ABSTRACT

AUTONOMOUS QUADROCOPTER CONTROL

Evan D. Boldt, M.S.
Department of Mechanical Engineering
Northern Illinois University, 2014
Dr. Brianno Coller, Director

Quadrocopters, or quad rotor helicopters, are typically small and unmanned. New uses are still being developed for these aircraft such as autonomous delivery services or surveillance. The quadrocopter in this thesis is optimized for indoor flight. So, a safer chassis is designed to keep the propellers protected. Also, a linkage for cargo manipulation is designed and optimized using a genetic algorithm. PID controllers are designed, which rely on many different sensors to determine the present position and orientation of the craft such as accelerometers, cameras, and wireless trilateration. A simple use-case is simulated where a quadrocopter can sort blocks based on a learned color pattern using an artificial neural network (ANN). A data storage and search system is developed to quickly search for obstacles to avoid. Finally, a prototype quadrocopter is constructed to implement some of the designed components and features, such as the controller and some sensors. Many of the systems developed, like the linkage solver, wireless trilateration, and spatial data storage system, have significant uses beyond those for a quadrocopter.

NORTHERN ILLINOIS UNIVERSITY
DE KALB, ILLINOIS

MAY 2014

AUTONOMOUS QUADROCOPTER CONTROL

BY

EVAN D. BOLDT

A THESIS SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE

MASTER OF SCIENCE

DEPARTMENT OF MECHANICAL ENGINEERING

Thesis Director:
Dr. Brianno Coller

ACKNOWLEDGEMENTS

In no particular order, I would like to thank:

- Addison Merchut and Jennifer Case for introducing me to the wonderful world of Arduino microcontrollers and unlocking the door for my mechatronics exploration.
- Dr. Brianno Coller for allowing me the freedom to take my work in the directions I saw fit, as well as providing the outstanding teaching that is the foundation of my mechanical engineering knowledge.
- Dr. Peng-Yung Woo for teaching me the very complex problem-solving methods that I was able to apply to my work and for his significant help in revising part of the thesis.
- Dr. Ji-Chul Ryu for being understanding as a professor I work for and for his understanding of control systems.
- Northern Illinois University for providing a great learning environment.
- Mom and Dad for their support in my education.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

A quadrocopter is a flying robot with four propellers. Unlike a conventional helicopter, stabilization of a quadrocopter does not require any reorientation of the propellers, nor does it require a tail rotor for anti-torque. If two of the propellers spin in opposite directions, then the tail rotor is unnecessary because the torques applied by two propellers on the craft can cancel the torque created by the other two. This is why there are few multi-rotor helicopters with an odd number of propellers. To prevent spin about the z-axis (yaw), one of the propellers must be a different size or shape, or be going a different speed. For example, multi-rotor helicopters with three rotors typically are in a Y-shape with one of the propellers being significantly larger than the other two. As a result, the large propeller is farther away from the others to prevent collision between propellers, but the distance does not change the torque exerted.

Quadrocopters are more common on small scales primarily due to the fact that the propellers can be fixed and do not require an intricate linkage assembly to control the pitch of the blades known as a swashplate which is shown in Figure 1.1. Propeller blade manipulations would be mechanically fragile in small scale, add unnecessary complexity to the control system, and increase cost and weight. Still, yaw needs to be controlled.

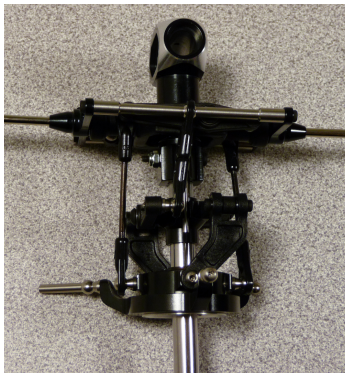


Figure 1.1: Swashplate [1].

Using a pair of propellers spinning in opposite directions cancels the torque applied by spinning but does not allow direct control over other orientations by use of thrust like pitch or roll. For example, the CH-47 Chinook [2, pg. 36,15] uses two propellers - each spinning in opposite directions. Like a typical helicopter, the blades of the top propeller must be twisted in order to allow the helicopter to roll or pitch. However, differences in thrust and torque also change orientations. Differences in thrust will change pitch, but mismatched propeller torques will cause a yaw. The pitch can be controlled by twisting the blades with the swashplate, but yaw must be controlled by the two propeller speeds.

