}_{N_0}\right)t + \dot{\vec{r}}_{N_0}\right) \tag{3.16}$$

where $C = \frac{1}{k}$. If the velocity is constant, this reduces to

$$\psi = C\,\dot{\vec{r}}_{N_0} \tag{3.17}$$

The question now becomes how the velocity can be determined directly. Rather than trying to analytically develop a relationship for $\dot{\vec{r}}_N$ or ψ, geometrical methods will be a bit friendlier.

To determine the direction the UAV should take to insure contact with a moving target, first consider the simple example in Figure 3.6. At an initial time, t_0, the UAV is positioned at the origin traveling along the x axis. The anticipated target is traveling parallel to the x axis and crosses the y axis at exactly t_0. At this moment the UAV desires a velocity control vector that will point it in a direction that at some point $t > t_0$ will result in contact with the target. Obviously both the UAV and the target will share the same coordinates when contact is made, but what may not be immediately obvious is the fact that if both the target and UAV are traveling at constant velocities, then at every t, $t_0 < t \leq t_f$, the two must share the same x coordinate in order to make contact. Essentially the UAV can construct a velocity control vector by matching the target's velocity in the x direction and then applying whatever power is left in the UAV along the y direction. The velocities in the x and y direction are called the normal velocity and closing velocity, respectively. The closing velocity is called such because it is the part of the velocity vector that intends to bring the UAV closer to the target, and the normal

velocity is called so because it is normal to the line of sight, LOS, between the UAV and target. In this case the LOS is simply the y axis at t_0, and a set of lines parallel to the y axis at each successive t.

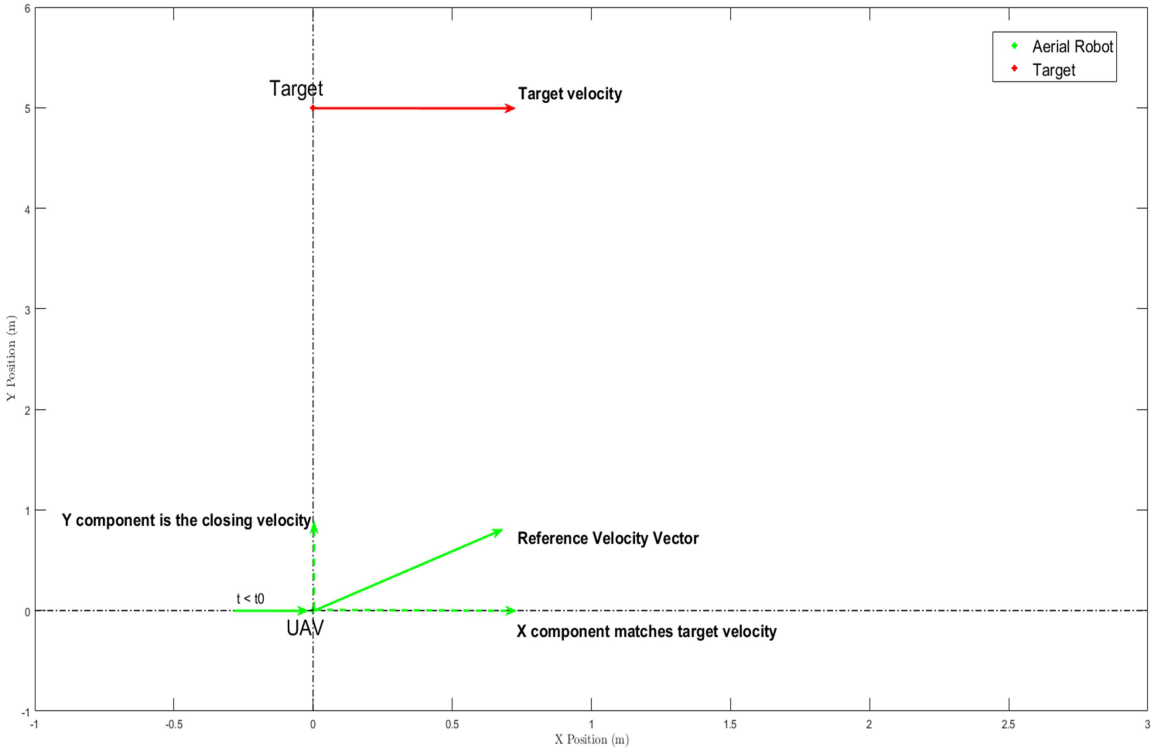


Figure 3.6 Proportional Navigation along the X axis

It may not be immediately obvious from Figure 3.6 that the target does not have to be traveling parallel to the x axis in order for the two agents to share x coordinates for all t. The significant attribute of this scenario that makes this so is the fact that at t_0 the line of sight, LOS, was the y axis. If the target was traveling on a path other than the path parallel to the x axis, then only the x component of its velocity vector would need to be considered and matched by the x component of the UAV velocity vector so long as the LOS was y at t_0.

To further generalize this concept, rather than thinking in terms of the x axis, simply using the proper term, the normal component of the velocity vector, brings to light the fact that *if the velocity components normal to the line of sight of both the UAV and the target are matched, the UAV has a higher velocity magnitude than the target, and the component of the UAV's velocity parallel to the line of sight is pointed at the target, then contact will be made at some time* $t > t_0$. This reduces the complexity of the problem to performing a change of basis on the target's velocity vector to get parallel and normal components to the LOS as shown in Figure 3.7. To do this let's consider two unit basis vectors, $\vec{b}_1$ pointing parallel to LOS, and $\vec{b}_2$ pointing normal to LOS; and define them as the basis β.

$$\beta = [\vec{b}_1, \vec{b}_2]$$

The target's velocity can then be expressed relative to β.

$$\dot{\vec{t}} = \mathrm{t}_1 \vec{b}_1 + \mathrm{t}_2 \vec{b}_2 = \left(\mathrm{Proj}_{\vec{b}_1} \dot{\vec{t}}\right) + \left(\mathrm{Proj}_{\vec{b}_2} \dot{\vec{t}}\right) \tag{3.18}$$

Calculating $\vec{b}_2$ is straight forward since the line of sight vector is easily computed by equation (3.11), $\vec{b}_2$ can be calculated by normalizing this equation.

$$\vec{b}_2 = \frac{LOS}{|LOS|} \tag{3.19}$$

If contact is to be made, then the two agents must share the same coordinates relative to $\bar{b}\vec{}_1$, therefore their velocities relative to $\bar{b}\vec{}_1$ must be matched.

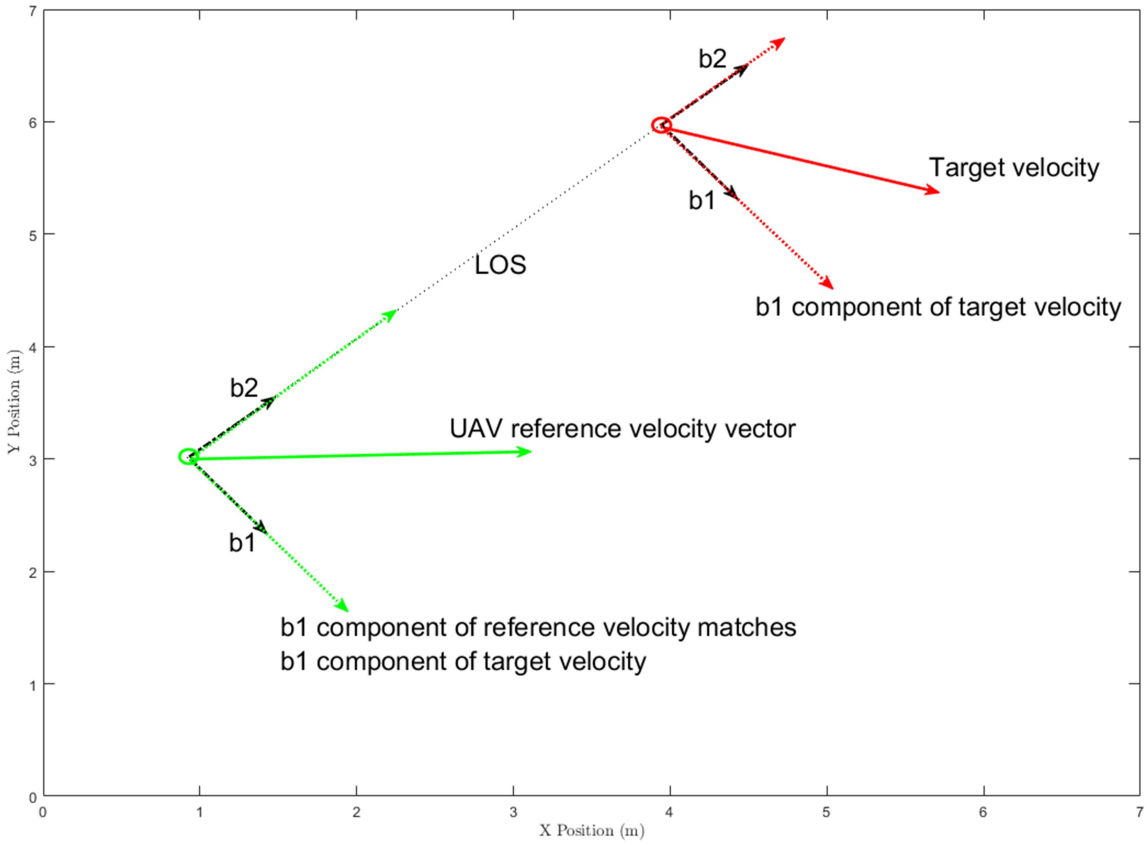


Figure 3.7 General Proportional Navigation

Rather than calculating $\bar{b}_1$ directly, the projection of the target's velocity, $\dot{t}$, onto $\bar{b}_1$, $Proj_{\bar{b}_1}\dot{t}$ can be calculated directly from the projection of $\dot{t}$ onto $\bar{b}_2$, $Proj_{\bar{b}_2}\dot{t}$

$$Proj_{\vec{b}_1}\dot{\vec{t}} = \dot{\vec{t}} - Proj_{\vec{b}_2}\dot{\vec{t}} \tag{3.20}$$

This is the desired quantity to facilitate constructing a velocity control vector. Therefore,

$$\dot{\vec{r}}_N = \dot{\vec{t}} - \mathrm{Proj}_{\vec{b}_2}\dot{\vec{t}} \tag{3.21}$$

where $\dot{\vec{r}}_N$ is the velocity control vector component normal to the line of sight. Since β is a set of unit vectors, $Proj_{\vec{b}_2}\dot{t}$ can be calculated by

$$\mathrm{Proj}_{\vec{b}_2}\dot{\vec{t}} = \left(\dot{\vec{t}}\cdot\vec{b}_2\right)\vec{b}_2 = \left(\dot{\vec{t}}^{\mathrm{T}}\vec{b}_2\right)\vec{b}_2 = \vec{b}_2\left(\vec{b}_2^T\dot{\vec{t}}\right) = \left(\vec{b}_2\vec{b}_2^T\right)\dot{\vec{t}} \tag{3.22}$$

which leads to

$$\dot{\vec{r}}_N = \dot{\vec{t}} - \left(\vec{b}_2\vec{b}_2^T\right)\dot{\vec{t}} = \left(I - \vec{b}_2\vec{b}_2^T\right)\dot{\vec{t}} \tag{3.23}$$

where I is the identity matrix. Referring back to Figure 3.7 and equation (3.11) shows all elements of equation (3.23) are known quantities. Furthermore, the form of equation (3.23) is such that the desired velocity normal to LOS can be calculated in just two calculations: constructing the projection matrix, and then using it to transform the target's velocity. Defining $\dot{\vec{r}}_N$ in terms of a projection matrix, equation (3.23) becomes

$$\dot{\vec{r}}_N = P * \dot{\vec{t}} \tag{3.24}$$

where $P = I - \vec{b}_2\vec{b}_2^T$.

A moment of care should be given to the idea that P actually is a projection matrix. Since $\dot{\vec{t}}$ can be any vector, the statement that P is a projection matrix suggests that *any* vector transformed by P will produce a corresponding vector projected onto b_2. One check that is readily available as to whether P is in fact a projection matrix is to check that it is idempotent, or $P^2 = P$. Although the discussion has only be in $\mathbb{R}^2$ thus far, here it will be proven that P is in fact a projection matrix in $\mathbb{R}^n$ to avoid repeating the exercise later. Proving it is fairly straight forward. First consider the general unit vector $\vec{x} = [x_1\ x_2\ \dots x_n]^T$. Therefore

$$\vec{x}\vec{x}^T = \begin{bmatrix} x_1x_1 & \cdots & x_1x_n \\ \vdots & \ddots & \vdots \\ x_nx_1 & \cdots & x_nx_n \end{bmatrix}$$

Notice each element of $\vec{x}\vec{x}^T$ is simply $(\vec{x}\vec{x}^T)_{(i,j)} = x_ix_j$. Each element of $(\vec{x}\vec{x}^T)^2$ can be represented as

$$(\vec{x}\vec{x}^T)^2_{(i,j)} = \sum_k x_ix_k^2x_j = x_ix_j\sum_k x_k^2$$

Recall that $\vec{x}$ is a unit vector, therefore $\sum_k x_k^2 = 1$, which leads to

$$(\vec{x}\vec{x}^T)^2_{(i,j)} = x_i x_j = \vec{x}\vec{x}^T_{(i,j)}$$

$\vec{x}\vec{x}^T$ is idempotent. This is expected because $\vec{x}\vec{x}^T$ represents the projection $\vec{b}\vec{b}^T$ in Equation (3.22). From here P can be shown to be idempotent directly.

$$P^2 = (I - \vec{x}\vec{x}^T)^2 = (I - \vec{x}\vec{x}^T)(I - \vec{x}\vec{x}^T) = I^2 - 2\vec{x}\vec{x}^T + (\vec{x}\vec{x}^T)^2 = I - 2\vec{x}\vec{x}^T + \vec{x}\vec{x}^T$$

$$= I - \vec{x}\vec{x}^T = P$$

So P is idempotent, therefore there is no immediate reason to believe that this isn't indeed the projection matrix sought.

Now that the normal component of the velocity control vector,$\dot{\vec{r}}_N$, has been found, a final velocity control vector, $\dot{\vec{r}}_{ref}$ may be calculated by finding the tangential component to the LOS,$\dot{\vec{r}}_T$.

$$\dot{\vec{r}}_{ref} = \dot{\vec{r}}_T + \dot{\vec{r}}_N \tag{3.25}$$

For a typical missile guidance application, full throttle would be held in order to reach the target in a minimum time. This allows for easy calculation of the final velocity control vector. Consider at full throttle the missile reaches a maximum velocity magnitude, $\dot{r}_{max}$. Then the magnitude of the tangential component can be found by

$$|\dot{\vec{r}}_T| = \sqrt{\dot{r}^2_{max} - |\dot{\vec{r}}^2_N|} = \sqrt{\dot{r}^2_{max} - \dot{\vec{r}}^T_N \dot{\vec{r}}_N}$$

giving a final velocity control vector as:

$$\dot{\vec{r}}_{ref} = \left(\sqrt{\dot{r}^2_{max} - \dot{\vec{r}}^T_N \dot{\vec{r}}_N}\right)\vec{b}_2 + \dot{\vec{r}}_N \tag{3.26}$$

A plot of the trajectory in pursuit of the target using the proportional navigation algorithm can be viewed in Figure 3.8.

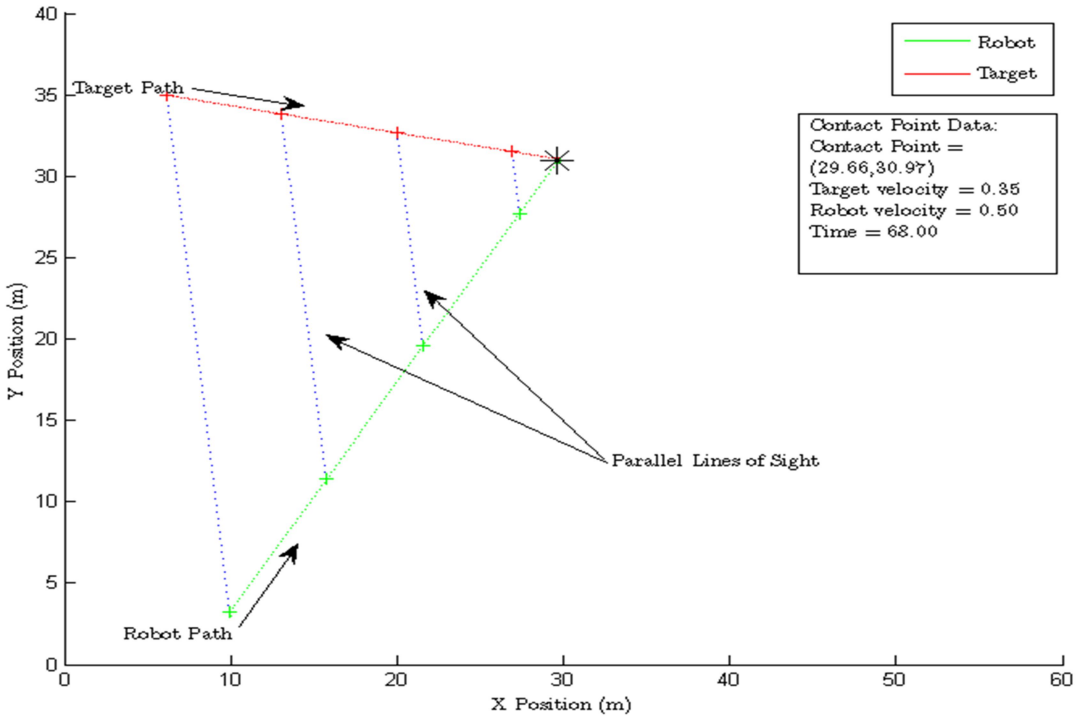


Figure 3.8 UAV Path using Proportional Navigation with Constant Speed

Notice that by calculating the velocity normal to the LOS, the rate of rotation of the LOS is zero as we set out to do. This means LOS lines at various time samples are parallel. A target going a constant velocity in a straight line may be an oversimplification of the problem so an evasive target is viewed next. Figure 3.9 shows a scenario where the target is told to try and move away from the UAV, rather than a straight line. Notice how even when the target evades the UAV, all the line of sight vectors are parallel, which means a zero rate of rotation as desired. A noticeable downfall to this approach for the purposes of the competition presents itself in the final contact speed in both Figure 3.8 and Figure 3.9. The UAV is traveling full speed. Since the strategy desired here is to rendezvous, the magnitude of the velocity on approach should change inversely to the magnitude of the LOS vector. Also, there is no mechanism built into the proportional

navigation scheme to handle scenarios where obstacles obstruct the flying path of the UAV. Both of these issues will be addressed in the next section.

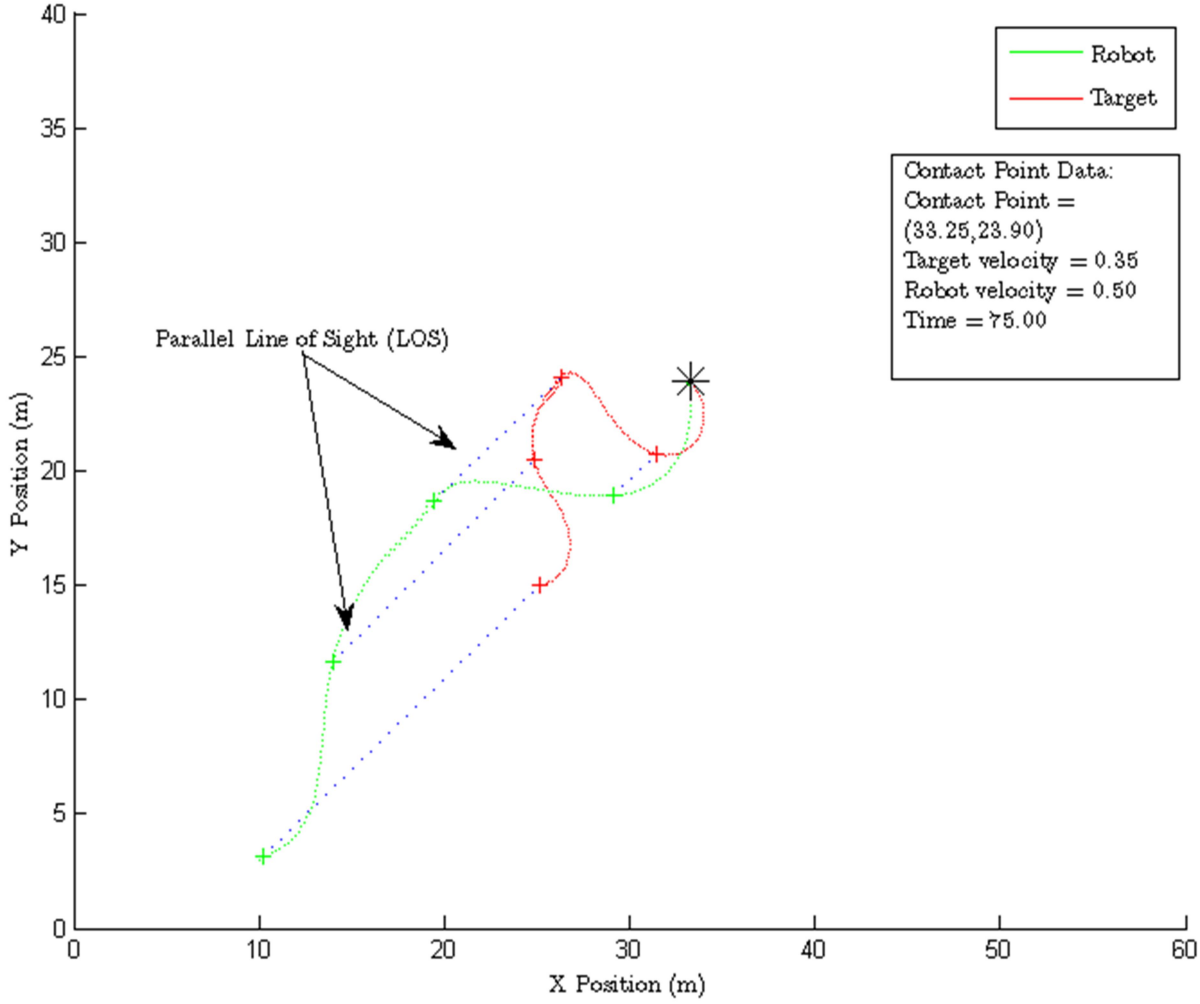


Figure 3.9 Proportional Navigation Response to an Evasive Target

Combining the Two Guidance Laws

In order to allow for the UAV to take a straight line solution to the target and rendezvous, rather than collide with target, $\dot{\vec{t}}, \vec{t}, and\ \vec{r}$ will be developed so that the magnitude of the UAV's velocity, $|\dot{\vec{r}}|$, can be adjusted using potential fields. Since the UAV also needs to avoid obstacles, it makes since that the potential field approach

discussed in section *Potential Field Guidance Law* should be used. However, the magnitude of the UAV's velocity needs to account for the velocity of a moving target in order to rendezvous, so U_{att} will be adjusted to include the speed of the target, $\dot{\vec{t}}$. To avoid simply pointing directly at the moving target and creating a tail chase proportional navigation will be used to set the direction of the UAV's reference velocity vector. To account for $\dot{\vec{t}}$,

$$U_{att} = \frac{1}{2} k_{att} \left(\left| \vec{t} - \vec{r} \right| \right)^2 + \frac{1}{2} k_{vel} \left| \dot{\vec{t}} \right|^2 \tag{3.27}$$

The new energy equation for the attractive portion of the potential changes the vector field associated with it.

$$\nabla U_{att} = k_{att} \left(\vec{t} - \vec{r} \right) + k_{vel} \dot{\vec{t}} \tag{3.28}$$

The repelling portion of the potential field, U_{rep}, will remain unchanged for the moment. The new gradient for the potential field, ∇U_{att}, will be used to set the magnitude of the attractive reference vector.

$$\left| \dot{\vec{r}}_{att} \right| = \left| \nabla U_{att} \right| = \left| k_{att} \left(\vec{t} - \vec{r} \right) + k_{vel} \dot{\vec{t}} \right| \tag{3.29}$$

Now that the magnitude of attractive component of the velocity control vector is determined, $|\dot{\vec{r}}_{att}|$ can be substituted for r_{max} in equation (3.26), giving a final control law for the attractive portion of the reference velocity vector:

$$\dot{\vec{r}}_{att} = \left(\sqrt{\left| \dot{\vec{r}}_{att} \right|^2 - \dot{\vec{r}}_N^T \dot{\vec{r}}_N} \right) \vec{b}_2 + \dot{\vec{r}}_N \tag{3.30}$$

A final total control law is obtained by substituting equation (3.30) back into (3.10).

$$\dot{\vec{r}}_{ref} = \dot{\vec{r}}_{att} + \dot{\vec{r}}_{rep}$$

Notice in Figure 3.10 that the UAV takes a straight line approach to the target, but when the UAV reaches the target its speed is relatively close to the target's speed.

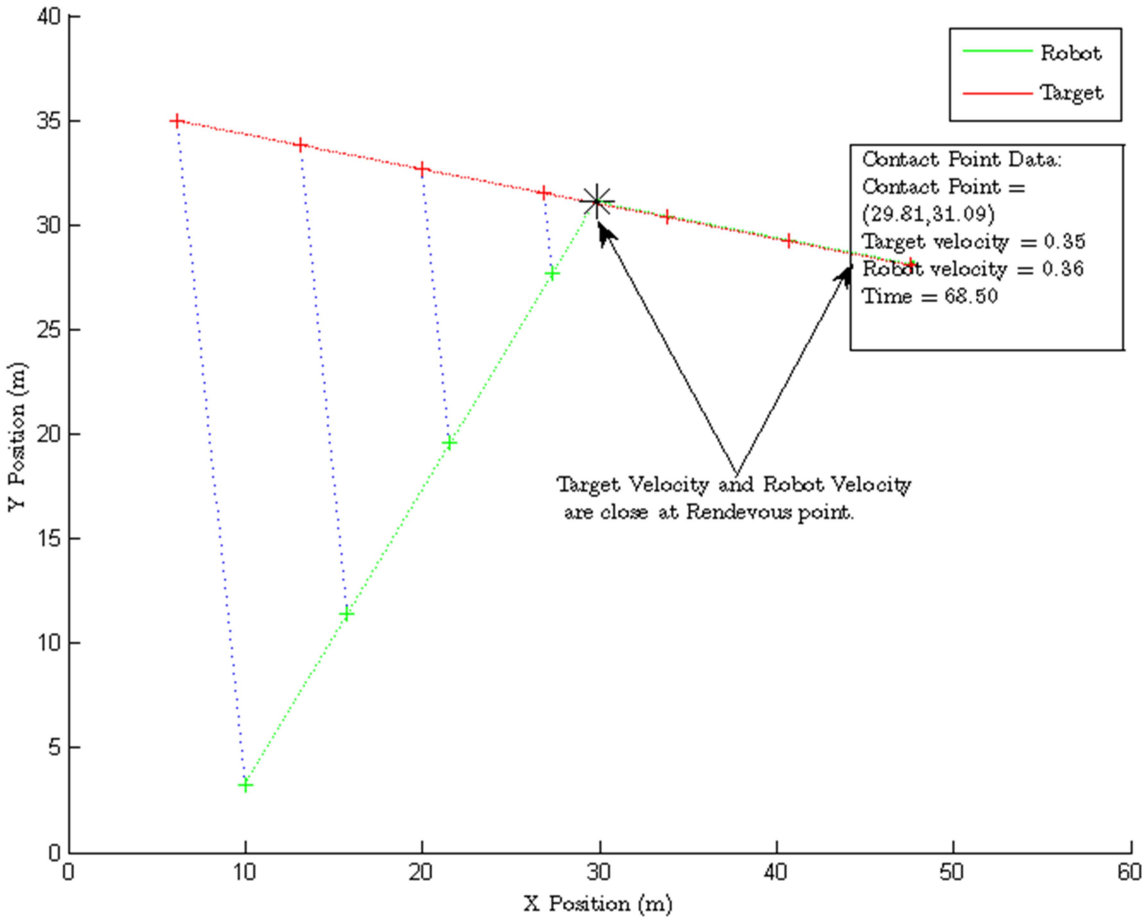


Figure 3.10 UAV Path using Proportional Navigation with Controlled Speed

Optimizing the Repelling Vector

Equation (3.5) creates a repulsive energy field that is circularly symmetric about the object, and depends only on position of the UAV and the obstacles. The symmetry causes the UAV to stay a radial distance from the object regardless of the direction of the UAV's velocity. Figure 3.11 depicts this.

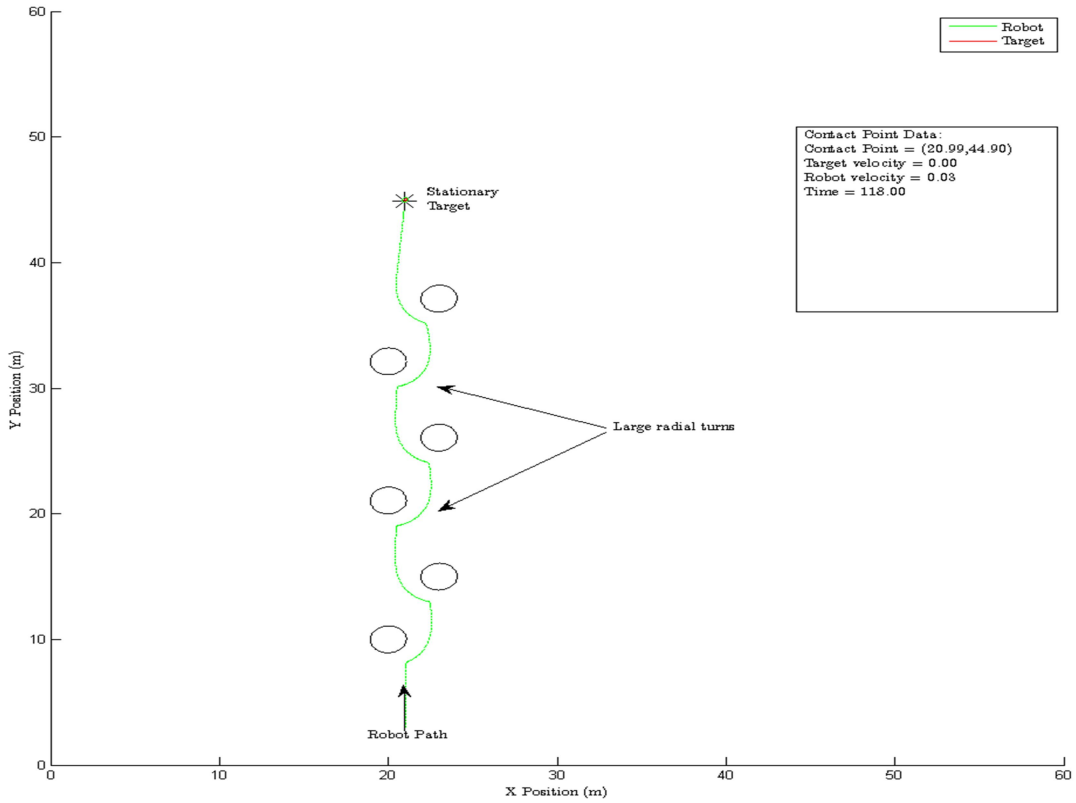


Figure 3.11 UAV Path using Equation (3.5)

By simply taking the direction of $\dot{\vec{r}}$ into account, the path can be smoothed significantly. To do this consider the angle between the UAV's velocity vector and the line of sight from the UAV to the obstacle in question, θ, as depicted in Figure 3.12.

The correlation between the vectors is $\cos(\theta)$, and in this case we will restrict the domain to $\theta = \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. This implies, $0 < \cos(\theta) < 1$, which is ideal for a scaling function. Following algebraic calculations for the correlation of two vectors yields:

$$f(\dot{\vec{r}}) = \begin{cases} corr(\dot{\vec{r}}, \vec{o} - \vec{r}), & corr(\dot{\vec{r}}, \vec{o} - \vec{r}) \geq 0 \\ 0, & otherwise \end{cases} \tag{3.31}$$

$$corr(\vec{u}, \vec{v}) = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}||\vec{v}|} = \cos(\theta_{u,v})$$

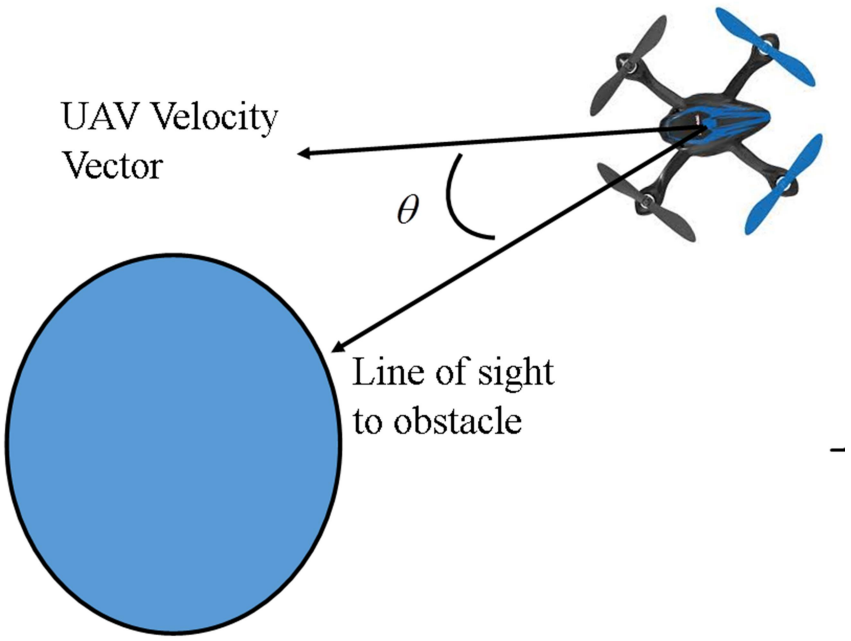


Figure 3.12 Correlation of UAV Velocity and LOS to Obstacle

If the UAV is traveling straight toward the obstacle then $f(\dot{\vec{r}}) = 1$. $f(\dot{\vec{r}})$ decreases as the UAV's velocity vector rotates out of the direction of the obstacle until the UAV is perpendicular to the LOS, at which point $f(\dot{\vec{r}}) = 0$. $f(\dot{\vec{r}})$ can be used to scale the magnitude of U_{rep} in Equation (3.5) according to magnitude of the UAV's velocity tangent to the LOS to the obstacle.

$$U_{rep} = \begin{cases} \frac{1}{2} k_{rep} f(\dot{\vec{r}}) \left(\frac{1}{|\vec{r} - \vec{o}|} - \frac{1}{\rho} \right)^2, & |\vec{r} - \vec{o}| \leq \rho \\ 0, & otherwise \end{cases} \tag{3.32}$$

As the UAV gets alongside the obstacle, $f(\dot{\vec{r}}) \to 0$, making $U_{rep} \to 0$ and allowing the UAV to continue pursuit of the target. However, if the UAV is directed toward the target then $f(\dot{\vec{r}}) \to 1$, and U_{rep} goes unchanged from equation (3.5). Figure 3.13shows a simulation with the same environment as Figure 3.11 except this time the correlation

scaling is applied. Notice the UAV takes a significantly smoother path, allowing it to cut its time to reach the target by 18.4 seconds!

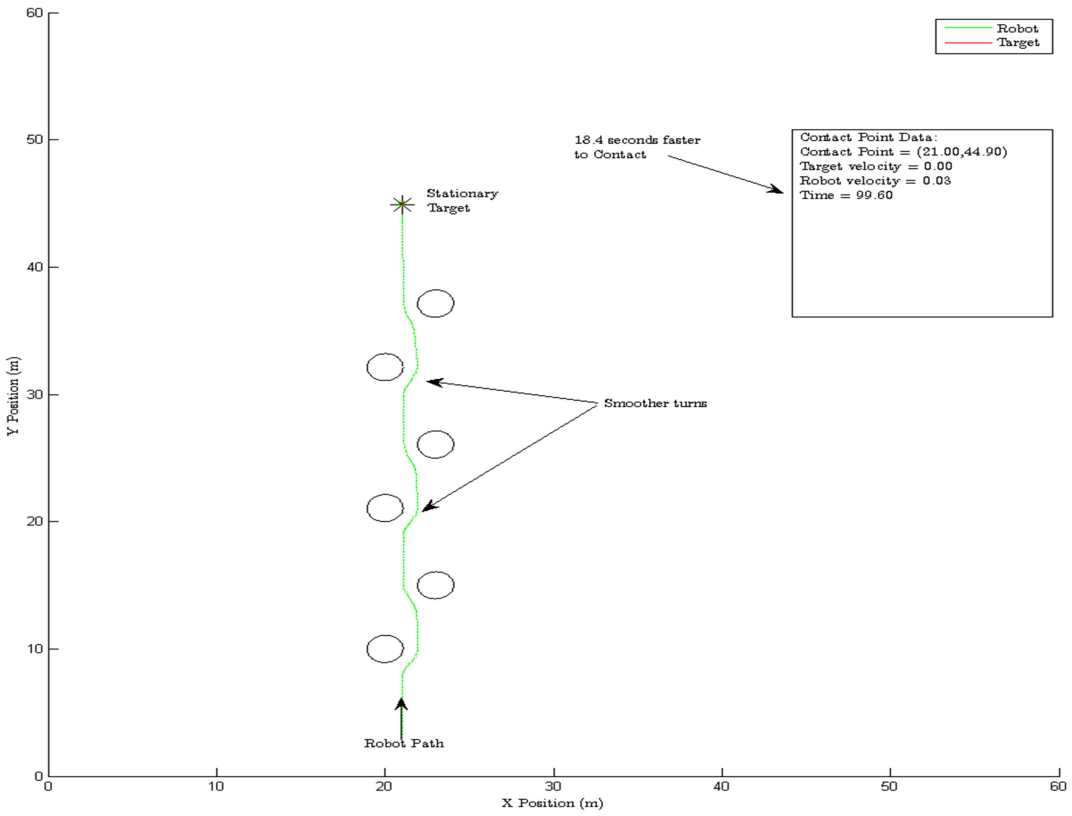


Figure 3.13 Path using Equation (3.32)

Extending the Algorithm to the Third Dimension

Until now the UAV and the target have been considered to be in the x, y plane. Now we will extend the method to the third dimension. To do this the competition rules will be taken into consideration. The height of the UAV cannot exceed three meters and the obstacles can be up to two meters tall. Taking this into consideration with the fact that the obstacle's radii are only the size of an iRobot, it is assumed that the UAV will lose minimal performance by going around the obstacles rather than over them. This allows us

to leave the repelling portion of the field, U_{rep}, as is. The repelling portion will act as if the obstacle in infinitely tall.

In general the UAV must fly which requires us to take gravity, and the altitude of the UAV into account. As far as gravity is concerned, it is assumed that if a velocity vector is passed to the UAV, the control structure is developed to handle outside forces to allow it to follow the velocity vector, including gravitational forces. Therefore, we are left only with the task of setting the altitude of the UAV. The way our current algorithm is developed, the proportional navigation portion sets the direction of the velocity vector. The idea that the UAV needs to gain altitude when away from the target suggests that the velocity vector needs to have a z component. This is the approach we will take. To develop the third dimension 3 new variables will be defined for control as shown in Table 3.3.

Table 3.3 Third Dimension Algorithm Constants

Symbol	Units	Description
t_ρ	Meters	Radius from target to ignore altitude commands
t_{alt}	Meters	Height above target to fly to from the far field
t_{art}	Meters	Artificial position of target

Rather than trying to command the UAV to try and reach a specific altitude, we'll realize that for the competition we simply want to be sure it is above the height of the iRobot, which is relatively low, and that it doesn't crash into the ground. To accomplish this we will have the UAV fly toward a set point, t_{art} which is t_{alt} above the target, when in the farfield, $\left|\vec{r} - \vec{t}\right| < t_\rho$. This in essence is simply raising the target into the air a set amount. So for equation (3.28), an artificial position is set above the target by t_{alt}. Of course if we want to actually touch down on the target, we will have to move our

reference point back to the top of the target. This is the purpose of defining t_ρ. So the new target location is set as

$$\vec{t}_{art} = \begin{cases} \vec{t} + \begin{bmatrix} 0 \\ 0 \\ t_{alt} \end{bmatrix}, & \left|\vec{r} - \vec{t}\right| \geq t_\rho \\ \vec{t} & \left|\vec{r} - \vec{t}\right| < t_\rho \end{cases} \tag{3.33}$$

The only question now is will proportional navigation indeed provide a straight path solution in three dimensions toward the artificial target, and then a straight line solution from the boundary of t_ρ to the target. In general, will proportional navigation work in three dimensions? By using vectors, we can see that the target's velocity vector along with the LOS creates a two dimensional plane that slices the three dimensional space. In this plane, proportional navigation will guide the UAV on a straight path to the target. In our case we have a plane up to and including t_ρ, and then back down to the target from t_ρ.

Summary of Developed Algorithm

To summarize the development of the algorithm, we modified a velocity control vector developed using potential fields.

$$\dot{\vec{r}}_{ref} = \dot{\vec{r}}_{att} + \dot{\vec{r}}_{rep}$$

Where $\dot{\vec{r}}_{att}$ is the component of the control vector that guides the UAV to the target, and $\dot{\vec{r}}_{rep}$ is the component of the control vector that guides the UAV away from any obstacles. The two components are defined with potential fields as below.

$$\dot{\vec{r}}_{att} = \nabla U_{att} = k_{att}(t - r)$$

$$\dot{\vec{r}}_{rep} = \nabla U_{rep} = \begin{cases} \sum_i \mathrm{k_{rep}} \left(\frac{1}{|\vec{r}-\vec{o}_i|} - \frac{1}{\rho}\right)\left(\frac{1}{|\vec{r}-\vec{o}_i|^2}\right)\left(\frac{\vec{r}-\vec{o}_i}{|\vec{r}-\vec{o}_i|}\right), & |\vec{r}-\vec{o}_i| \le \rho \\ 0 & otherwise \end{cases}$$

We used proportional navigation theory to modify the direction of $\dot{\vec{r}}_{att}$ as follows.

$$\dot{\vec{r}}_{att} = \left(\sqrt{\left|\dot{\vec{r}}_{att}\right|^2 - \dot{\vec{r}}_N^T \dot{\vec{r}}_N}\right)\vec{b}_2 + \dot{\vec{r}}_N$$

$$\dot{\vec{r}}_N = \left(I - \vec{b}_2 \vec{b}_2^T\right)\dot{\vec{t}}$$

Notice that the use of Pythagorean's theorem insures that the magnitude of $\dot{\vec{r}}_{att}$ remains unchanged. Only the direction is modified to give a more optimal path to the target.

We then used a correlation function to scale $\dot{\vec{r}}_{rep}$ according to the direction of the UAV's velocity relative to the position of an obstacle. In other words, we took the correlation between the UAV's current velocity vector and the line of sight vector from the UAV to the intended obstacle. Taking the correlation of two vectors produces $\cos(\theta)$, where θ is the angle between the two vectors. Furthermore

$$\dot{\vec{r}}_{rep} = \begin{cases} \sum_i \mathrm{k_{rep}} f(\dot{\vec{r}}) \left(\frac{1}{|\vec{r}-\vec{o}_i|} - \frac{1}{\rho}\right)\left(\frac{1}{|\vec{r}-\vec{o}_i|^2}\right)\left(\frac{\vec{r}-\vec{o}_i}{|\vec{r}-\vec{o}_i|}\right), & |\vec{r}-\vec{o}_i| \le \rho \\ 0 & otherwise \end{cases}$$

$$f(\overrightarrow{\dot{r}}) = \begin{cases} corr\left(\dot{\vec{r}}, \vec{o}-\vec{r}\right), & corr\left(\dot{\vec{r}}, \vec{o}-\vec{r}\right) \ge 0 \\ 0 & otherwise \end{cases}$$

$$corr(\vec{u}, \vec{v}) = \frac{\vec{u}\cdot\vec{v}}{|\vec{u}||\vec{v}|} = \cos\left(\theta_{u,v}\right)$$

CHAPTER FOUR

Simulating the Algorithm

A simulator to test our developed algorithm has been built. The model for the quadrotor and the guidance algorithm has been modeled as presented in this thesis. A simple linear model is developed to use as a target, and obstacles are placed as stationary cylinders in the field. Various scenarios have been run and measurements acquired to ensure the algorithm meets performance metrics. The block representations of the quad rotor have been shown in Figure 2.5 and Figure 2.6 therefore they will not be covered in this chapter. .A simple target model, a simple obstacle model, and a simple model to determine when contact has been made with the target will be covered. An extensive model representing the actual IARC has also been built for use after a mission planning algorithm has been developed. For the purpose of this thesis, it is assumed that all required sensory data to execute the path planning algorithm is obtainable. Therefore, the sensor block and mission level decision block in Figure 1.3 is bypassed, and the iRobot states, along with the obstacle states, are passed straight through to the Potential Fields algorithm. As discussed in Chapter 3, the guidance algorithm produces a reference velocity vector for the quad rotor to follow.

iRobot and Obstacle Model

The iRobot model used for proving the algorithm need not be an actual representation of the model. We are merely looking for a moving target whose

performance is less than that of the quadrotor. To make a model that resembles the real world, but is not overly complicated, we will use a second order equation.

$$p_{t+1} = p_t + v_t T + \frac{1}{2} a_t(T) T^2$$

$$v_t = a(T)T$$

p is the position of the iRobot, v is the velocity, $T$is the sampling time and $a(T)$ represents the acceleration dependent on each sample, but not on any time frame less than the sampling time. In other words the acceleration will be treated constant in each time frame, but will be allowed to change with each time frame to provide a space of paths to test the UAV on.