1.1 Basic Dynamics

The three dimensional dynamics of the quadrocopter can be decomposed into the four torques and four forces shown in Figure 1.2. The torque is directly related to the force applied by a propeller, since both are the result of propeller speed and air resistance, and both vectors are parallel. The basic force and moment equations using Newton's Second Law are:

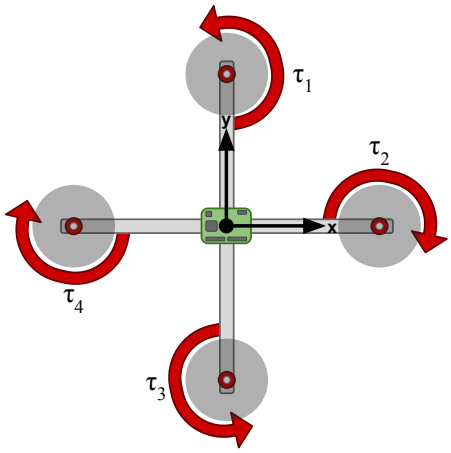


Figure 1.2: Top-down view of FBD.

Force:

$$\sum \mathbf{F} = ma \tag{1.1}$$

$$\sum_{i=1}^{4} \left(\vec{T_i}\right) - mg = m\vec{a} \tag{1.2}$$

Moment:

$$\sum M_G = I_G \alpha \tag{1.3}$$

$$\sum_{i=1}^{4} \left(\vec{T_i} \times \vec{r_i} + \vec{\tau}_{prop,i} \right) = I_G \vec{\alpha} \tag{1.4}$$

Where:

T_i is the force vector for the thrust applied by a propeller.

r_i is the distance vector from the propeller to the center of mass.

a is the acceleration vector.

α is the angular acceleration vector following right-hand rule.

τ_{prop} is the torque applied by the propeller to the craft due to changes in speed or air resistance, which is equal to the torque applied by the motor. Note that two propellers will be spinning in opposite directions.

I_G is a tensor representing the principal moment of inertia of the entire craft.

Thrust and torque applied by propellers are a function of their current angular velocities, air speed, and signal applied to change their angular acceleration.

1.2 Applications for Quadrocopters

Potential uses for quadrocopters are still being explored. They are gaining in popularity due to decreasing costs and increasing flight times and payloads due to better lithium-polymer battery technology. While most are still limited to less than 20 minutes of flight time and several pounds of cargo, they still have potential commercial applications. One of the first popular uses for quadrocopters is for inexpensive aerial photography, since they are substantially cheaper than a typical helicopter and camera equipment is often light enough for a quadrocopter to carry. More recently, quadrocopters are being tested as lightweight and fast-response delivery services. For example, Amazon has revealed that it has been developing a

method to deliver orders in under 30 minutes for orders less than 5 lbs, or 86% of orders [3].

The legal environment in the United States is also a potential limiting factor. Unmanned aerial vehicles may not be operated commercially unless operated by a a government public safety agency and the aircraft weighs under 4.4 lbs, is within line of sight of the operator, during daylight, less than 400 feet above ground, and many other limitations [4]. Use by businesses is restricted unless experimental certification is obtained [5]. The FAA is currently working on expanding standards and certifications for unmanned aerial vehicles, but until then, outdoor commercial applications such as aerial photography or delivery services will not be legal in the United States.

Therefore, a more immediate potential use for quadrocopters is to focus on indoor tasks, since they are not subject to FAA regulations. Potential applications include three dimensional interior mapping or fast transport of lightweight objects.

To accomplish tasks indoors, a manipulator will be designed to allow it to pick up cargo. The design will be done using a novel genetic algorithm optimization method. Then a simulation will be created to test a control system and path planning system. To be used indoors, a new method for positioning is developed to overcome existing limitations like the poor indoor reception of GPS or expensive external sensor. Additionally, a camera based position tracking system is developed. Finally, a prototype is created using a unique design that shields the propellers from collisions.