Obstacles will be placed in the field as constants. The number of obstacles placed in the field at each trial and their location is random so the UAV can run a Monte Carlo style simulation

Collision Detection

A block is built to check if the UAV has collided with obstacles. By checking if the norm of the line of sight vector is less than the sum of their radii, we can tell if they made a collision.

$$norm(\vec{r} - \vec{o}) < r.radius + o.radius$$

This model implies that all the obstacles are cylindrical; and for the IARC, they are.
To detect when the UAV actually rendezvous with the target, the same equation will be used.

$$norm(\vec{r} - \vec{t}) < r.radius + t.radius$$

At the point of rendezvous, the UAV and the target speed are compared to see if indeed a rendezvous has happed or if the UAV has merely crashed into it.

World Model

The world model is made to resemble the playing field specified by the IARC, including the arena, iRobots, and all obstacles. It has been developed for testing the full UAS with mission level decision capabilities to show that competition can be completed, and therefore will not be used to obtain results in this thesis. For the purpose of proving our algorithms are robust, various parameters are made tunable in the start.m file even though they would be set in the competition. Figure 4.1 shows the available variables. The real time parameters that these variables control are the runtime of the simulation, maxtime, the rate at which the states of the various robots are updated, maxtimeinc, the number of obstacles placed in the arena, maxobstacles, the number of iRobots placed in the arena, maxpieces, the number of aerial robots running the mission, maxplayers, and the respective maximum velocities of the iRobots and obstacles. The help the processor run multiple missions quickly, mgraphics controls whether each mission is plotted or not. Data from multiple missions can be collected in numerical form for use in performance measurements.

After saving the tunable parameters, the classes necessary for functionality are instantiated, and the various states of the world model are updated with time. Figure 4.2 is a flowchart that represents the discussion to follow. The top level class of the World Model is called RefereeClass. Analogous to an actual referee, the referee class stores the rules of the mission, points earned, and time. When the class is initiated, it pulls

parameters from the Mission Rules class such as the size of the arena, and how and when points are accrued.

```
60      %--------- tunable variables ----
61 -    maxtime = 60;
62 -    maxtimeinc = .1;
63 -    maxplayers = 0;
64 -    maxpieces = 10;
65 -    groundvel = .33;
66 -    maxobstacles = 4;
67 -    obstaclevel = .33;
68 -    obstacleRadialEffect = 1;
69 -    mgraphics = 1;
70      %--------------------------------
```

Figure 4.1 Tunable Variables for World Model

Then the referee class instantiates the number of obstacle, iRobot, and aerial robot agents specified by the user. It also instantiates the collision detection class, and if the mgraphics flag is set it instantiates the graphics data class. To make the model active, the Referee class provides an event notifier that triggers all other classes under it to perform functions every time the referee class time property is updated. Upon a time update, all agents in the world block have their states updated according to their specific trajectory algorithm.

The new positions of all agents are then sent to the Collision Detection Block, which detects if any agent has made contact with any other agent, including Aerial Robot contact with any iRobot or obstacle, and iRobot contact with any other iRobot or obstacle. The collision detection block adjusts the trajectories accordingly if a collision has been made. If the mgraphics flag is set in start.m then the graphics data class will structure and record the appropriate data for post mission plotting. At this point the world model has performed all necessary functions and waits for another time update.

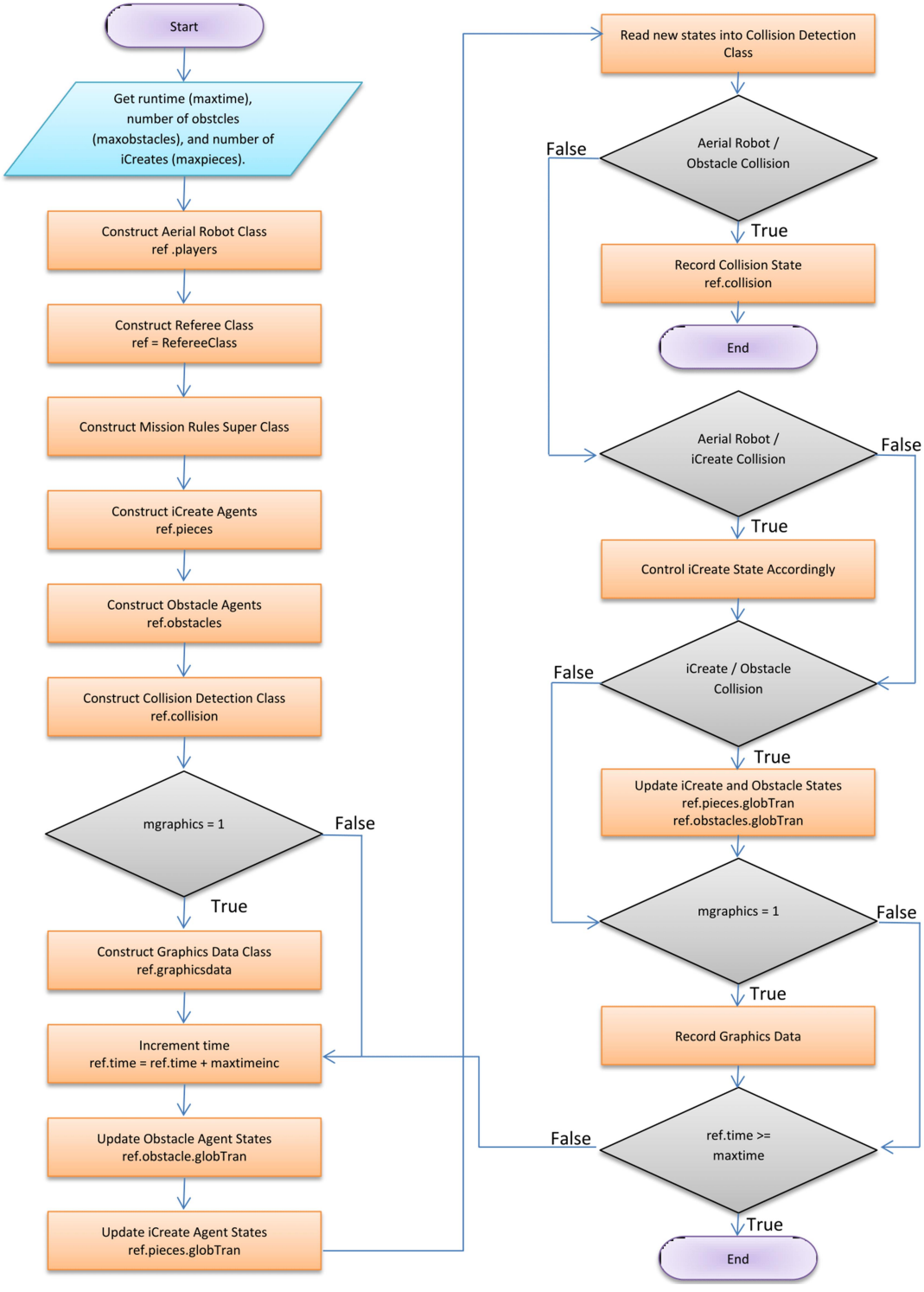


Figure 4.2 Program Flow for World Model Block

iRobot and Obstacle Trajectories

When the referee class's time parameter is updated, each robot's state is updated according to the time increment and the velocity during the previous time increment. At this time the velocities specified in the start.m file, Figure 4.1, are propagated through the entirety of the simulation, effectively giving the iRobot and obstacle agents a constant magnitude for their velocity vectors. Therefore there is only a need to calculate any rotation that the velocity may incur. The position for the obstacles is fairly simple to obtain, because per the mission rules, they simply drive in a circle.

$$\vec{o} = \begin{bmatrix} k_{rad}\cos(vt) \\ k_{rad}\sin(vt) \end{bmatrix}$$

k_{rad} represents the desired radius of the circle the obstacles will traverse, and v represents the velocity of the obstacle.

The trajectories of the iRobots are dependent on the position of the aerial robot, the time, and whether or not it is colliding with one of the obstacles. Figure 4.3 shows the logic used to determine the direction of travel for each time stamp. Per the mission rules, every five seconds the iRobot trajectory will change randomly up to 20° clockwise, and every 20 seconds it will reverse its trajectory. If the collision class has determined that the aerial robot has made contact with the top of the iRobot, then its trajectory will be altered by 45° clockwise. The velocity vector is rotated using a standard rotation matrix

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{4.1}$$

$$\vec{v}_{t+1} = R * \vec{v}_t$$

Θ is the amount the iRobot is rotated clockwise as discussed above and shown in Figure 4.3, and $\vec{v}$ is the velocity of the iRobot. Over a single time increment the iRobot is assumed to have a constant velocity which makes the position easily calculated by

$$\vec{p}_{t+1} = \vec{p}_t + \vec{v}_t t$$

$\vec{p}$ is the position of the iRobot and t is the time increment since the last update.

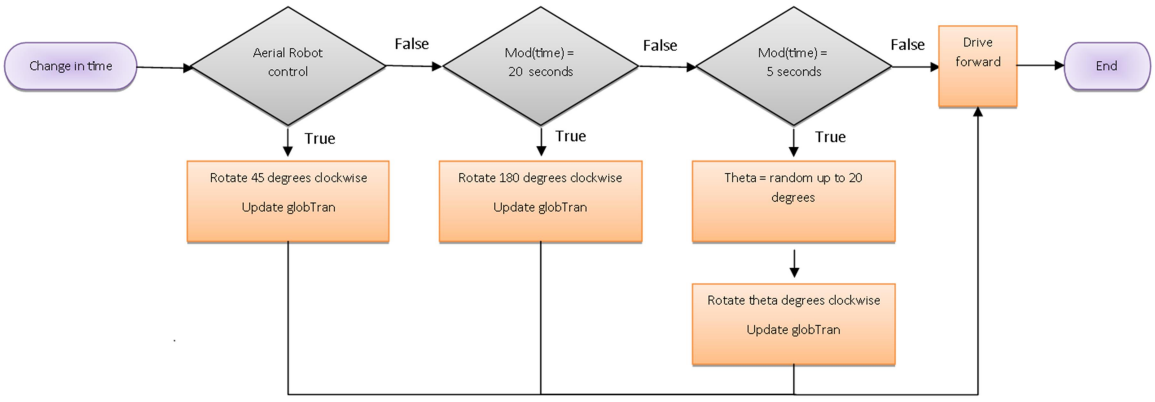


Figure 4.3 Program Flow for iRobot trajectories

Collision Detection Block

When the states are passed to the Collision Detection block, for loops are used to first check for aerial robot contact with any obstacles or iRobots, and then each iRobot is checked for collision with any obstacles or other iRobots. Refer to Figure 4.6 for the logical flowchart. Since each agent is cylindrical, the task of determining if a collision has been made is greatly reduced. All information needed to calculate collision with the aerial robot or obstacles are position vectors, radii, and heights, along with the aerial robots altitude. Let $\vec{r}, \vec{t},\ and\ \vec{o}$ represent the aerial robot, iRobot, and obstacle's position vectors, respectively. Let $r.radius, t.radius, and\ o.radius$ represent their radii, and $r.height, t.height, and\ o.heigh$ represent their respective heights. The aerial robot's altitude is represented by $r.altitude$. If the aerial robot has made a collision with an

obstacle the distance in the x, y plane between the center of the aerial robot and center of the obstacle will be less than the sum of their radii, and the altitude of the aerial robot will be less than the height of the obstacle.

$$norm(\vec{r} - \vec{o}) < r.radius + o.radius \;\&\; r.altitude < o.height$$

If a collision with an obstacle is detected, the collision detection class notifies the referee class, which stores the state of the mission at time of collision and then ends the mission immediately. A basic collision of the aerial robot or obstacle with an iRobot can be calculated the same way, but the addition of the front bumper and control magnet make it necessary to determine where the contact is made with the robot to properly adjust the velocity vector. Figure 4.4 shows the logical flow of how the aerial robot can control an iRobot.

To determine if the aerial robot has made contact with the top of the iRobot the following equation is used,

$$norm(\vec{r} - \vec{t}) < r.radius + t.radius \;\&\; r.altitude = t.height$$

When this is true the iRobot's velocity vector is rotated 45° clockwise using Equation (4.1). However if the aerial robot has descended lower than the height then a collision has been made and it must be determined if the front bumper has made contact or some other area of the iRobot. Since the velocity vector, $\dot{\vec{t}}$, of the iRobot always points in the forward direction, it can be used to represent the orientation of the agent. $\vec{r} - \vec{t}$ produces a vector pointing directly from the iRobot to the aerial robot. By taking the correlation of these two vectors, the cosine of the angle between them, $\cos(\psi)$, can be found. Since the bumper traverses the circumference of the iRobot, its entirety can be represented by an interval of angles with endpoints at the edge of the bumper, ϕ. By comparing $\cos(\psi)$ and

$cos\phi$, it can be determined if the aerial robot is in contact with the front bumper or some other portion of the iRobot.

$$|\cos(\psi)| > \cos(\phi) \tag{4.2}$$

$$\cos(\psi) = \frac{(\vec{r} - \vec{t}) \cdot \dot{\vec{t}}}{|\vec{r} - \vec{t}| \left|\dot{\vec{t}}\right|}$$

The bumper covers the front half of the circumference of the iRobot, or 90° in both directions from the front, $\phi = \frac{\pi}{2}$. Refer to Figure 4.5. Since the cosine function decreases from 1 to 0 as the angle moves from 0° to ±90°, a larger correlation value indicates the aerial robot being positioned toward the front of the iRobot, hence the greater than sign in equation (4.2). When it is determined that the aerial robot has indeed made contact with the front bumper, the iRobot's velocity vector is rotated 180.

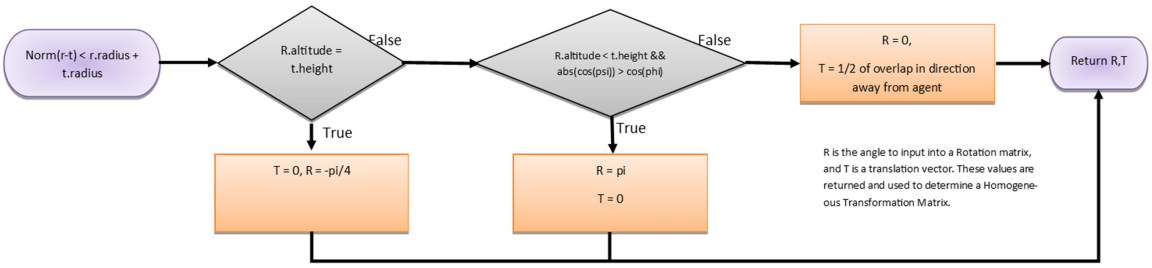


Figure 4.4 Program Flow for Aerial Robot Control of the iRobot

If it has been determined that the aerial robot has bumped the iRobot somewhere other than the magnet or front bumper, then a direct translation of the iRobot is applied, analogous to what would happen if a crash were to actually happen in the real world. In order to make this happen, first our Rotation matrix needs to be extended to a Homogeneous Transformation matrix, H that includes both Rotations, R, and Translations,T.

$$H = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}$$

$$\vec{p}_{t+1} = H * \vec{p}_t$$

H is a $3x3$ matrix with R being the rotation matrix from equation (4.1), and T is a 2x1 column vector to represent a translation. In the case of a contact with the iRobot not involving either of its control mechanisms there are no rotations and R just becomes the identity matrix, I. To determine the magnitude of the translation the physical overlap between the aerial robot and iRobot is used, and the direction is pointed directly away from the aerial robot.

$$T = \left(r.radius + t.radius - norm\left(\vec{t} - \vec{r}\right)\right)\left(\frac{\vec{t} - \vec{r}}{\left|\vec{t} - \vec{r}\right|}\right)$$

Once the appropriate H matrix is calculated, it is returned to the iRobot agent class so that the globTran property can be updated to represent the new position and orientation.

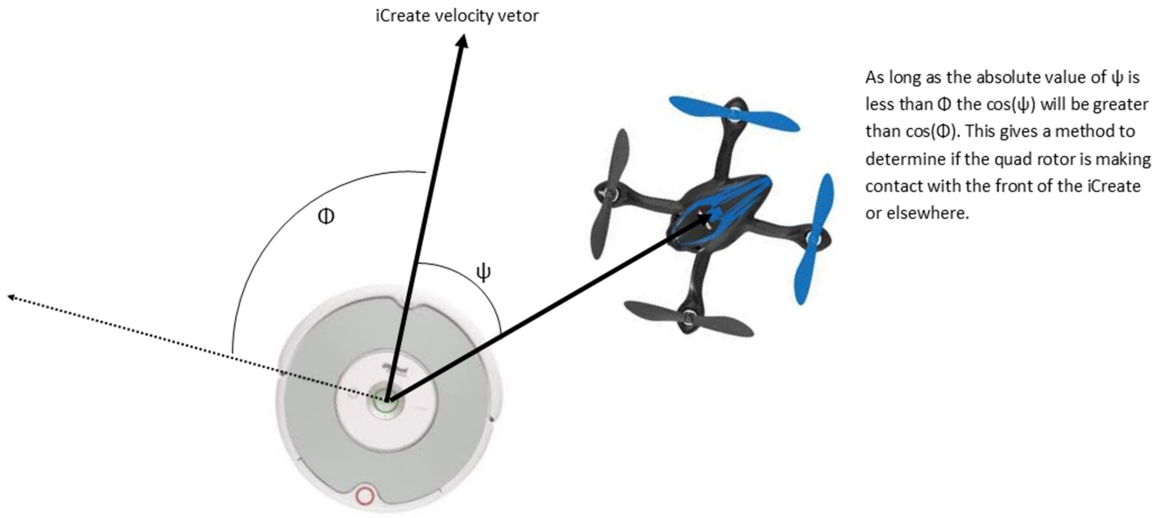


Figure 4.5 Correlation between iRobot Front Bumper and Quad Position

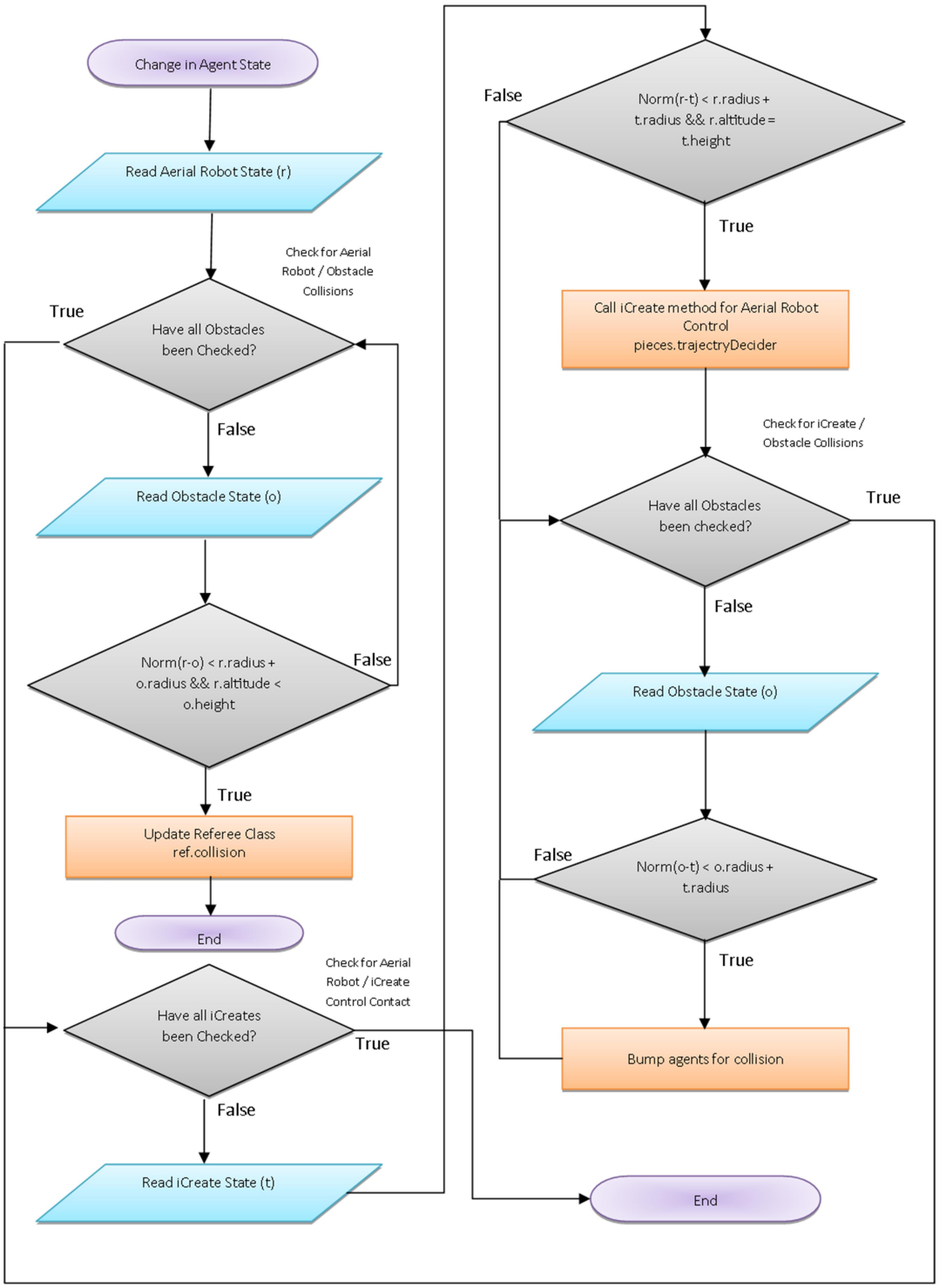


Figure 4.6 Program Flow for Collision Detection Block

Finally the iRobots are checked for collision with any other iRobots or any of the obstacles. This is performed in the exact same way as the aerial robot was checked, with the exclusion of the magnetic control on the top. Because all obstacles and iRobots are ground robots, no heights or altitudes are needed. Whether the agents are within each other's radii and whether the iRobot has made contact with the front bumper or not are the only parameters that need to be considered. Calculating rotation and translations follow the same logic as above; if the front bumper of the iRobot is hit reverse direction, and if not, simply bump the iRobot over.

CHAPTER FIVE

Hardware Progress

Significant progress has been made on the hardware implementation, however, there is still much left to accomplish before the UAV will be competition ready. The work load was divided into 3 phases: give the UAV control and vision to track one iRobot, give the UAV control and vision to track more than one iRobot and avoid obstacles, and then give the UAV mission level decision making capabilities along with safety features required by competition guidelines. If time permits we would like to embed all developed algorithms onboard the UAV, however we use a telemetry unit to communicate with the UAV and do all processing on a ground computer at the moment. Having the algorithms embedded on the UAV is not pertinent for the competition. Although phase 1 is not complete, it is all but complete, and while working on phase 1 some of the required algorithms for future phases were also implemented. In this chapter our UAV will be introduced along with details of our progress, and ideas as to the next steps needed to move forward.

When running the UAV simulation, it was assumed that all required data had been acquired by the sensors of the system and was readily available to the UAV. Unfortunately it is not that easy when dealing with the hardware implementation. In order to acquire the states of the target and obstacles in the field, computer vision algorithms have to be implemented. While acquiring data in successive time frames a memory block has to be built to give the UAV a sense of its surroundings wider than the camera view, and to allow for localization of itself. The UAV firmware and its communication

software also present a learning curve to getting the UAV to respond to the developed guidance algorithm. With these considerations in mind, the Senior Design team at Baylor was hired to research UAV's and select one appropriate for competing in the IARC. Based on high payload capabilities for carrying required telemetry and video peripherals, optimum performance, and open source firmware, their conclusion was that 3D Robotic's X8 would be the best candidate to complete the competition.

3D Robotic's X8

The X8 is called so because it is in the configuration of an X configured quadrotor, but is has 8 rotors total, one on top and one on bottom for every one rotor that a true quadrotor has. Refer to Figure 5.1 to see the configuration. The rotor pairs spin opposite direction which gives the X8 redundancy in the controllability of its yaw position, and a boost in performance from the added propulsion from an extra set of rotors. Each motor has its own electronic speed control (ESC) to maximize motor performance while tracking an input as shown in Figure 5.2. It can carry up to a 1.7 lb. payload, and fly for up to 15 minutes [33].

Figure 5.1 X8 from 3D Robotics

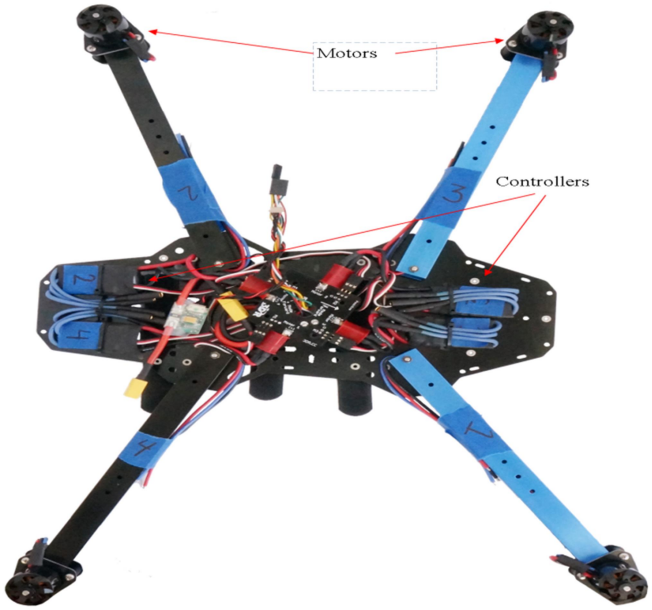


Figure 5.2 Electronic Speed Controllers

The X8 comes with a Pixhawk autopilot, a second generation to the PX4. The Pixhawk comes equipped with its own inertial measurement unit (IMU), microprocessor and backup microprocessor, and multiple interface connections for peripherals. In its entirety it has the following.

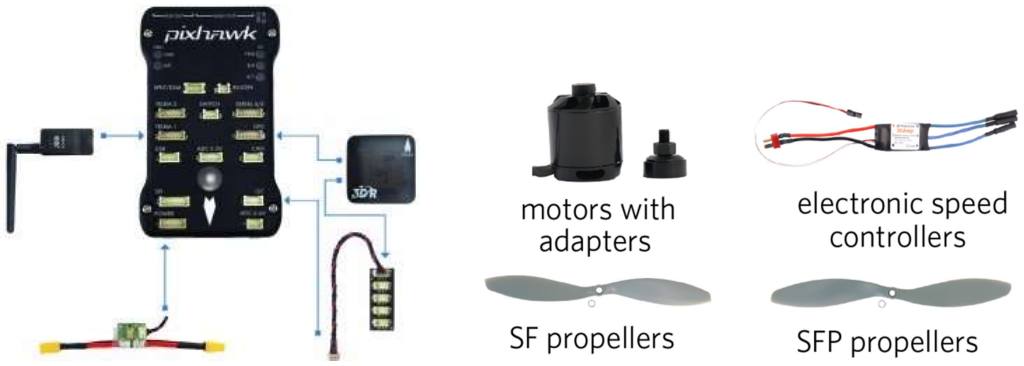
- Microprocessor:
  - 32-bit STM32F427 Cortex M4 core with FPU
  - 168 MHz/256 KB RAM/2 MB Flash
  - 32 bit STM32F103 failsafe co-processor

- Sensors (Stock sensors and GPS/Compass – we added an ultrasonic range sensor):
  - ST Micro L3GD20 3-axis 16-bit gyroscope
  - ST Micro LSM303D 3-axis 14-bit accelerometer / magnetometer
  - Invensense MPU 6000 3-axis accelerometer/gyroscope
  - MEAS MS5611 barometer
  - 3DR GPS and compass module w/ antenna, noise regulations, EEPROM and case

- Interfaces:
  - 5x UART (serial ports), one high-power capable, 2x with HW flow control
  - 2x CAN

- Spektrum DSM / DSM2 / DSM-X® Satellite compatible input up to DX8 (DX9 and above not supported)
- Futaba S.BUS® compatible input and output
- PPM sum signal
- RSSI (PWM or voltage) input
- I2C®
- SPI
- 3.3 and 6.6V ADC inputs
- External micro USB port

As add-ons, we purchased 3D Robotics' Mini high resolution 1/3" Sony Super HAD Color CCD 520TV line camera to give the X8 vision, and their self-made 915 MHz radios to allow the X8 to communicate with a ground station computer. Refer to Figure 5.3.



(a) Autopilot Assembly (b) Motor Assembly

Figure 5.3 X8 Electronic Components

Computer Vision

To give the UAV vision we used a Python version of openCV. Links to documentation and install files can be found in the appendix section OpenCV on page 119. The first type of vision we wished to give the UAV was the ability to see an iRobot directly below it with its downward facing camera. To do this we considered the iRobot as a simple circle and through setting thresholds on noise and the radius of the iRobot we

were able to use a Hough Transform to consistently get detection of multiple iRobots. Refer to Figure 5.4 for a code snippet showing the function and the parameters use for noise filtering. Through experimentation we found these parameters were required for an optimal detection rate with minimal false detections.

```
# Convert BGR to Gray Scale
imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

# cv2.HoughCircles(grayscaleImg,DetectionMethod,dp=1,MinDistBetweenCenters,
#                          CannyUpperThresh,CenterDetectThresh,MinRad,MaxRad)
cannyUpperThresh = 50    # Sensitivity of Edge Detector
centerThresh = 15        # Sensitivity of Circle Detector
minRadius = 85           # Min/Max radius of circles to detect
maxRadius = 87
minCenterDist = 2*minRadius # Minimum Distance between Circles
circles = cv2.HoughCircles(imgGray,cv.CV_HOUGH_GRADIENT,1,minCenterDist,
    param1=cannyUpperThresh,param2=centerThresh,minRadius=minRadius,maxRadius=maxRadius)
```

Figure 5.4 Circle Detection with Hough Transform

OpenCV packages the Hough Transform into one nice function, but inside the function many operations are occurring that a user should know about in order to properly adjust parameters and maximize performance. First, the video frame must change color spaces to a gray scale image. Then the frame is transformed with a Canny edge detection algorithm which transforms grayscale frames into black and white images where edges are white and the background is black. In the edge detection process, the derivative of each pixel's color is taken with respect to its position in both the x and y directions. In a gray scale image, pixels on edges have higher derivatives than pixels away from images so a threshold can be used to determine if a pixel should be white or black. However, noise tends to have a higher derivative; therefore a band pass filter must be used rather than simply a high pass filter. This upper noise threshold is what is being set by the cannyUpperThresh parameter in the Hough Transform [34]

Once the edges have been detected, the Hough Transform algorithm can be applied. Figure 5.5 illustrates the algorithm. From each edge lines are projected normal to the edge which give each pixel it passes through a weight. If the lines are weighted at 1 for example, then any pixel that has a line going through it will be weighted one, and any pixel lands where lines cross will have a weight equal to the number of lines crossing the pixel. Since the properties of circle make lines projected inward and normal to its surface cross the center of the circle, the centers of circles are where the highest weighted pixels will be [34]. To account for noise, the threshold with which the transform will detect a circle can be set by the centerThresh parameter, and a minimum distant can be set which governs how far apart two detections must be in order to be considered two circles. Then the radii of the detected circles can be restricted as a method to extract a desired circle from the list of detected circles, like an iRobot.

The feed from the onboard camera is transmitted to a computer on the ground through its own dedicated telemetry unit. Therefore, we can process the images on the computer and send control commands back in parallel.

To prepare for the competition, we needed to be able to handle detecting more than one iRobot at a time. We also needed to consider the fact that the X8 would only be equipped with a forward facing and a downward facing camera at the time of the competition, therefore it would not be able to see the entire competition field at all times. Lastly, the Hough Transform method produces both noisy measurements and false detections.

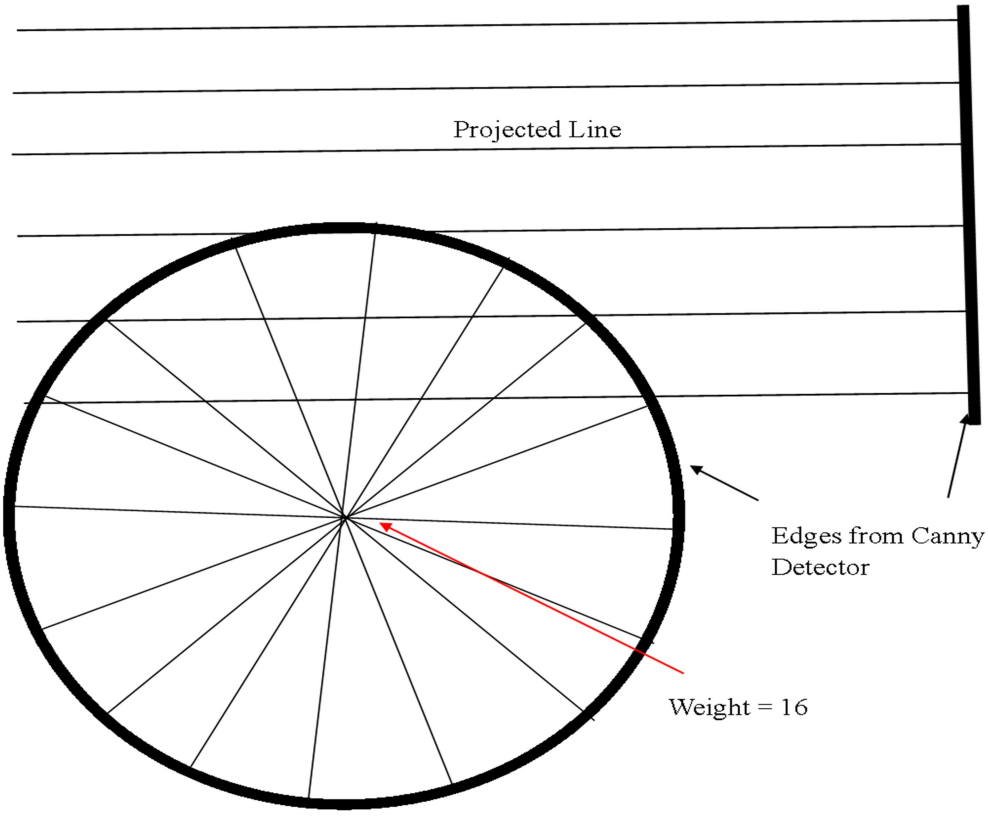


Figure 5.5 Hough Transform Algorithm

The specific algorithms needing developed were the ability to remember what iRobots were detected and where for a period of time, the ability to match detected iRobot states in successive time frames with iRobot states in the memory block, the ability to filter successive noisy measurements into a precise state measurement, and the ability to forget states that are products of false detection. The code to produce these abilities is too long to provide snippets here so the code in its entirety has been provided in the appendix section Multiple iRobot Detection and Memory Block on page 108.

We defined a class, Mem, which contained a block of memory for the states of the iRobot, a threshold for determining whether new detections were existing in the memory block already or if they were new iRobots, a method for correlating new detections with

the states in the memory block, and kalman filter methods to filter noisy measurements and make predictions on existing states in the memory block with no new detections.

To determine if a newly detected iRobot was already in the memory block we determined the norm of the vectors pointing from the iRobot to each element in the memory block and compared this to a predefined threshold.

$$norm(\boldsymbol{t} - \boldsymbol{m_i}) \leq h \tag{5.1}$$

$\boldsymbol{t}$ is the state vector of the iRobot detected, $\boldsymbol{m_i}$ represents each state vector in the memory block, and h represents a predefined threshold based on the shutter rate of 50 Hz of the camera [35]. If (5.1) is true then $\boldsymbol{m_i}$is updated with $\boldsymbol{t}$. If it is not true for all i then the iRobot is considered a new detection with no previous detections and added to the end of the memory block. In the competition there will only be 10 iRobots, so the memory block is capped at 10 to reduce the probability of a false detection entering the memory block as time passes. After a period of time, if no new measurement is made for a particular state in the block then the track is considered a noise track and dropped from the block to provide room for a positive detection.

As measurements come in from the camera they are passed through a kalman filter to remove any noise from successive measurements. The estimate of the kalman filter is what is used as the true detection of the iRobot. Once the state has been passed through the kalman filter, the same filter can be used to predict the state of the iRobot in the event of a failed detection. This will give the UAV a sense of its surroundings for a period of time after the iRobot has gone out of view of the camera. This sense of surrounding will get more inaccurate as time passes without a new detection. This idea, along with the time limit set to decay noise out of the memory block is the heuristic

intended to be used in developing the control for the yaw of the UAV. This controller has yet to be developed.

Control Software

To control the UAV we had to establish a communication loop with that allows us to both receive state parameters from the UAV and send control commands back. The position and velocity of the UAV can be determined through the camera feed, but for higher accuracy, data coming from the inertial measurement unit will also be taken and fused with the camera using a kalman filter.

The protocol used for communication with UAV's that have the ardupilot firmware is called MAVLink. Documentation on the structure of the protocol can be found by following the link on page 119. There is also a class of Python functions called pymavlink that the MAVLink user community has developed as a user friendly wrapper to MAVLink. When the functions are called, they invoke packaging and sending functions that structure the data according to the protocol and sends it to the UAV. Before any communication with the UAV can happen, a connection needs to be established and the baud rate defined using the function mavlink_connection(Port,baud=k), where Port is the COM port number the radio is connected to obtained from device manager in windows, and *k* is the baud rate that the radio is set to; usually 57600. To get states from the inertial measurement unit there is a set of param_fetch_* functions. To see the code implemented to fetch parameters refer to Fetch Parameters from UAV on page 112. Table 5.1 shows experiments performed to determine the lag time from the requesting parameters to receiving parameters. It turns out that the communication link from the computer to the UAV is slower than desired. The lag from both the parameter telemetry

unit and the camera telemetry unit is a reason to want to obtain both measurements and try to merge the data.

Table 5.1 Average Time to Receive Parameters

Timeout (s)	Avg params received	Avg time (s)	Trials
0	0.77	0.060050213	200
0.1	5.005	0.196600499	200
0.2	5.825	0.18172945	200
0.3	5.915	0.206181965	200
0.4	5.985	0.200707571	200
0.5	5.895	0.308520924	200
0.6	5.83	0.387252246	200
0.7	5.82	0.398728337	200
0.8	5.91	0.371041759	200
0.9	5.895	0.416568048	200
1	5.84	0.564434116	150
Total Avg	5.32372093	0.293088986	2150

Once we have information from the UAV we will want to adjust its states according to our control laws. We control the UAV by sending reference commands for attitude angles to the rc override ports. There are 8 total ports on the Pixhawk that can be used for commands via the r_channels_override_send function. Channel 1 controls the pitch, channel 2 controls the roll, channel 3 controls the throttle, and channel 4 controls the yaw. The channels are pwm which usually operate between 1165 pwm and 1600 pwm. These numbers can be set, but going lower than 1165 will cause the throttle safety to engage and leave the UAV stuck in land mode. A linear mapping is made between the minimum and maximum rc channel values and the minimum and maximum angles. Each angle will have a min and max value in the parameters. For instance YAW_MAX can be seen by receiving all the parameters, get_all_params. Each channel also has an RC_TRIM value that acts as neutral. It would normally be set to the midpoint of the min

and max on the channel and be used to represent an angle of zero. Since the throttle should never be set to zero in flight, the trim is set low and used as a safety feature to land the UAV if it is reached. With the linear mapping from pwm to radians in place, the velocity controller from our simulation can be used to set the desired angel. For test code used to test the rc override channels refer to the appendix on Motor Control on page 114.

CHAPTER SIX

Results

To show performance on our guidance algorithm we use the idea of comparing proportional navigation coupled with potential fields and just potential fields alone. The performance metric we utilize is the time to contact with the target, while insuring no obstacles are contacted. A monte carlo style of simulation is developed to test the algorithms in many scenarios. We will also show how gains were found for our linearized UAV model and show simulations of the performance of both the attitude controller and the velocity controller. Finally, we will demonstrate the guidance algorithm coupled with the model itself in a 3 dimensional environment.

Guidance Algorithm

To set up an initial test environment for the guidance algorithm a square area of obstacles were simulated. The square area consists of 49 obstacles with a radius of 1 meter, evenly spaced at 5 meters in both the x and y directions. First a particular scenario is chosen where the target moves along the top of the obstacle square from the west to the east, and the UAV is placed at the origin. Figure 6.1 demonstrates the scenario with the simple pursuit algorithm for tracking, along with potential fields for obstacle avoidance. As expected the UAV consistently heads directly toward the target, only veering off its course to avoid obstacles. With this algorithm it takes the UAV a total of 191.7 seconds to rendezvous with the moving target. Figure 6.1 shows a time stamp at every 20 seconds to add a time component to the path for easy understanding of what is going on. At the

actual time of contact, a black asterisk is displayed to show where, and the time of contact is mentioned towards the bottom of the figure, along with the legend. This format is the same for Figure 6.1-Figure 6.4 where both algorithms are shown with and without the optimizing factor applied to the repelling vector.

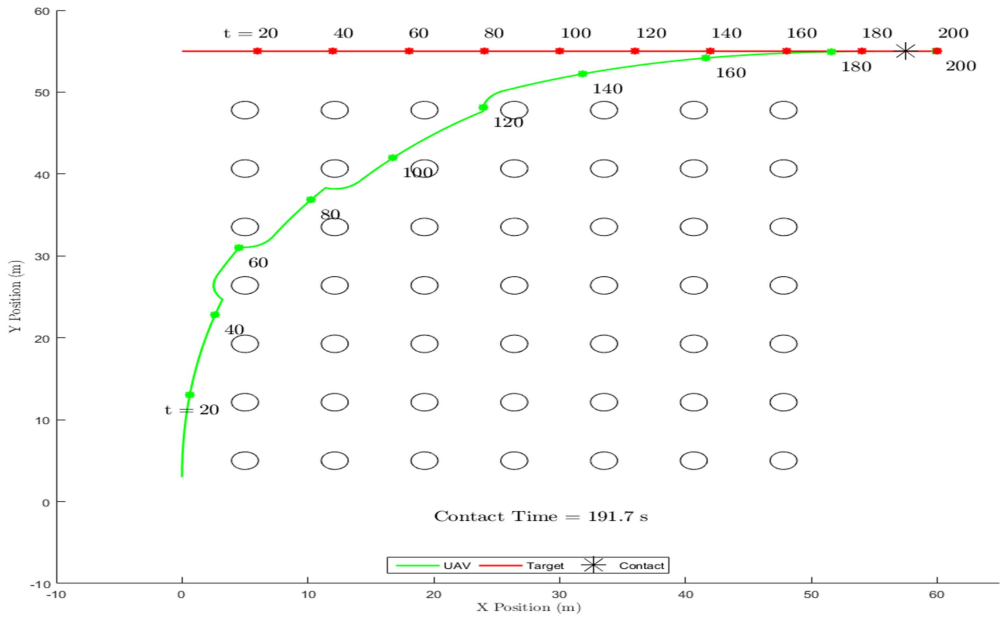


Figure 6.1 Pursuit Algorithm with Potential Fields

Notice that the path taken by the UAV contains large radial portions around the obstacles. We added the optimization scaling factor to the algorithm to see how it may impact the effectiveness of a simple pursuit algorithm. Figure 6.2 depicts the path with this added feature. As expected the radii of the obstacle avoidance portions of the path are reduced to allow the UAV to take a shorter overall path to the target. By adding the optimization factor, the UAV reduced its time to rendezvous by 6.5 seconds to 185.2 seconds.

The proportional navigation was then tested in the same scenario as the pursuit algorithm. Figure 6.3 shows this scenario. Notice the UAV takes almost a straight path to

the target, only veering off course in order to avoid obstacles. Without any optimized repelling vector, the proportional navigation algorithm still reduces the time to contact from the pursuit algorithm by over 20 seconds to 162.25 seconds.

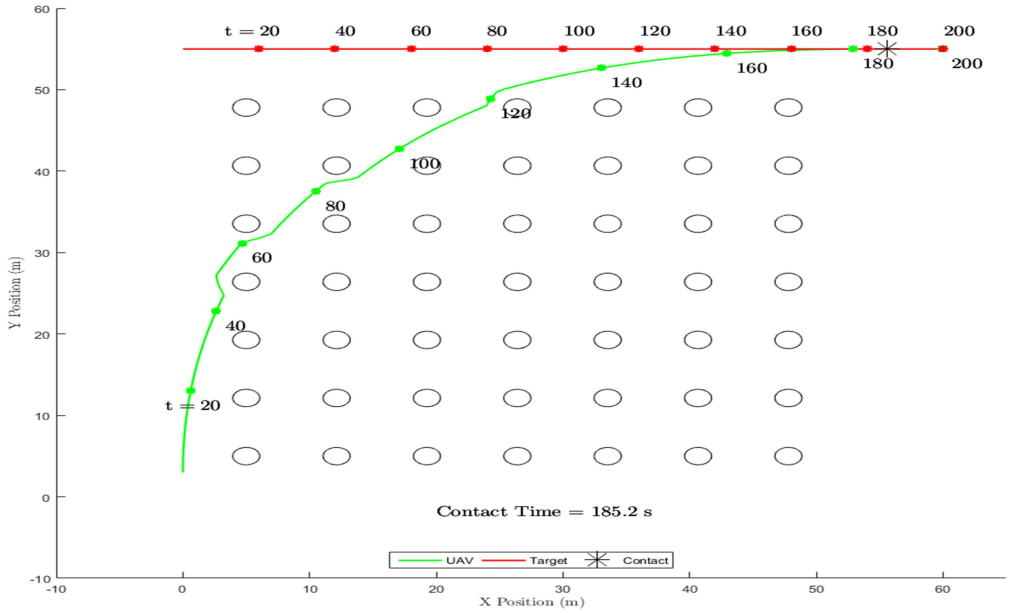


Figure 6.2 Pursuit with Optimize Repelling Function

Lastly, Figure 6.4 shows the scenario simulated with the proportional navigation algorithm and the repelling vector optimized. In this particular case the optimization only provides a minimum boost in performance when compared to the pursuit algorithm. We were able to gain an additional 1.5 seconds with the optimized algorithm. Although the time is minimal, the extra smoothness delivered from the algorithm puts less stress on the control structure to make fast adjustments and is therefore still a useful feature.

To make it easier to compare the 4 paths taken by the UAV, they have been casted together onto one plot in Figure 6.5. It is easier to see that optimized vector does indeed provide a smoother path for both algorithms. To be able to quantify the performance, we used the energy function developed by U_{art}. The faster this function is

minimized directly correlates with how fast the UAV reaches the target. Figure 6.6 shows the 4 algorithms' energy functions plotted with respect to time.

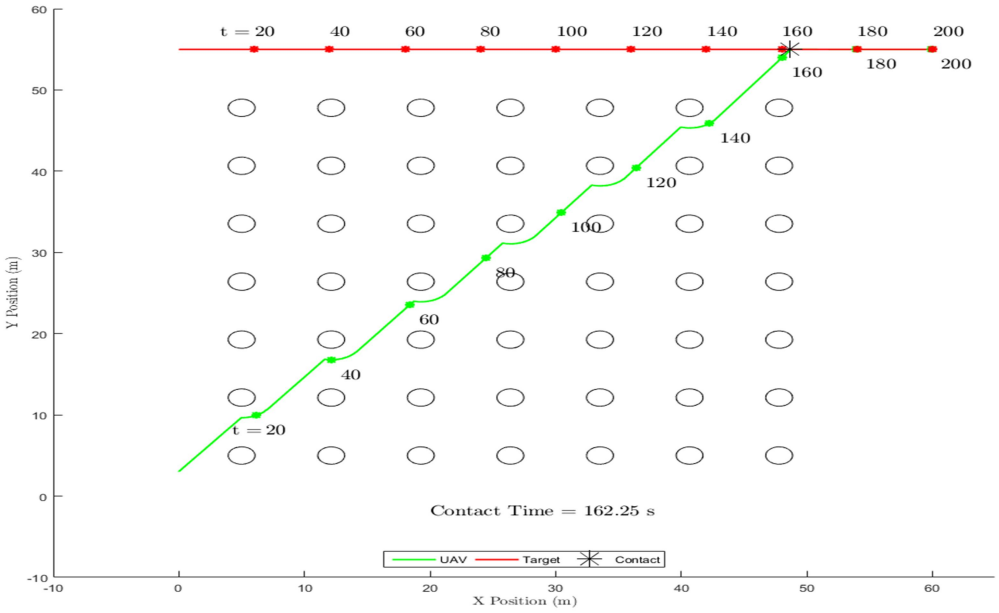


Figure 6.3 Proportional Navigation Algorithm with Potential Fields

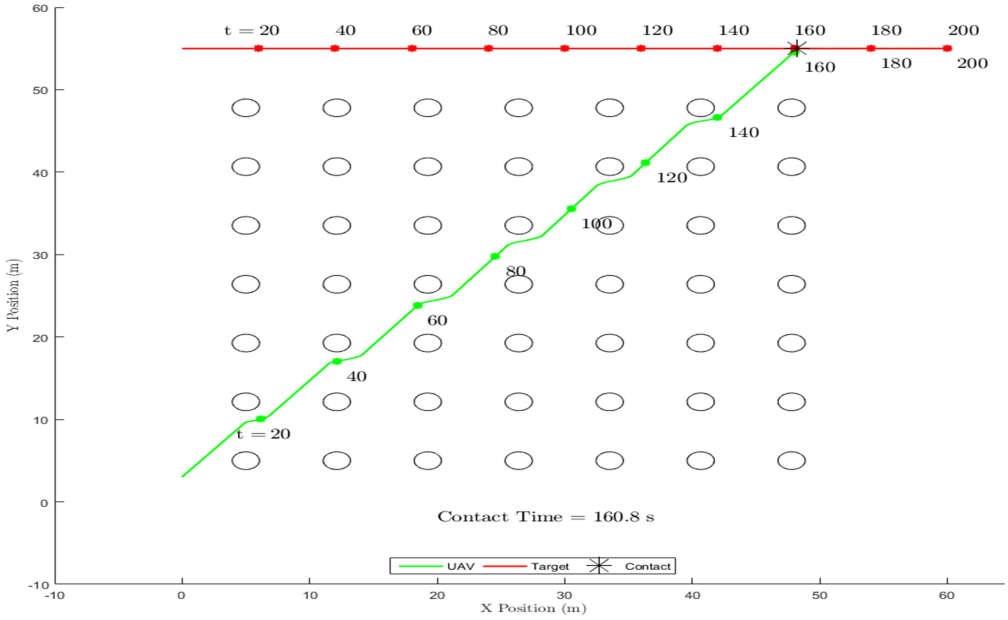


Figure 6.4 Proportional Navigation with Optimized Repelling Function

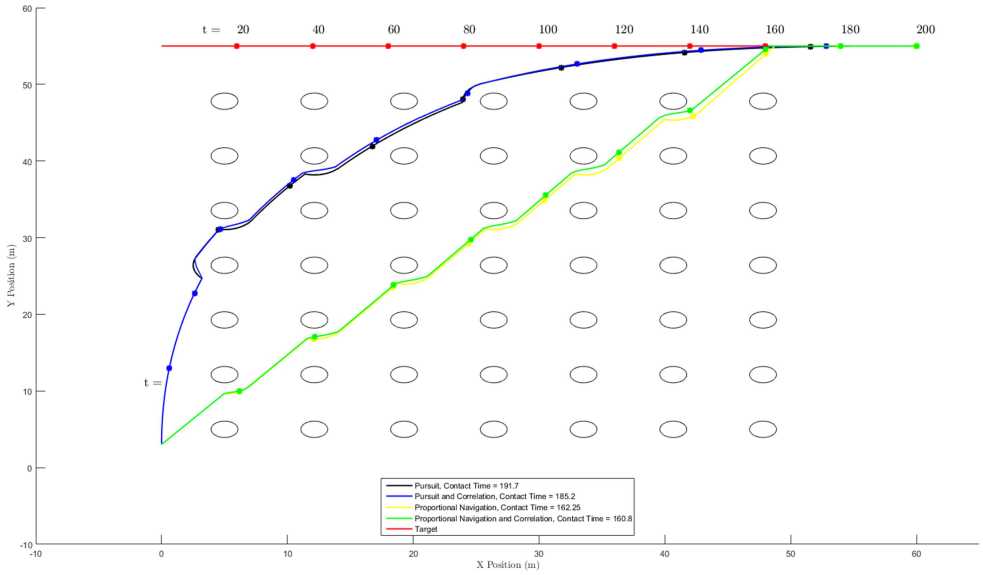


Figure 6.5 All Four Algorithms Together

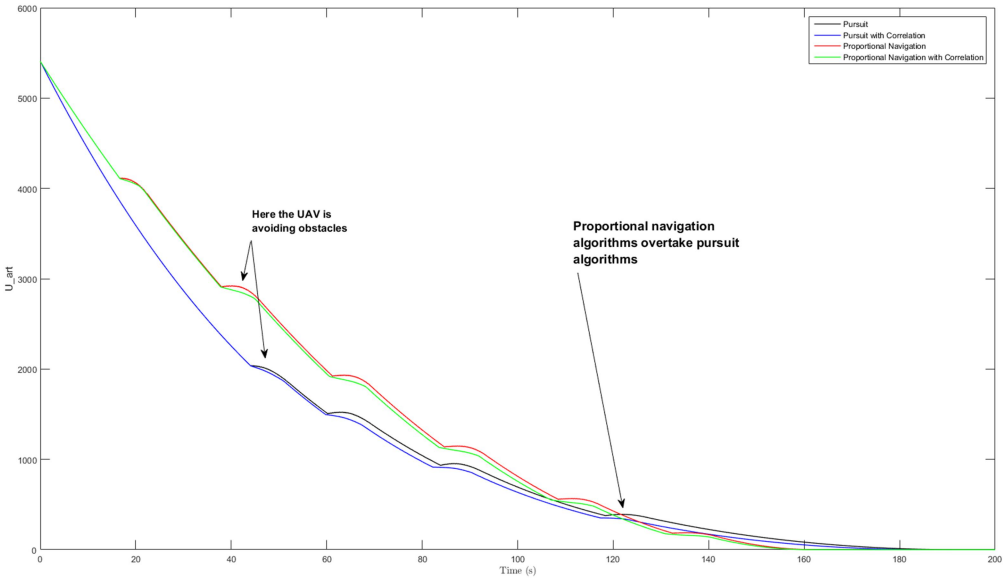


Figure 6.6 Energy of Four Algorithms

The sudden increases in energy are from the UAV having to avoid obstacles, and therefore momentarily losing ground on the target. It's interesting to see how the pursuit algorithms look superior to the proportional navigation algorithms upon first inspection. Since the UAV is pointed directly at the target, the closing velocity is much faster at the

start of the simulation, which decreases the energy function rapidly. Closer inspection shows that once the UAV has reached the targets path and is caught in a tail chase, it dissipates energy slower. For this reason, its energy function levels out while the potential field energy function decreases at a steady rate until the energy is minimized. Figure 6.7 shows Figure 6.6 zoomed in at 140 seconds. Notice that just after 120 seconds the proportional navigation algorithms overtake the pursuit algorithms. From this time to when the energy functions are minimized, we can see the 20 second gain that the proportional navigation techniques provide.

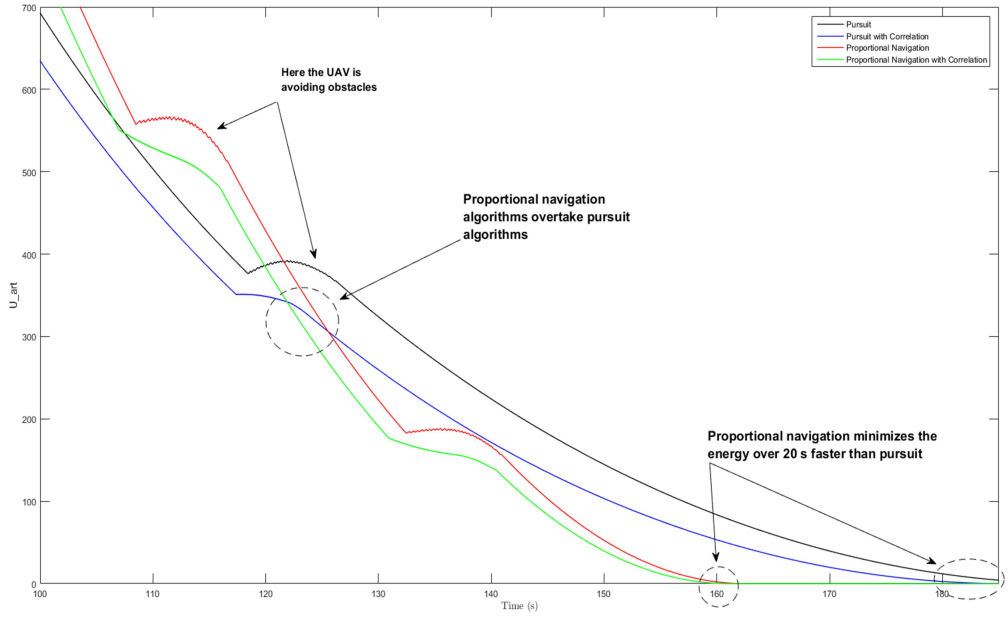


Figure 6.7 Zoomed Graph of Energies

After obtaining favorable results in our initial test scenario, we decided to expand the test to include a number of scenarios. To test a multitude of scenarios, we expanded the field to 100 X 100 meters. We tested performance by varying the number of obstacles in the field. For each set of obstacles, the course was ran 50 times, while perturbing the initial position of the UAV down the x axis by 2 meters for each trial. For a depiction of

this refer to Figure 6.8. Once the 50 scenarios are ran for both potential fields and potential fields combined with proportional navigation, the number of objects in the field is changed and another 50 scenarios are ran.

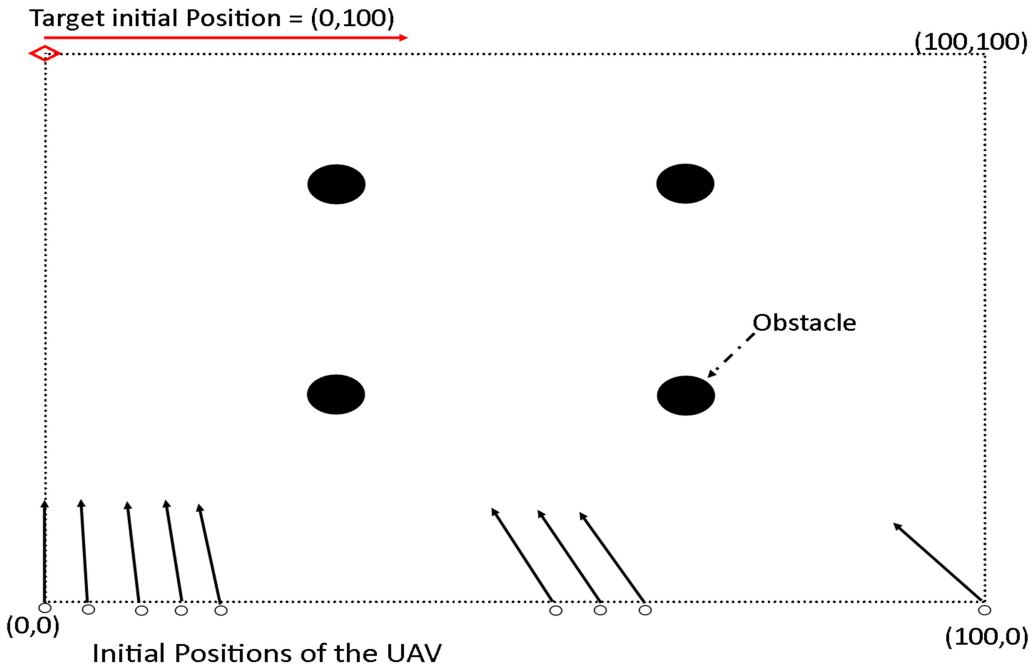


Figure 6.8 Test Scenario Setup

A total of 1200 scenarios were ran, 600 with potential fields and 600 with potential fields combined with proportional navigation. Figure 6.9 shows the time to contact with the potential field method verses the time to contact with the combined method. A line with a slope of 1 is plotted as a reference. Any trials that resulted in the both algorithms performing the same will land on this line. If the combined law performs better, the trial will be plotted below the line, and if the potential field theory alone performs better, the trial will be plotted above the line. The overwhelming majority of the trials turned out to have a better performance using the combined law method we developed.

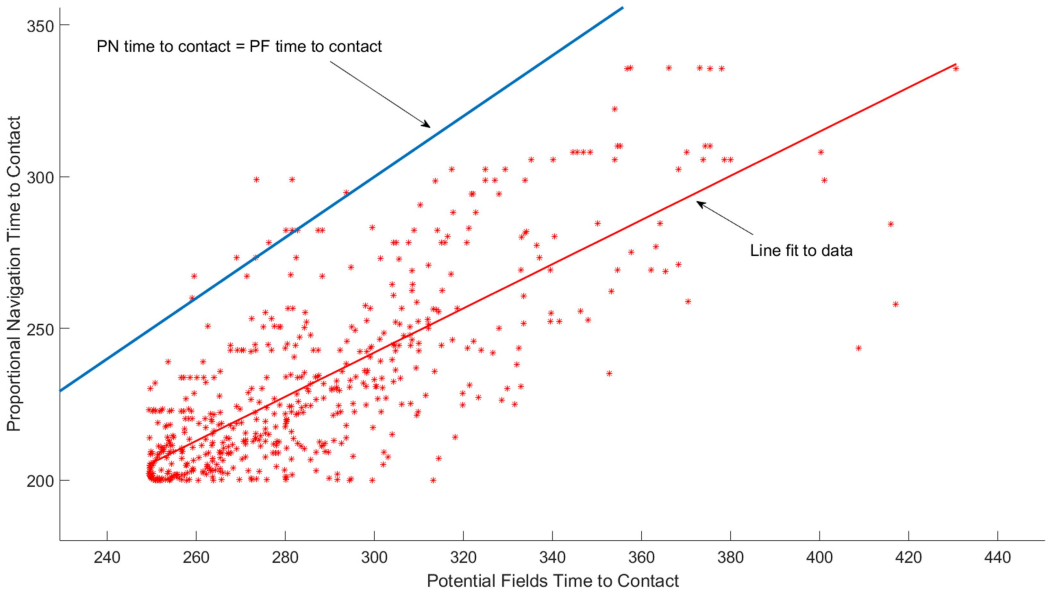


Figure 6.9 Comparison of Time to Contact

To quantify the increase in performance of the combined method, we took each data point and calculated the percent difference.

$$\frac{pf - pn}{pf} * 100$$

pf is the data collected using just the potential field method, and pf is the data collected using the combined method. Plotting the data in a histogram shows that the combined algorithm gives a 15%-25% increase in performance. Refer to Figure 6.10.

All data collected from a respective obstacle configuration was then averaged, and the mean time to contact was compared to the density of obstacles in the field. Refer to Figure 6.11. As expected, as the density of the obstacles increase, the average time to contact also increases. The average time to contact is still improved roughly 20% over the domain of densities.

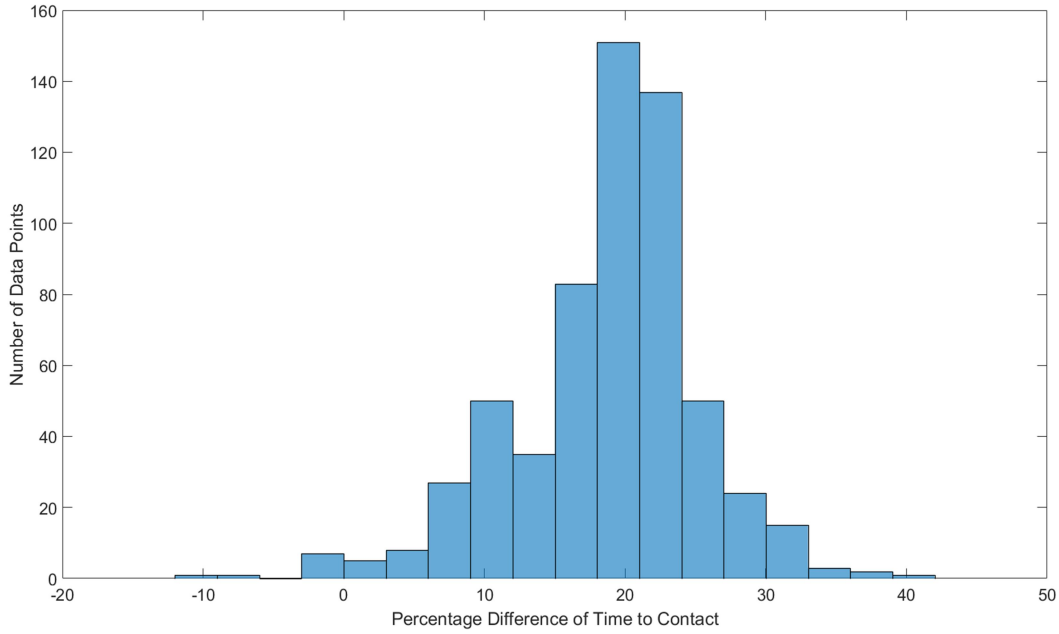


Figure 6.10 Percent Difference of Algorithm Performances

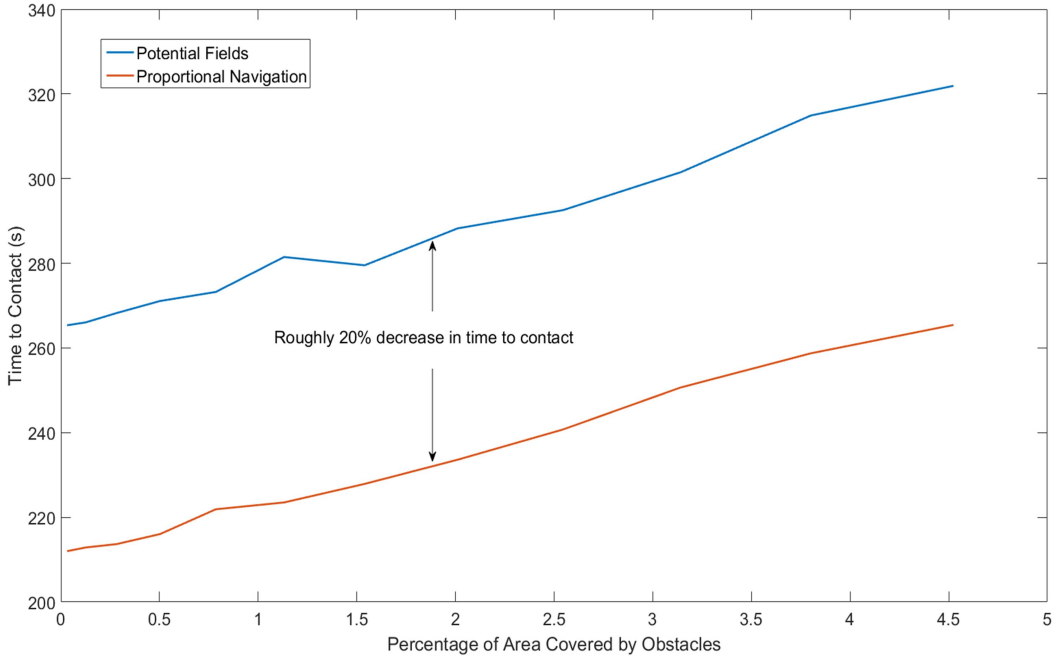


Figure 6.11 Mean Time to Contact by Density of Obstacles

UAV Model

To implement the model, gains were searched for and found using a simple particle swarm optimization search, PSO. The basic idea of PSO is to create possible solutions to the search either randomly or by some other method, and treat them as if individually they are simpletons that follow a simple rule or two, with the hope that the group as a whole will accomplish a complex task like search. In this search we set the velocity of the individual agents by

$$v_i = c_1 r_1 (GB - S_i) + c_2 r_2 (PB_i - S_i)$$

where c_1, c_2 are the chosen by the designer as a maximum step size for the agent. Care must be taken to choose an appropriate step size because an agent may get stuck in a local minimum if the step size is too small, or jump over the global maximum if the step size in too large. r_1, r_2 are both random numbers between 0 and 1. S_i is the individual agent, PB_i is the personal best solution that the agent has found as of yet, and GB is the best solution that any agent has found as of yet. In essence the algorithm drives the agents toward the global best, while also driving them to their personal best to see if they can do better than the current global best. Table 6.1 shows the final solutions found by the PSO.

To be more specific the gains for the attitude controller only were found by the PSO, as the gains for the velocity controller were simple to find manually. Figure 6.12 shows the control commands and the response from $\phi, \theta,$ and ψ. The coupling of the motors to each orientation angle becomes obvious when plotting the control commands. While each angle individually performs well, each angle is also perturbed from its steady state by other angles undergoing a change in control reference. These perturbations have

been circled in the figure. The three graphs are all relevant to the same time axis as the bottom figure.

Table 6.1 Controller Gains found with PSO

Gain	Value	Description
$k_{p,\phi}$	151.2	Proportional gain for the rotation about x
$k_{d,\phi}$	10.2	Differential gain for the rotation about x
$k_{p,\theta}$	151.2	Proportional gain for the rotation about y
$k_{d,\theta}$	10.2	Differential gain for the rotation about y
$k_{p,\psi}$	216.4	Proportional gain for the rotation about z
$k_{d,\psi}$	28.3	Differential gain for the rotation about z
$k_{p,x}$	.025	Proportional gain for the velocity along x
$k_{p,y}$	.025	Proportional gain for the velocity along y
$k_{p,z}$	.5	Proportional gain for the velocity along z

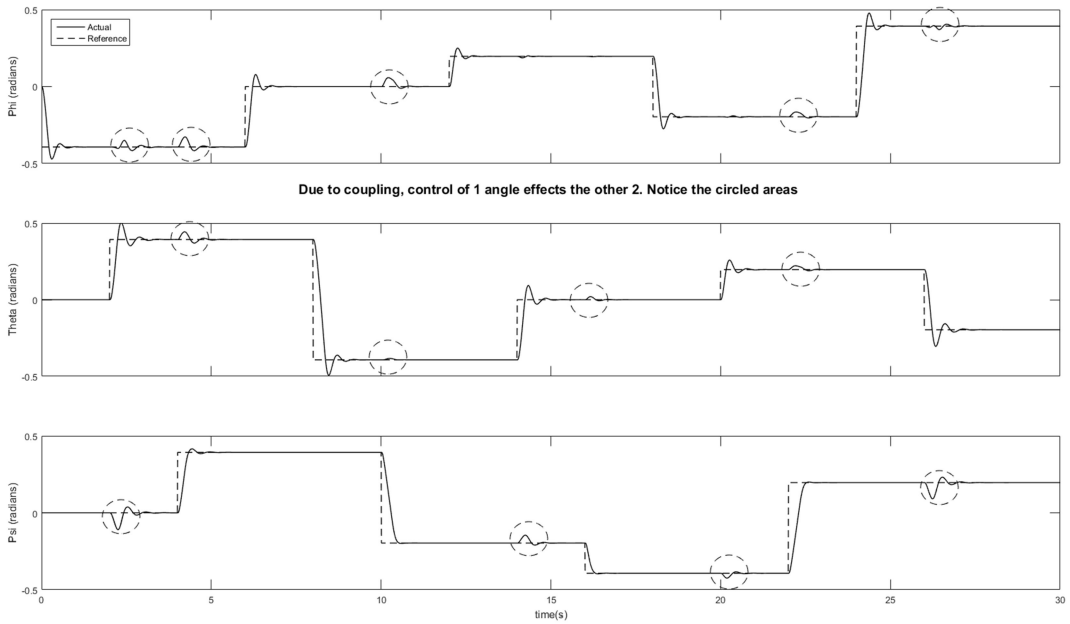


Figure 6.12 Attitude Control of UAV

Notice that each circled perturbation is paired with a control perturbation from one of the other angles. This, along with the errors in large perturbations due to linearizing the model forced us to restrict ϕ and θ to less than a $\frac{\pi}{8}$ perturbation from 0.

Furthermore, ψ was able to reach the full 2π circumference as is necessary, but the reference command can only perturb is at $\frac{\pi}{8}$ increments from its current position.

After the attitude controller was working efficiently, the velocity controller had to be tuned also. Considering that the controller is only a proportional controller with 3 gains, it was easily tuned by manual experimentation. Figure 6.13 shows the performance of the velocity controller in the $x, y,$ and z directions.

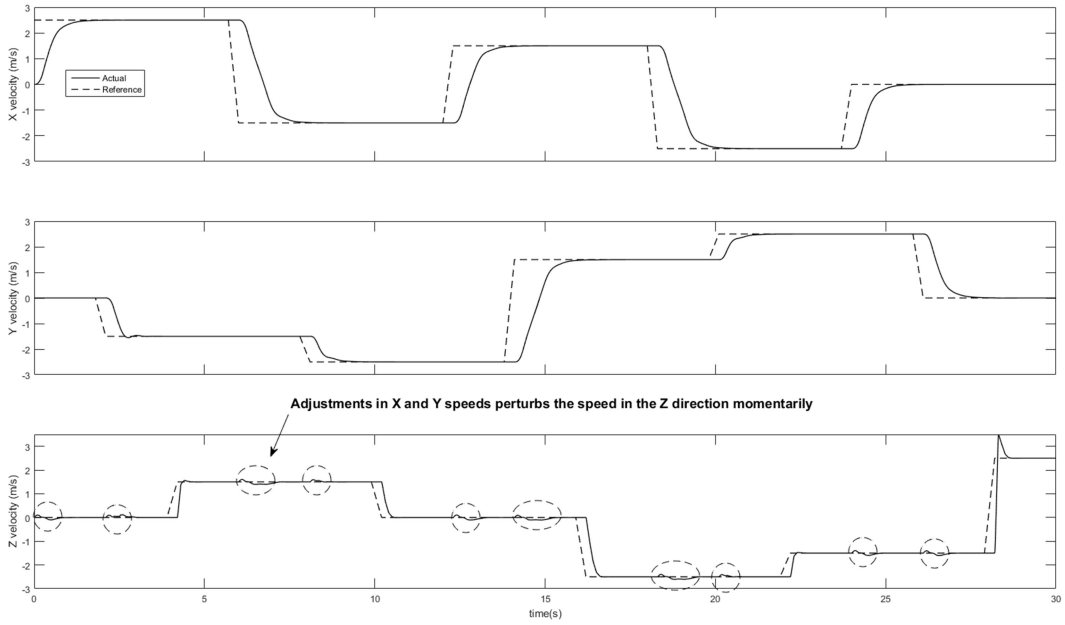


Figure 6.13 Velocity Control of UAV

Compared to the attitude controller, the velocity controller has lag when first perturbed. Anytime the UAV is tilted in order to produce a velocity in either the x or y directions, it begins lose altitude briefly until the motors can catch up. The bottom figure is the z direction. Notice, like the attitude controller, the z component is coupled to the x, y component such that any controller commands given to the x and y components cause a momentary perturbation of the z velocity component. Furthermore, if the velocity is

controlled too quickly, the UAV becomes unstable. By experiment it was found that the UAV model could travel at a max of about 6 m/s, or about 11 mph.

Combined Algorithm on Quad Model

The combined algorithm was placed on the linearized UAV model. Refer to Figure 6.14 and Figure 6.15 to see an example with one obstacle to avoid, and an example with multiple obstacles. The UAV successfully avoids the obstacles and rendezvous with the target as expected.

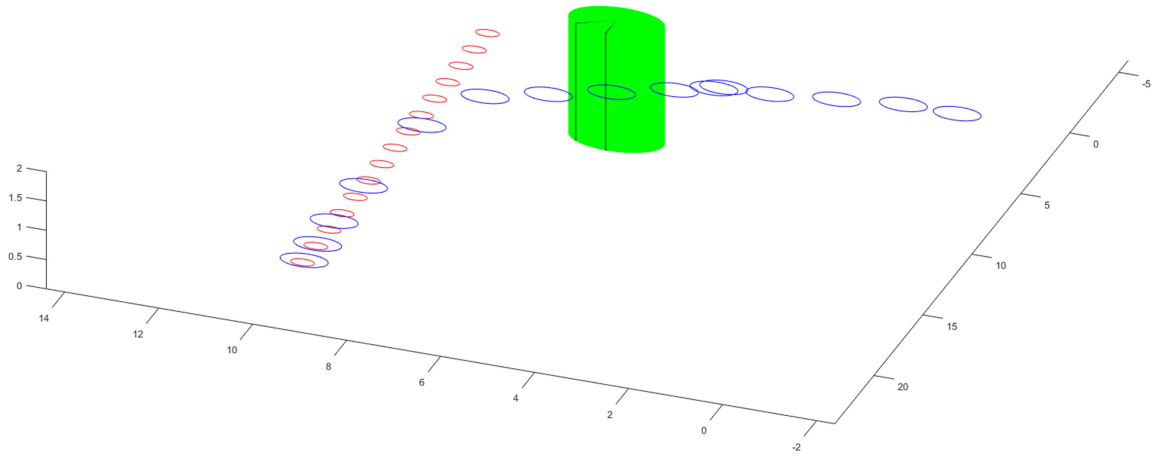


Figure 6.14 Quad Model with One Obstacle

The desired altitude when in the far field from the target was set to 2 meters. The UAV is successfully guided by the combined law to 2 meters while simultaneously taking an optimum path to the target, and taking obstacles into consideration. Once the UAV reaches the short field of the target, the artificial target position is moved back to the top of the target, and the combined law successfully guides the UAV back down to rendezvous with the target.

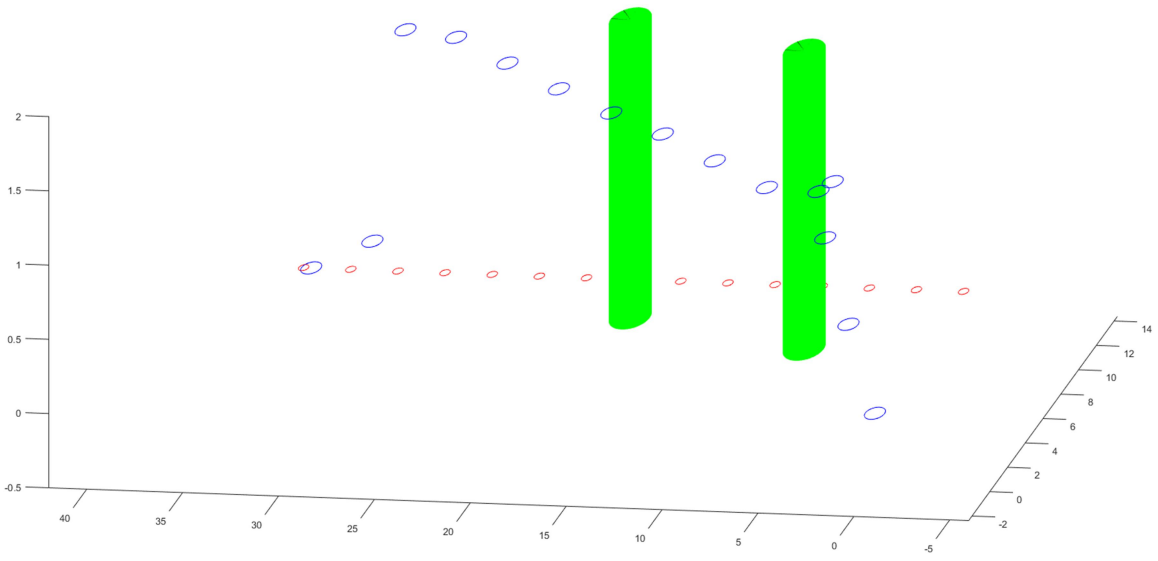


Figure 6.15 Quad Model with Multiple Obstacles

APPENDICES

APPENDIX A

Rotations

A detailed explanation of rotation matrices can be found in [54] and [55]. They are also well documented in many places online. They can also be derived from by considering them as a change of basis. They are of a class of matrices called special orthogonal matrices, $SO(n)$, where n is the dimension of the matrix. For our purposes we only need to know that they can be used to rotate a reference frame around a particular axis, and the properties of the matrix that are important for that purpose.

Representing Frames with Rotation Matrices

Consider Figure A.1 where a point is placed at $P(x_2, y_2, z_2)$ and we want a representation of the point in the standard Cartesian coordinate system, $\{x_1, y_1, z_1\}$. For simplification the point is in the x, y plane in both coordinate systems. In both coordinate systems the basis vectors are unit vectors. The 2 reference frame has been rotated by ψ about the z axis which would be pointing our of the page. We are looking for a matrix that will perform a transformation on P to give a new set of coordinates that are a representation of P in the 1 frame. By taking the projection of the 2 frame onto the 1 frame we get

$$R = \begin{bmatrix} x_2 \cdot x_1 & y_2 \cdot x_1 & z_2 \cdot x_1 \\ x_2 \cdot y_1 & y_2 \cdot y_1 & z_2 \cdot y_1 \\ x_2 \cdot z_1 & y_2 \cdot z_1 & z_2 \cdot z_1 \end{bmatrix} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

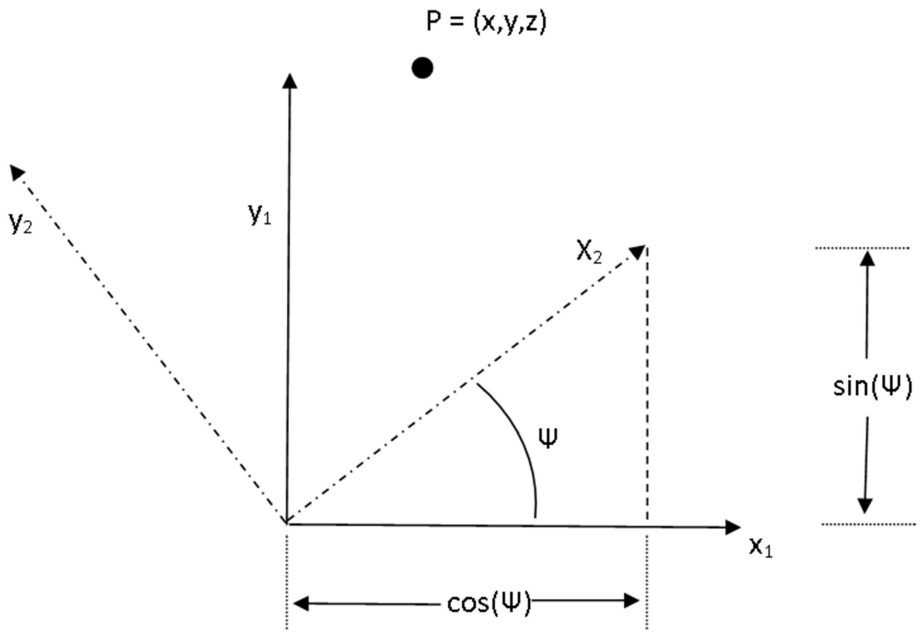


Figure A.1 Rotation about the Z Axis

Since this is a projection onto the 1 frame, we could use this matrix to transform a point given to us in the 2 frame to a representation of that point in the 1 frame.

$$p_1 = Rp_2$$

Similarly, if we want to rotate a set of points from the 2 frame to the 1 frame

$$p_1 = R[p_2^1 \dots p_2^n]$$

where the superscripts represent the individual points.

If we want to move from $\{x_1, y_1, z_1\}$ to $\{x_2, y_2, z_2\}$ we would be rotating by $-\psi$.

$$R(-\psi) = \begin{bmatrix} \cos(-\psi) & -\sin(-\psi) & 0 \\ \sin(-\psi) & \cos(-\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} = R(\psi)^T$$

It's is trivial to see that the inverse of a rotation is a rotation of equal magnitude in the opposite direction, or through an angle with equal magnitude and opposite signs. The

rotation in the opposite direction can be obtained through the transpose of the original rotation matrix, therefore

$$R^{-1} = R^T$$

In the section Equations of Motion on page 13 we want to represent points given in the Y configuration to points represented in the X configuration. This can be thought of as rotating the UAV $\psi = \frac{\pi}{4}$ radians, or as rotating the reference frame $\psi = -\frac{\pi}{4}$ radians. Which leads to our rotation matrix for R_M.

$$R_M = \begin{bmatrix} \cos\left(\frac{\pi}{4}\right) & \sin\left(\frac{\pi}{4}\right) & 0 \\ -\sin\left(\frac{\pi}{4}\right) & \cos\left(\frac{\pi}{4}\right) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Using the same approach as rotating around the z axis, rotation matrices about the x and y axis can be found also. In summary the three rotation matrices are

$$R_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$R_x(\phi) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

Representing Frames with Euler Angles

The orientation of a UAV represented in the earth frame may have to be rotated about all three axes. To handle this Euler angles are commonly used. Consider the UAV represented in the body frame, $\{x_b, y_b, z_b\}$, and we want the orientation in the earth frame, $\{x_E, y_E, z_E\}$. To obtain this representation we consider having two intermediate

frames, $\{x_1, y_1, z_1\}$, and $\{x_2, y_2, z_2\}$. Then we set an order with which we will use our rotation matrices to move through these frames, and we keep that order for all calculations. In this thesis we transform from $\{x_b, y_b, z_b\}$, to $\{x_1, y_1, z_1\}$ by rotating about the x_b axis. Then we go from $\{x_1, y_1, z_1\}$ to $\{x_2, y_2, z_2\}$ by rotating about the y_1 axis. Finally the $\{x_E, y_E, z_E\}$ representation is found by rotating about the z_2 axis. These rotations can be presented as one rotation matrix by left multiplying successive rotation matrices.

$$R_E = R_z(\psi) * R_y(\theta) * R_x(\phi)$$

To save space we let $\cos(\theta) = c_\theta$ and $\sin(\theta) = s_\theta$.

$$R_E = \begin{bmatrix} c_\psi c_\theta & s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\psi s_\phi + c_\phi s_\theta c_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\psi s_\theta c_\phi - c_\psi s_\phi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix}$$

Now any vector represented in the body frame can be represented in the earth frame by

$$\vec{v}_E = R_E \vec{v}_b$$

Body Frame Angular Rates and Euler Angle Rates

To represent the change in Euler rates we cannot simply transform the body rates to the earth frame. This would represent the angular rates of the UAV with respect to the earth frame, but the Euler rates represent not only the earth frame, but also two intermediate frames. In our construction of the R_E matrix, we used a combination that rotated first by ϕ around the x axis of the UAV body frame, and then by θ around this rotated intermediate frame, and then by ψ around this second rotated intermediate frame to finally end up in the earth frame. We have to determine how to each of these angles change in their respective frames, relative to the angular rates in the body frame.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & s_\phi \\ 0 & -s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & s_\phi \\ 0 & -s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} c_\theta & 0 & -s_\theta \\ 0 & 1 & 0 \\ s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix}$$

$$+ \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & s_\phi \\ 0 & -s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} c_\theta & 0 & -s_\theta \\ 0 & 1 & 0 \\ s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} c_\psi & s_\psi & 0 \\ -s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix}$$

This reduces to

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -s_\theta \\ 0 & c_\phi & s_\phi c_\theta \\ 0 & -s_\phi & c_\phi c_\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix}$$

Then by inverting this equation, Equation (2.10) is obtained.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} 1 & \frac{s_\phi s_\theta}{c_\theta} & \frac{c_\phi s_\theta}{c_\theta} \\ 0 & c_\phi & -s_\phi \\ 0 & \frac{s_\phi}{c_\theta} & \frac{c_\phi}{c_\theta} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

APPENDIX B

Feedback Linearization

To linearize the state space model a truncated Taylor Series is used. If $f = f(x_1, x_2 \dots x_n)$ is a nonlinear equation, then f can by approximated by a linear equation about an operating point, $\{x_{1_0}, x_{2_0}, \dots, x_{n_0}\}$ by

$$f \approx f(x_{1_0}, x_{2_0}, \dots, x_{n_0}) + \sum_i \frac{\partial f(\ _{1_0}, x_{2_0}, \dots, x_{n_0})}{\partial x_i}(x_i - x_{i_0})$$

For our model we choose an operating point based on the UAV hovering in place, $\{\omega_i = \omega_h, \psi = \psi, p = 0, q = 0, r = 0, \theta = 0, \phi = 0$. Due to the nature of the dependency of the UAV velocity on the yaw angle, ψ, the operating point is left floating for better performance.

Angular Acceleration

Here we'll copy equation (2.8) of our state model.

$$\dot{\boldsymbol{\Omega}} = J^{-1}\begin{bmatrix} \frac{1}{\sqrt{2}}LK_F[(\omega_2^2 + \omega_3^2) - (\omega_1^2 + \omega_4^2)] \\ \frac{1}{\sqrt{2}}LK_F[(\omega_3^2 + \omega_4^2) - (\omega_1^2 + \omega_2^2)] \\ K_M\big((\omega_1^2 + \omega_3^2) - (\omega_2^4 + \omega_4^4)\big) \end{bmatrix}$$

Handling each element of $\dot{\boldsymbol{\Omega}}$ independently, $\{\dot{p}, \dot{q}, \dot{r}\}$, the truncated Taylor series is used to establish a linearized approximation of (2.8).

$$\dot{p}(\omega_1, \omega_2, \omega_3, \omega_4)$$

$$\approx \dot{p}(\omega_h, \omega_h) + \frac{\partial \dot{p}(\omega_h, \omega_h)}{\partial \omega_2}(\omega_2 - \omega_h) + \frac{\partial \dot{p}(\omega_h, \omega_h)}{\partial \omega_4}(\omega_4 - \omega_h)$$

$$+ \frac{\partial \dot{p}(\omega_h, \omega_h)}{\partial \omega_3}(\omega_3 - \omega_h) + \frac{\partial \dot{p}(\omega_h, \omega_h)}{\partial \omega_1}(\omega_1 - \omega_h)$$

$$= \frac{LK_F}{\sqrt{2}J_{xx}}\big(2\omega_h(\omega_2 - \omega_h) - 2\omega_h(\omega_4 - \omega_h) - 2\omega_h(\omega_1 - \omega_h)$$

$$+ 2\omega_h(\omega_3 - \omega_h)\big) = \frac{2\omega_h LK_F}{\sqrt{2}J_{xx}}\big((\omega_3 + \omega_2) - (\omega_1 + \omega_4)\big)$$

$$\dot{q}(\omega_1, \omega_2, \omega_3, \omega_4)$$

$$\approx \dot{q}(\omega_h, \omega_h) + \frac{\partial \dot{q}(\omega_h, \omega_h)}{\partial \omega_2}(\omega_2 - \omega_h) + \frac{\partial \dot{q}(\omega_h, \omega_h)}{\partial \omega_4}(\omega_4 - \omega_h)$$

$$+ \frac{\partial \dot{q}(\omega_h, \omega_h)}{\partial \omega_3}(\omega_3 - \omega_h) + \frac{\partial \dot{q}(\omega_h, \omega_h)}{\partial \omega_1}(\omega_1 - \omega_h)$$

$$= \frac{LK_F}{\sqrt{2}J_{yy}}\big(2\omega_h(\omega_4 - \omega_h) - 2\omega_h(\omega_2 - \omega_h) - 2\omega_h(\omega_1 - \omega_h)$$

$$+ 2\omega_h(\omega_3 - \omega_h)\big) = \frac{2\omega_h LK_F}{\sqrt{2}J_{yy}}\big((\omega_4 + \omega_3) - (\omega_1 + \omega_2)\big)$$

$$\dot{r}(\omega_1, \omega_2, \omega_3, \omega_4)$$

$$\approx \dot{r}(\omega_h, \omega_h) + \frac{\partial \dot{r}(\omega_h, \omega_h)}{\partial \omega_2}(\omega_2 - \omega_h) + \frac{\partial \dot{r}(\omega_h, \omega_h)}{\partial \omega_4}(\omega_4 - \omega_h)$$

$$+ \frac{\partial \dot{r}(\omega_h, \omega_h)}{\partial \omega_3}(\omega_3 - \omega_h) + \frac{\partial \dot{r}(\omega_h, \omega_h)}{\partial \omega_1}(\omega_1 - \omega_h)$$

$$= \frac{K_M}{J_{zz}}\big(-2\omega_h(\omega_2 - \omega_h) - 2\omega_h(\omega_4 - \omega_h) + 2\omega_h(\omega_1 - \omega_h)$$

$$+ 2\omega_h(\omega_3 - \omega_h)\big) = \frac{2\omega_h K_M}{J_{zz}}\big((\omega_1 + \omega_3) - (\omega_2 + \omega_4)\big)$$

Consider the following relationships.

$$\Delta\omega_{\phi} = (\omega_2 + \omega_3) - (\omega_1 + \omega_4)$$

$$\Delta\omega_{\theta} = (\omega_3 + \omega_4) - (\omega_1 + \omega_2)$$

$$\Delta\omega_{\psi} = (\omega_1 + \omega_3) - (\omega_2 + \omega_4)$$

Now by direct substitution we obtain the linearized model used in this thesis.

$$\dot{p}(\omega_1, \omega_2, \omega_3, \omega_4) \approx \frac{2\omega_h L K_F}{\sqrt{2} J_{xx}} \Delta\omega_{\phi}$$

$$\dot{q}(\omega_1, \omega_2, \omega_3, \omega_4) \approx \frac{2\omega_h L K_F}{\sqrt{2} J_{yy}} \Delta\omega_{\theta}$$

$$\dot{r}(\omega_1, \omega_2, \omega_3, \omega_4) \approx \frac{2\omega_h K_M}{J_{zz}} \Delta\omega_{\psi}$$

We can further implement it as a state space model simply expressing the equations in matrix form.

$$\dot{\boldsymbol{\Omega}}_{\boldsymbol{ref}} \approx \begin{bmatrix} \frac{2\omega_h L K_F}{\sqrt{2} J_{xx}} & 0 & 0 \\ 0 & \frac{2\omega_h L K_F}{\sqrt{2} J_{yy}} & 0 \\ 0 & 0 & \frac{2\omega_h L K_M}{J_{zz}} \end{bmatrix} \begin{bmatrix} \Delta\omega_{\phi} \\ \Delta\omega_{\theta} \\ \Delta\omega_{\psi} \end{bmatrix}$$

Linear Acceleration

For convenience equation (2.9) is copied here.

$$\ddot{\vec{\mathrm{r}}} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \frac{1}{m} R_E \begin{bmatrix} 0 \\ 0 \\ K_F \sum_i \omega_i^2 \end{bmatrix}$$

By substituting R_E and simplifying we get the following.

$$\ddot{\vec{\mathrm{r}}} = \frac{K_F \sum_i \omega_i^2}{m} \begin{bmatrix} s_{\psi} s_{\phi} + c_{\phi} s_{\theta} c_{\psi} \\ s_{\psi} s_{\theta} c_{\phi} - c_{\psi} s_{\phi} \\ c_{\phi} c_{\theta} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

Now the truncated Taylor series will be applied to each element of $\dot{\boldsymbol{v}}$, $\{\ddot{x}, \ddot{y}, \ddot{z}\}$, to obtain a hybrid linearized model. It is considered a hybrid model because there will not be a set operating point for ψ.

$$\ddot{x} = \frac{K_F}{m}\Big((\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2)\big(c_\phi s_\theta c_\psi + s_\psi s_\phi\big)\Big)$$

$$\ddot{x}(\omega_h, \psi, 0) = 4\omega_h^2 \frac{K_F}{m}(\cos(0)\sin(0)\cos(\psi) + \sin(0)\sin(\psi)) = 0$$

$$\sum_i \frac{\partial \ddot{x}(\omega_h, \psi, 0)}{\partial \omega_i}(\omega_i - \omega_h)$$

$$= 2\omega_h \sum_i (\cos(\psi)\sin(0)\cos(0) + \sin(0)\sin(\psi))(\omega_i - \omega_h) = 0$$

$$\frac{\partial \ddot{x}(\omega_h, \psi, 0)}{\partial \phi}(\phi) = 4\omega_h^2 \frac{K_F}{m}(-\cos(0)\sin(\psi))(\phi) = 4\omega_h^2 \frac{K_F}{m}\sin(\psi)(\phi)$$

$$\frac{\partial \ddot{x}(\omega_h, \psi, 0)}{\partial \theta}(\theta) = 4\omega_h^2 \frac{K_F}{m}\cos(\psi)\cos(0)\cos(0)\,(\theta) = 4\omega_h^2 \frac{K_F}{m}\cos(\psi)\,(\theta)$$

$$\frac{\partial \ddot{x}(\omega_h, \psi_0, 0)}{\partial \psi}(\psi - \psi) = 0$$

Combining the terms leads to the linearized equation for $\ddot{x}$.

$$\ddot{x}(\omega, \phi, \theta, \psi) \approx 4\omega_h^2 \frac{K_F}{m}[\phi\sin(\psi) + \theta\cos(\psi)]$$

Now the equation for $\ddot{y}$ is developed.

$$\ddot{y} = \frac{K_F}{m}\Big((\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2)\big(s_\psi s_\theta c_\phi - c_\psi s_\phi\big)\Big)$$

$$\ddot{y}(\omega_h, \psi, 0) = 4\omega_h^2 \frac{K_F}{m}(\sin(\psi)\sin(0)\cos(0) - \cos(\psi)\sin(0)) = 0$$

$$\sum_i \frac{\partial \ddot{y}(\omega_h, \psi, 0)}{\partial \omega_i}(\omega_i - \omega_h) = 0$$

$$\frac{\partial \ddot{y}(\omega_h, \psi, 0)}{\partial \phi}(\phi) = -4\omega_h^2 \frac{K_F}{m}(\cos(\psi)\cos(0))(\phi) = [-4C\omega_h^2 \cos(\psi)]\phi$$

$$\frac{\partial \ddot{y}(\omega_h, \psi, 0)}{\partial \theta}(\theta) = 4\omega_h^2 \frac{K_F}{m}(\sin(\psi)\cos(0)\cos(0))(\theta) = [4C\omega_h^2 \sin(\psi)]\theta$$

$$\frac{\partial \ddot{y}(\omega_h, \psi, 0)}{\partial \psi}(\psi - \psi) = 0$$

Combining the terms leads to the linearized equation for $\ddot{y}$.

$$\ddot{y}(\omega, \phi, \theta, \psi) \approx 4\omega_h^2 \frac{K_F}{m}[\theta \sin(\psi_0) - \phi \cos(\psi_0)]$$

Finally the equation for $\ddot{z}$ is developed.

$$\ddot{z} = \frac{K_F}{m}\left((\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2)c_\phi c_\theta\right) - g$$

$$\ddot{z}(\omega_h, \psi, 0) = 4\omega_h^2 \frac{K_F}{m} - g$$

$$\sum_i \frac{\partial \ddot{z}(\omega_h, \psi, 0)}{\partial \omega_i}(\omega_i - \omega_h) = 2\omega_h \frac{K_F}{m}\sum_i (\omega_i - \omega_h) = 2\omega_h \frac{K_F}{m}\left[\sum_i (\omega_i) - 4\omega_h\right]$$

$$\frac{\partial \ddot{z}(\omega_h, \psi, 0)}{\partial \phi}(\phi) = 0$$

$$\frac{\partial \ddot{z}(\omega_h, \psi, 0)}{\partial \theta}(\theta) = 0$$

$$\frac{\partial \ddot{z}(\omega_h, \psi, 0)}{\partial \psi}(\psi - \psi) = 0$$

Combining the terms leads to the linearized equation for $\ddot{z}$.

$$\ddot{z}(\omega, \phi, \theta, \psi) \approx 4\omega_h^2 \frac{K_F}{m} - g + 2\omega_h \frac{K_F}{m}\left[\sum_i (\omega_i) - 4\omega_h\right]$$

Now we consider the following two relationships.

$$\omega_h^2 = \frac{mg}{4K_F}$$

$$4\omega_h + \Delta\omega_z = \omega_1 + \omega_2 + \omega_3 + \omega_4 = \sum_i \omega_i$$

Substituting these relationships into our linearized equations gives us the following.

$$\ddot{x}(\omega, \phi, \theta, \psi) \approx g[\phi\sin(\psi) + \theta\cos(\psi)]$$

$$\ddot{y}(\omega, \phi, \theta, \psi) \approx g[\theta \sin(\psi_0) - \phi \cos(\psi_0)]$$

$$\ddot{z}(\omega, \phi, \theta, \psi) \approx 2\omega_h \frac{K_F}{m} \Delta\omega_z$$

Our hybrid linearized state space model is represented as

$$\dot{\boldsymbol{v}} = \begin{bmatrix} gsin(\psi) & gcos(\psi) & 0 \\ -gcos(\psi) & gsin(\psi) & 0 \\ 0 & 0 & 2\omega_h \frac{K_F}{m} \end{bmatrix} \begin{bmatrix} \phi \\ \theta \\ \Delta\omega_z \end{bmatrix}$$

Euler Rates

Now a linear model will be developed for the Euler angle's rate of change. Equation (2.10) is copied here for convenience.

$$\dot{\boldsymbol{\Theta}} = \begin{bmatrix} 1 & \frac{s_\phi s_\theta}{c_\theta} & \frac{c_\phi s_\theta}{c_\theta} \\ 0 & c_\phi & -s_\phi \\ 0 & \frac{s_\phi}{c_\theta} & \frac{c_\phi}{c_\theta} \end{bmatrix} \boldsymbol{\Omega}$$

We will now develop an approximate linear model by applying the truncated Taylor series expansion to the individual elements of $\dot{\boldsymbol{\Theta}}, \{\dot{\phi}, \dot{\theta}, \dot{\psi}\}$. We start with $\dot{\phi}$.

$$\dot{\phi}(\theta, p, r) = p + s_\phi t_\theta q + c_\phi t_\theta r$$

$$\dot{\phi}(0) = 0 + \sin(0)\tan(0)(0) + \cos(0)\tan(0)(0) = 0$$

$$\frac{\partial \dot{\phi}(0)}{\partial p}(p) = (1)(p) = p$$

$$\frac{\partial \dot{\phi}(0)}{\partial q}(q) = \sin(0)\tan(0)\, q = 0$$

$$\frac{\partial \dot{\phi}(0)}{\partial r}(r) = \cos(0)\tan(0)(r) = 0$$

Combining the terms leads to the linearized equation for $\dot{\phi}$.

$$\dot{\phi} \approx p$$

Now the equation for $\dot{\theta}$ is developed.

$$\dot{\theta}(\theta, \phi, p, q, r) = c_\phi q - s_\phi r$$

$$\dot{\theta}(0) = \cos(0)(0) + \sin(0)(0) = 0$$

$$\frac{\partial \dot{\theta}(0)}{\partial \theta}(\theta) = (0)\theta = 0$$

$$\frac{\partial \dot{\theta}(0)}{\partial \phi}(\phi) = [-\sin(0)\,(0) - \cos(0)\,(0)]\phi = 0$$

$$\frac{\partial \dot{\theta}(0)}{\partial p}(p) = (0)p = 0$$

$$\frac{\partial \dot{\theta}(0)}{\partial q}(q) = \cos(0)q = q$$

$$\frac{\partial \dot{\theta}(0)}{\partial r}(r) = -\sin(0)r = 0$$

Combining the terms leads to the linearized equation for $\dot{\theta}$.

$$\dot{\theta} \approx q$$

Finally the equation for $\dot{\psi}$ is developed.

$$\dot{\psi}(\theta, \phi, p, r) = \frac{s_\phi}{c_\theta} q + \frac{c_\phi}{c_\theta} r$$

$$\dot{\psi}(0) = \tan(0)\,(0) + \frac{\cos(0)}{\cos(0)}(0) = 0$$

$$\frac{\partial \dot{\psi}(0)}{\partial \theta}(\theta) = \left[\frac{1}{\cos(0)^2}\sin(0)(0) + \frac{1}{\cos(0)^2}\cos(0)\,(0)\right] r = 0$$

$$\frac{\partial\dot{\psi}(0)}{\partial\phi}(\phi) = \left[\frac{\cos(0)}{\cos(0)}(0) - \frac{\sin(0)}{\cos(0)}(0)\right]\phi = 0$$

$$\frac{\partial\dot{\psi}(0)}{\partial p}(p) = (0)p = 0$$

$$\frac{\partial\dot{\psi}(0)}{\partial q}(q) = \left[\frac{\sin(0)}{\cos(0)}\right]q = 0$$

$$\frac{\partial\dot{\psi}(0)}{\partial r}(r) = \left[\frac{\cos(0)}{\cos(0)}\right]r = r$$

Combining the terms leads to the linearized equation for $\dot{\psi}$.

$$\dot{\psi} \approx r$$

Putting the terms into vector form we have the following.

$$\dot{\boldsymbol{\Theta}} \approx \boldsymbol{\Omega}$$

Summarized Modified State Model

Putting all the sections of this appendix together we get a semi-linear state model summarized here.

$$\dot{\boldsymbol{r}} = \boldsymbol{v}$$

$$\dot{\boldsymbol{v}} = \begin{bmatrix} gsin(\psi) & gcos(\psi) & 0 \\ -gcos(\psi) & gsin(\psi) & 0 \\ 0 & 0 & 2\omega_h\dfrac{K_F}{m} \end{bmatrix}\begin{bmatrix} \phi \\ \theta \\ \Delta\omega_z \end{bmatrix}$$

$$\dot{\boldsymbol{\Omega}}_{\boldsymbol{ref}} \approx \begin{bmatrix} \dfrac{2\omega_h LK_F}{\sqrt{2}J_{xx}} & 0 & 0 \\ 0 & \dfrac{2\omega_h LK_F}{\sqrt{2}J_{yy}} & 0 \\ 0 & 0 & \dfrac{2\omega_h LK_M}{J_{zz}} \end{bmatrix}\begin{bmatrix} \Delta\omega_\phi \\ \Delta\omega_\theta \\ \Delta\omega_\psi \end{bmatrix}$$

$$\dot{\boldsymbol{\Theta}} \approx \boldsymbol{\Omega}$$

APPENDIX C

Python Code for Hardware Implementation

iRobot Detection with Hough Transform

```
# Authors:          Patrick Friudenberg
(patrick_friudenberg@baylor.edu)
# Organization:     Baylor University
# Project:          IARC Avionics Research
# Description:      Functions to detect iRobot as circles and return
centers.
#                      Tested 3DR Video/OSD System Kit
# Date Created:     10/13/14

import numpy as np
import cv2
import cv2.cv as cv
import sys as sys

cap = cv2.VideoCapture(0)

while(True):
    # Capture frame-by-frame
    ret, img = cap.read()
    if img == None:
        cap.release()
        cv2.destroyAllWindows()
        sys.exit('Error: Unable to connect video')

    # Convert BGR to Gray Scale
    imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    #
cv2.HoughCircles(grayscaleImg,DetectionMethod,dp=1,MinDistBetweenCenter
s,CannyUpperThresh,CenterDetectThresh,MinRad,MaxRad)
    cannyUpperThresh = 50
    centerThresh = 15
    minRadius = 85
    maxRadius = 87
    minCenterDist = 2*minRadius
    circles =
cv2.HoughCircles(imgGray,cv.CV_HOUGH_GRADIENT,1,minCenterDist,param1=ca
nnyUpperThresh,param2=centerThresh,minRadius=minRadius,maxRadius=maxRad
ius)

    # Draw circles on Image
    if circles != None:
        for i in circles[0,:]:
            cv2.circle(img,(i[0],i[1]),i[2],(255,255,255),2)
            print circles
```

```
    cv2.imshow('Camera View',img)

    k = cv2.waitKey(1) & 0xFF
    if k == 27:
        break
cap.release()
cv2.destroyAllWindows()
```

Multiple iRobot Detection and Memory Block

```
# Authors:          Patrick Friudenberg
(patrick_friudenberg@baylor.edu)
#                   Andres Bizzarri (andres_bizzarri@baylor.edu)
# Organization:     Baylor University
# Project:          IARC Avionics Research
# Description:      Functions to detect multiple iRobots and place in
Memory Block.
#                   Kalman filter applied to predict states of failed
detection
# Date Created:     11/23/14

import numpy as np
import cv2
import cv2.cv as cv
import sys as sys
import math as m

class Mem:
    """
    A class for a memory block with applicable methods
    @author: Patrick Friudenberg // Andres Bizzarri
    """
    def __init__(self,size,minCenterDist):
        self.state = 0;
        #---- Main Memory Block
        self.memBlock = np.zeros([size,3])
        self.threshold = (minCenterDist)*(1.1)
        #--- Kalman Filter Blocks
        self.kalmen = []
        self.kalmen_state = []
        self.kalmen_process_noise = []
        self.kalmen_measurement = []
        self.kalmen_state_gain = [1,2]

    def memoryCorr(self,circ):
        """
        correlates given circle with memory, returns its index
        using min distance between circles to see if circle we're
putting into memory is a new circle or the old one
        using minimum distance of new circle vs all mem circles to see
which one it most likely is
        @author: Andres Bizzarri
        """
```

```
        rowcount = 0
        index = 0
        u = self.memBlock[0,0:3]
        b = circ - u
        mindist = m.sqrt(pow(b[0],2)+pow(b[1],2)+pow(b[2],2))
        if self.state == 0:    #If memory is empty, Place new circle in
first index
            self.memBlock[0,:] = circ
            self.state = 1
            return 0
        elif self.state == 1:   #If 0 < memory < full, correlate energy
of memory and cirlce
                    #------ Find index of closest entity in memory to
circle -----
            rowcount = 0    # Initialize temporary variables
            index = 0
            u = self.memBlock[0,:]
            b = circ - u
            mindist = m.sqrt(pow(b[0],2)+pow(b[1],2))
            for k in self.memBlock:       # Find index of closest
circle in memory
                if k[2] == 0:
                    break
                else:
                    v = [(circ[0] - k[0]),(circ[1] - k[1]),(circ[2] - k
[2])]
                    measdiff = m.sqrt(pow(v[0],2)+pow(v[1],2))
                if mindist >= measdiff:
                    mindist = measdiff
                    index = rowcount
                rowcount += 1
            if mindist > self.threshold: #Consider Circles outside
threshold as new circles
                index = rowcount
                self.memBlock[index,:] = circ
            return index;
    def kalmanManager(self,circ,index):
        """
        Class to manage the self.kalmen array of cv.kalman objects
including
        creating new kalman in the array; and correcting and predicting
with kalman
        objects already in the array. The kalman array is managed vis
the index
        of the corresponding element in self.memBlock, and correction
is made
        with the circ value.
        @author: Andres Bizzarri // Patrick Friudenberg
        """

        if len(self.kalmen) < index+1:  # Initiate new Kalman object
for new self.memBlock entity
            self.kalmen.append(cv.CreateKalman(4, 2, 0))
            self.kalmen_state.append(cv.CreateMat(2, 1, cv.CV_32FC1))
            self.kalmen_process_noise.append(cv.CreateMat(4, 1,
cv.CV_32FC1))
```

```
            self.kalmen_measurement.append(cv.CreateMat(2, 1,
cv.CV_32FC1))
            self.kalmen[index].transition_matrix[0,0] =
self.kalmen_state_gain[0]#1
            self.kalmen[index].transition_matrix[0,1] = 0#0
            self.kalmen[index].transition_matrix[0,2] =
self.kalmen_state_gain[1]#2
            self.kalmen[index].transition_matrix[0,3] = 0#0
            self.kalmen[index].transition_matrix[1,0] = 0#0
            self.kalmen[index].transition_matrix[1,1] =
self.kalmen_state_gain[0]#1
            self.kalmen[index].transition_matrix[1,2] = 0#0
            self.kalmen[index].transition_matrix[1,3] =
self.kalmen_state_gain[1]#2
            self.kalmen[index].transition_matrix[2,0] = 0#0
            self.kalmen[index].transition_matrix[2,1] = 0#0
            self.kalmen[index].transition_matrix[2,2] =
self.kalmen_state_gain[0]#1
            self.kalmen[index].transition_matrix[2,3] = 0#0
            self.kalmen[index].transition_matrix[3,0] = 0#0
            self.kalmen[index].transition_matrix[3,1] = 0#0
            self.kalmen[index].transition_matrix[3,2] = 0#0
            self.kalmen[index].transition_matrix[3,3] =
self.kalmen_state_gain[0]#1
             # Set Filter
            cv.SetIdentity(self.kalmen[index].measurement_matrix,
cv.RealScalar(1))
            cv.SetIdentity(self.kalmen[index].process_noise_cov,
cv.RealScalar(1e-5))
            cv.SetIdentity(self.kalmen[index].measurement_noise_cov,
cv.RealScalar(1e-1))
            cv.SetIdentity(self.kalmen[index].error_cov_post,
cv.RealScalar(.1))
            kalman_estimated = cv.KalmanPredict(self.kalmen[index])
        else:
            self.kalmen[index].state_pre[0,0] = self.memBlock[index,0]
            self.kalmen[index].state_pre[1,0] = self.memBlock[index,1]

            # predict new point
            self.kalman_prediction =
cv.KalmanPredict(self.kalmen[index])

            #correction
            if circ != None:
                self.kalmen_measurement[index][0, 0] = circ[0]
                self.kalmen_measurement[index][1, 0] = circ[1]
                kalman_estimated = cv.KalmanCorrect(self.kalmen[index],
self.kalmen_measurement[index])
            elif circ == None:
                kalman_estimated = cv.KalmanPredict(self.kalmen[index])
        #Return predicted center of circle
        x = kalman_estimated[0,0]
        y = kalman_estimated[1,0]
        return(x,y,0)

    def getUndetectedIndexes(self,detectedIndexes):
        undetectedIndexes = range(len(self.memBlock))
```

```
        count = 0
        for i in detectedIndexes:
            undetectedIndexes.__delitem__(i-count)
        return undetectedIndexes

##------------------------ Main ---------------------------------------
---

#--- Hough variables
cannyUpperThresh = 50         # Edge detection Threshold
centerThresh = 18             # Circle detection Threshold
minRadius = 85                # Radius Threshold
maxRadius = 87
minCenterDist = 2*minRadius   # Distance between circles Threshold

iCreates = 10                   #number of iCreates
mem = Mem(iCreates,minCenterDist)                  #init memory

cap = cv2.VideoCapture(1)      # Opening video stream
if cap.isOpened() != True:
    cap.open(0)

while(True):
    # Capture frame-by-frame
    ret, img = cap.read()
    if img == None:
        cap.release()
        cv2.destroyAllWindows()
        sys.exit('Error: Unable to connect video')

    # Convert BGR to Gray Scale
    imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    # Extract circle center and radius [x y r]
    circles =
cv2.HoughCircles(imgGray,cv.CV_HOUGH_GRADIENT,1,minCenterDist,param1=ca
nnyUpperThresh,param2=centerThresh,minRadius=minRadius,maxRadius=maxRad
ius)
    detectedIndexes = []
    if circles != None:
        for circ in circles[0,:]:
            index = mem.memoryCorr(circ) # Find Correlating entity in
mem.memBlock
            detectedIndexes.append(index)
            xy=mem.kalmanManager(circ,index) #Process kalman from
mem.kalman[index] object
            cv2.circle(img,(circ[0],circ[1]),circ[2],(255,255,255),4)
            cv2.circle(img,(int(xy[0]),int(xy[1])),circ[2],(0,0,0),2)

    for f in mem.getUndetectedIndexes(detectedIndexes): # Make
Prediction, but no Correction
        xy = mem.kalmanManager(None,f)
        cv2.circle(img,(int(xy[0]),int(xy[1])),minRadius,(0,0,0),2)

    cv2.imshow('Camera View',img)
    k = cv2.waitKey(1) & 0xFF
    if k == 27:
        break
```

```
cap.release()
cv2.destroyAllWindows()
```

Fetch Parameters from UAV

```
# Authors:          Samuel Taylor (samuel_taylor@baylor.edu)
#                   Matt Tinsley (matt_tinsley@baylor.edu)
# Organization:     Baylor University
# Project:          IARC Avionics Research
# Description:      Functions to fetch parameters from arducopter
drone.
#                      Tested using Pixhawk and arducopter v3.1.5
# Date Created:     1/23/15
# Date Modified:    1/26/15 - Added get_all_params fuction

import sys, os, time
from math import radians
from pymavlink import mavutil

# function to fetch specified parameters from arducopter
# parameters: mavonn - mavlink_connection
# params: list of parameters to fetch in the form of ['PARAM_1',
PARAM_2,...]
def get_specified_params(mavconn, params):
    '''given a mavlink_connection, gets the parameters with the names
    specified in params'''
    received_params = []

    # send all the "get parameter" requests
    for param in params:
        mavconn.param_fetch_one(param)

    # try to receive a number of parameters equal to the length of
params.
    # note that because we pass a timeout to recv_match, if a parameter
    # isn't found, None will be returned
    while len(received_params) < len(params):
        msg = mav1.recv_match(type='PARAM_VALUE', blocking=True,
timeout=1)
        received_params.append(msg)

    return received_params

# function to fetch all parameters from arducopter
# parameters: mavonn - mavlink_connection
def get_all_params(mavconn):
    '''given a mavlink_connection, gets the parameters with the names
    specified in params'''
    received_params = []

    # send the "get all parameters" request
    mavconn.param_fetch_all()

    # wait until the mavfile object (mav1) is done fetching params
    # note that because we pass a timeout to recv_match, if a parameter
```

```
    # isn't found, None will be returned
    while not mav1.param_fetch_complete:
        msg = mav1.recv_match(type='PARAM_VALUE', blocking=True,
timeout=1)
        received_params.append(msg)

    return received_params


# set port of 3DR antenna - COM3 or COM4 in our tests so far
Port = 'COM4'
baudrate = 57600
rc1_channel = 0
rc2_channel = 1
rc3_channel = 2
rc4_channel = 3
rc5_channel = 4
rc6_channel = 5
rc7_channel = 6
rc8_channel = 7

values = [1165]*8

# establish mavlink connection on proper port and baudrate
mav1 = mavutil.mavlink_connection(Port,baud=baudrate)
print("Waiting for HEARTBEAT")

# receive heartbeat to begin interaction with the system
mav1.wait_heartbeat()
print("Heartbeat from APM (system %u component %u)" %
(mav1.target_system, mav1.target_system))


# assume mav1 is a properly-initialized mavlink_connection

# uncomment the following line to fetch specific parameters
# params = get_specified_params(mav1, ['THR_MIN', 'THR_MAX', 'RC3_MIN',
'RC3_MAX'])

# uncomment the following line to fetch all parameters
params = get_all_params(mav1)

# print out fetched parameters' param_id and param_value
for p in params:
    # don't print None params - ones we requested that didn't exist
    if not p:
        continue
    print str(p.param_id), '\t', p.param_value

# close mavlink connection
mav1.close()
```

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov 23 17:30:35 2014
Main File for offboard control of X8 quadrotor
@author: Patrick Friudenberg (patrick_friudenberg@baylor.edu)

"""

import sys, os, time
from math import radians
from pymavlink import mavutil

Port = 'COM3'
baudrate = 57600
rc1_channel = 0
rc2_channel = 1
rc3_channel = 2
rc4_channel = 3
rc5_channel = 4
rc6_channel = 5
rc7_channel = 6
rc8_channel = 7
values = [1165]*8
mav1 = mavutil.mavlink_connection(Port,baud=baudrate)
print("Waiting for HEARTBEAT")
mav1.wait_heartbeat()
print("Heartbeat from APM (system %u component %u)" %
(mav1.target_system, mav1.target_system))
time.sleep(5)
mav1.arducopter_arm()
print('The system should now be armed')
time.sleep(10)
values[rc3_channel] = 1600
mav1.mav.rc_channels_override_send(mav1.target_system,
mav1.target_component, *values)
print('WOAH!!! the motors are humming pretty good!')
time.sleep(10)
values[rc3_channel] = 1210
mav1.mav.rc_channels_override_send(mav1.target_system,
mav1.target_component, *values)
print('And now they are at Throttle suitable for orientation tests')
time.sleep(10)
values[rc1_channel] = 1750
mav1.mav.rc_channels_override_send(mav1.target_system,
mav1.target_component, *values)
print('Now the motors should be doing full roll')
time.sleep(10)
values[rc1_channel] = 1500
mav1.mav.rc_channels_override_send(mav1.target_system,
mav1.target_component, *values)
print('Now running evenly again')
time.sleep(10)
values[rc1_channel] = 1250
```

```
mav1.mav.rc_channels_override_send(mav1.target_system,
mav1.target_component, *values)
print('Now the motors should be doing full reverese roll')
time.sleep(10)
values[rc1_channel] = 1500
mav1.mav.rc_channels_override_send(mav1.target_system,
mav1.target_component, *values)
print('Now running evenly again')
time.sleep(10)
values[rc2_channel] = 1750
mav1.mav.rc_channels_override_send(mav1.target_system,
mav1.target_component, *values)
print('Now the motors should be doing full pitch')
time.sleep(10)
values[rc2_channel] = 1500
mav1.mav.rc_channels_override_send(mav1.target_system,
mav1.target_component, *values)
print('Now running evenly again')
time.sleep(10)
values[rc2_channel] = 1250
mav1.mav.rc_channels_override_send(mav1.target_system,
mav1.target_component, *values)
print('Now the motors should be doing full reverse pitch')
time.sleep(10)
values[rc2_channel] = 1500
mav1.mav.rc_channels_override_send(mav1.target_system,
mav1.target_component, *values)
print('Now running evenly again')
time.sleep(10)
values[rc4_channel] = 1750
mav1.mav.rc_channels_override_send(mav1.target_system,
mav1.target_component, *values)
print('Now the motors should be doing full yaw')
time.sleep(10)
values[rc4_channel] = 1500
mav1.mav.rc_channels_override_send(mav1.target_system,
mav1.target_component, *values)
print('Now running evenly again')
time.sleep(10)
values[rc4_channel] = 1250
mav1.mav.rc_channels_override_send(mav1.target_system,
mav1.target_component, *values)
print('Now the motors should be doing full reverse yaw')
time.sleep(10)
values[rc4_channel] = 1500
mav1.mav.rc_channels_override_send(mav1.target_system,
mav1.target_component, *values)
print('Now running evenly again')
time.sleep(10)
values[rc3_channel] = 1165
mav1.mav.rc_channels_override_send(mav1.target_system,
mav1.target_component, *values)
print('And now they are at minimum Throttle again')
time.sleep(10)
mav1.arducopter_disarm()
print('The system should now be disarmed')
mav1.close()
```

APPENDIX D

List of all Links to Project Resources

The hardware implementation portion of this thesis required data sheets, software, schematic diagrams, and tutorials. Below is a list of links to most of the resources used. In the General Resources list, the Cheetah Drone was built with computer vision capability, along with localization and target tracking. This is very similar to our project. Also, the Pixhawk autopilot itself was built originally for this purpose. For general computer vision techniques, TUM Computer Vision is a good resource. The next section covers all data sheets, firmware, websites, and manuals for the required hardware. After that all required software from computer vision to control is listed with many items having links to actual source code.

General Resources

Similar Projects and Papers

1. Cheetah Drone
   1.1. https://pixhawk.ethz.ch/micro_air_vehicle/quadrotor/cheetah

2. TUM Computer Vision Group
   2.1. http://vision.in.tum.de/publications

3. Optic Flow Paper
   3.1. http://www.sciencedirect.com.ezproxy.baylor.edu/science/article/pii/0042698986900787

4. PIxhawk Group Publications
   4.1. https://pixhawk.ethz.ch/overview/publications

Hardware Links

Pixhawk Autopilot

1. Specifications, schematics
   1.1. http://pixhawk.org/modules/pixhawk

2. User Manual:
   2.1. http://www.3drobotics.com/wp-content/uploads/2014/03/pixhawk-manual-rev7.pdf

3. Product Page
   3.1. https://store.3drobotics.com/products/3dr-pixhawk
   3.2. http://3drobotics.com/learn/pixhawk-autopilot-system/

4. Software
   4.1. https://pixhawk.ethz.ch/software/start
   4.2. https://pixhawk.ethz.ch/installation/start

5. Sounds & LED's Explained
   5.1. http://copter.ardupilot.com/wiki/common-apm-board-leds/#PixhawkPX4_Sounds

GPS Mast

1. Product Page
   1.1. http://store.3drobotics.com/products/3dr-gps-ublox-with-compass

2. Setup
   2.1. http://www.3drobotics.com/wp-content/uploads/2014/02/GPS-Mast-Doc-V1.pdf

3. User manual
   3.1. http://www.3drobotics.com/wp-content/uploads/2013/08/3DR-uBlox-GPS-web-version.pdf

3DR Video/OSD System Kit

1. Product Page
   1.1. https://store.3drobotics.com/products/3dr-fpv-osd-kit

2. User Manual:
   2.1. http://3drobotics.com/wp-content/uploads/2014/05/FPVOSD-Kit-Manual-A.pdf

3. Sony HAD 520 line camera
   3.1. http://store.3drobotics.com/products/super-had-ccd-camera-1-3-sony-520tv-lines

4. On-Screen Display Board

4.1. http://store.3drobotics.com/products/apm-minimosd-rev

3DR 915 Mhz Telemetry (X8 / GCS communication)

1. Product Page
   1.1. http://store.3drobotics.com/products/3dr-radio
   1.2. http://copter.ardupilot.com/wiki/common-using-the-3dr-radio-for-telemetry-with-apm-and-px4/

2. FirmWare
   2.1. https://github.com/tridge/SiK

3. User Manual
   3.1. http://www.3drobotics.com/wp-content/uploads/2013/10/3DR-Radio-V2-doc1.pdf

4. Developer:
   4.1. https://code.google.com/p/ardupilot-mega/wiki/3DRadio

Spektrum DX7s (Radio)

1. User Manuals and Software
   1.1. https://www.spektrumrc.com/Products/Default.aspx?ProdId=SPM7800
      1.1.1. Choose Manuals and Software Tab

Software Links

PX4 Toolchain

1. Installation
   4.2. http://pixhawk.org/dev/toolchain_installation_win

2. USB driver
   2.1. http://pixhawk.org/firmware/downloads

Ground Control Stations

1. QGround Control
   1.1. Install
      1.1.1. http://qgroundcontrol.org/downloads
   1.2. Source Code
      1.2.1. http://www.qgroundcontrol.org/dev/build_source#ii_clone_the_repository

2. Mission Planner

2.1. Install
    2.1.1. http://firmware.diydrones.com/Tools/MissionPlanner/

MAVProxy / MAVLink

1. MAVLink Documentation
    1.1. http://www.samba.org/tridge/UAV/pymavlink/apidocs/classIndex.html#mavlink.MAVLink

2. MAVProxy START
    2.1. http://dev.ardupilot.com/wiki/mavproxy-on-windows-7/

OpenCV

1. Python Install
    1.1. http://docs.opencv.org/trunk/doc/py_tutorials/py_setup/py_setup_in_windows/py_setup_in_windows.html#install-opencv-python-in-windows
        1.1.1. Follow Installing OpenCV from pre-built libraries

2. C++ Install
    2.1. https://www.youtube.com/watch?v=POpMQPM9YlY

3. Documentation
    3.1. http://docs.opencv.org/

4. Tutorials
    4.1. http://docs.opencv.org/trunk/doc/py_tutorials/py_tutorials.html

REFERENCES

[1] *Unmanned Aircraft Operations in the National Airspace System,* FAA-2006-25714, July 9, 2015.

[2] "Official Rules for the International Aerial Robotics Competition," http://www.aerialroboticscompetition.org/rules.php.

[3] M. Siegel, "The sense-think-act paradigm revisited." p. 5 pp.

[4] D. J. Hall, *Robotic Sensing Devices*, Carnegie-Mellon University, 1984.

[5] M. Siegel, "Smart sensors and small robots." pp. 303-308 vol.1.

[6] R. Goodrich, "Accelerometer vs. Gyroscope: What's the Difference?," *Live Science*, October 1, 2013.

[7] D. H. T. a. J. L. Weston, *Strapdown Inertial Navigation Technology*, 2 ed., Reston, VA: The American Institute of Aeronautics, 2004.

[8] N. O. a. A. Administration. "What is Sonar? Sonar uses sound waves to 'see' in the water," http://oceanservice.noaa.gov/facts/sonar.html.

[9] J. L. S. a. W. H. Chung, *Stochastic Processes, Estimation , and Control*: Society for Industrial and Applied Mathematics, 2008.

[10] K. Mincheol, "Team-Soar: a computational model for multilevel decision making," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on,* vol. 31, no. 6, pp. 708-714, 2001.

[11] J. J. Roldan, J. del Cerro, and A. Barrientos, "A proposal of methodology for multi-UAV mission modeling." pp. 1-7.

[12] D. Pascarella, S. Venticinque, and R. Aversa, "Agent-Based Design for UAV Mission Planning." pp. 76-83.

[13] A. J. Pohl, and G. B. Lamont, "Multi-objective UAV mission planning using evolutionary computation." pp. 1268-1279.

[14] S. Leary, M. Deittert, and J. Bookless, "Constrained UAV mission planning: A comparison of approaches." pp. 2002-2009.

[15] Z. Shaolei, K. Yuhang, and S. Xianjun, "Indoor localization, navigation and mapping for quad-rotor." pp. 2669-2674.

[16] J. Seungho, and J. Seul, "Vision-based localization of a quad-rotor system." pp. 636-638.

[17] C. Rizzo, F. Lera, and J. L. Villarroel, "A Methodology for Localization in Tunnels Based on Periodic RF Signal Fadings." pp. 317-324.

[18] H. A. F. Almurib, P. T. Nathan, and T. N. Kumar, "Control and path planning of quadrotor aerial vehicles for search and rescue." pp. 700-705.

[19] S. R. Barros dos Santos, S. N. Givigi, and C. L. Nascimento, "Autonomous construction of structures in a dynamic environment using Reinforcement Learning." pp. 452-459.

[20] B. C. Min, E. J. Lee, S. H. Kang, and D. H. Kim, "Limit-cycle navigation method for a quad-rotor type UAV." pp. 1352-1357.

[21] H. Qiong, F. Qing, W. Qinghe, and G. Qingbo, "Research and application of nonlinear control techniques for quad rotor UAV." pp. 706-710.

[22] M. H. Tanveer, D. Hazry, S. F. Ahmed, M. K. Joyo, F. A. Warsi, H. Kamaruddin, Z. M. Razlan, K. Wan, and A. B. Shahriman, "NMPC-PID based control structure design for avoiding uncertainties in attitude and altitude tracking control of quad-rotor (UAV)." pp. 117-122.

[23] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, “The GRASP Multiple Micro-UAV Testbed,” *Robotics & Automation Magazine, IEEE,* vol. 17, no. 3, pp. 56-65, 2010.

[24] H. Sira-Ramirez, "On the linear control of the quad-rotor system." pp. 3178-3183.

[25] S. Khatoon, D. Gupta, and L. K. Das, "PID & LQR control for a quadrotor: Modeling and simulation." pp. 796-802.

[26] A. Serirojanakul, and M. Wongsaisuwan, "Optimal control of quad-rotor helicopter using state feedback LPV method." pp. 1-4.

[27] E. Pfeifer, and F. Kassab, "Dynamic Feedback Controller of an Unmanned Aerial Vehicle." pp. 261-266.

[28] I. Gonzalez, S. Salazar, R. Lozano, and J. Escareno, "Real-time altitude robust controller for a Quad-rotor aircraft using Sliding-mode control technique." pp. 650-659.

[29] B. Sumantri, N. Uchiyama, and S. Sano, "Second order sliding mode control for a quad-rotor helicopter with a nonlinear sliding surface." pp. 742-746.

[30] B. Sumantri, N. Uchiyama, and S. Sano, "Least square based sliding mode control for a quad-rotor helicopter." pp. 324-328.

[31] P. Cheng, B. Yue, G. Xun, G. Qingjia, Z. Changjun, and T. Yantao, "Modeling and robust backstepping sliding mode control with Adaptive RBFNN for a novel coaxial eight-rotor UAV," *Automatica Sinica, IEEE/CAA Journal of,* vol. 2, no. 1, pp. 56-64, 2015.

[32] R. Abbas, and W. Qinghe, "Formation Tracking for Multiple Quadrotor Based on Sliding Mode and Fixed Communication Topology." pp. 233-238.

[33] D. Robotics. "X8+," August 25, 2014; https://store.3drobotics.com/products/x8-plus.

[34] "OpenCV-Python Tutorials," September 12, 2104; http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html.

[35] D. Robotics. "Camera with 1/3" Sony Super HAD CCD," November 20, 2014; https://store.3drobotics.com/products/super-had-ccd-camera-1-3-sony-520tv-lines.

[36] L. Seunghan, and B. Hyochoong, "Guidance laws for target localization using vector field approach," *Aerospace and Electronic Systems, IEEE Transactions on,* vol. 50, no. 3, pp. 1991-2003, 2014.

[37] D. R. Nelson, D. B. Barber, T. W. McLain, and R. W. Beard, "Vector field path following for small unmanned air vehicles." p. 7 pp.

[38] D. R. Nelson, D. B. Barber, T. W. McLain, and R. W. Beard, "Vector Field Path Following for Miniature Air Vehicles," *Robotics, IEEE Transactions on,* vol. 23, no. 3, pp. 519-529, 2007.

[39] M. Fraiwan, A. Alsaleem, H. Abandeh, and O. Aljarrah, "Obstacle avoidance and navigation in robotic systems: A land and aerial robots study." pp. 1-5.

[40] C. Ji-wung, "A potential field and bug compound navigation algorithm for nonholonomic wheeled robots." pp. 166-171.

[41] L. Zhen-yu, J. Rui-huan, D. Xiang-Qian, and L. Jian-hua, "Trajectory Tracking Control of Wheeled Mobile Robots Based on the Artificial Potential Field." pp. 382-387.

[42] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots." pp. 500-505.

[43] L. Guan-Hao, C. Chih-Fu, and F. Li-Chen, "Navigation of a wheeled mobile robot in indoor environment by potential field based-fuzzy logic method." pp. 1-6.

[44] S. A. Murtaugh, and H. E. Criel, "Fundamentals of Proportional Navigation and Its Application to an Antisatellite Interceptor," *Aerospace and Electronic Systems, IEEE Transactions on,* vol. AES-2, no. 4, pp. 725-725, 1966.

[45] M. Mehrandezh, N. M. Sela, R. G. Fenton, and B. Benhabib, "Robotic interception of moving objects using an augmented ideal proportional navigation guidance technique," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on,* vol. 30, no. 3, pp. 238-250, 2000.

[46] F. Belkhouche, and B. Belkhouche, "A control strategy for tracking-interception of moving objects using wheeled mobile robots." pp. 2129-2130 Vol.2.

[47] F. Belkhouche, B. Belkhouche, and P. Rastgoufard, "Line of sight robot navigation toward a moving goal," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on,* vol. 36, no. 2, pp. 255-267, 2006.

[48] C. Jay Hyuk, L. Won-Suk, and B. Hyochoong, "Helicopter guidance for vision-based tracking and landing on a moving ground target." pp. 867-872.

[49] F. Kunwar, F. Wong, R. B. Mrad, and B. Benhabib, "Rendezvous Guidance for the Autonomous Interception of Moving Objects in Cluttered Environments." pp. 3776-3781.

[50] F. Kunwar, F. Wong, R. B. Mrad, and B. Benhabib, "Time-optimal rendezvous with moving objects in dynamic cluttered environments using a guidance based technique." pp. 283-288.

[51] K. Chien-Chun, C. Feng-Lung, and W. Chi-Yu, "Implement three-dimensional pursuit guidance law with feedback linearization control method." pp. 187-190.

[52] G. M. Dimirovski, S. M. Deskovski, and Z. M. Gacovski, "Classical and fuzzy-system guidance laws in homing missiles systems." pp. 3032-3047 Vol.5.

[53] P. Zarchan, *Tactical and Strategic Missile Guidance*, 5 ed.: American Institute of Aeronautics and Astronautics, Inc., 2007.

[54] M. W. S. a. S. H. a. M. Vidyasagar, *Robot Modeling and Control*: John Wiley & Sons, Inc., 2006.

[55] W. F. Phillips, *Mechanics of Flight*, 2nd ed., Hoboken, New Jersey: John Wiley & Sons, Inc., 2010.