CHAPTER 2

MANIPULATOR DESIGN

Some quadrocopters are intended to pick up, move, drop, or interact with a nearby object. These are actions performed by a manipulator. For example, the NIU Robotics student organization competed in the 2013 Jerry Sanders Creative Design Competition [6] where several robots were required to pick up plastic cones of a certain color and move them to other points on the course. The quadrocopter they designed would land, contract the mechanical manipulator shown in Figure 2.1 to pick up a cone, take off towards a destination, land, and release a cone with the manipulator on a target pin.

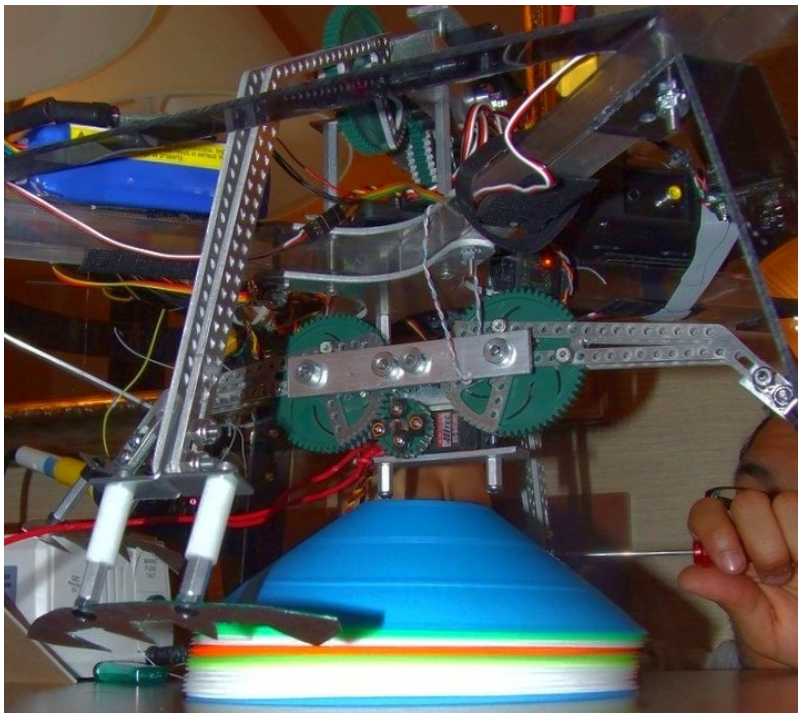


Figure 2.1: NIU Robotics quadrocopter used a manipulator to pick up cones [6].

Manipulators can have varying degrees of freedom, or the number of actuated joints. More degrees of freedom means a manipulator has more versatility and dexterity. An example of a very versatile manipulator would be a robotic arm, which would typically have 5 or more degrees of freedom. However, each actuator would add weight to the manipulator, and on a quadrocopter weight is a limiting factor. The manipulator designed by NIU Robotics for their quadrocopter has 2 degrees of freedom. One degree of freedom contracts one pair of grippers, and the other contracts a separate gripper. The grippers follow a simple semi-circular path as they contract, which is not ideal for picking up objects. Ideally, the manipulator might pull inward horizontally and then, as it begins to grasp the object, lift it upward.

2.1 Linkages

To alter the path that a gripper follows as it is actuated, without adding degrees of freedom, a linkage system can be used. However, linkages are not always able to perfectly achieve a desired motion path due to their limited degrees of freedom. When given three or four desired points in space to travel through, a linkage configuration can be found analytically which will travel exactly through those points. However, a desired motion path may be desired that has more than three or four important points to travel through. For example, if rectilinear motion is desired where the desired path is a straight line (Figure 2.2), choosing three or four points on the line does nothing to ensure the remainder of the path follows the straight line. Furthermore, there are other factors that can influence how ideal a linkage is, such as transmission angles, total material used, and force required on the input. Therefore, linkage design is always an optimization problem. When treated as an optimization

problem, a linkage can be found that passes close to any number of target points in space - not just three or four. By calculating other design parameters and carefully weighing their importance against others, even more ideal linkages can be found. Optimization problems can be solved quickly and efficiently through a well-designed computer program.

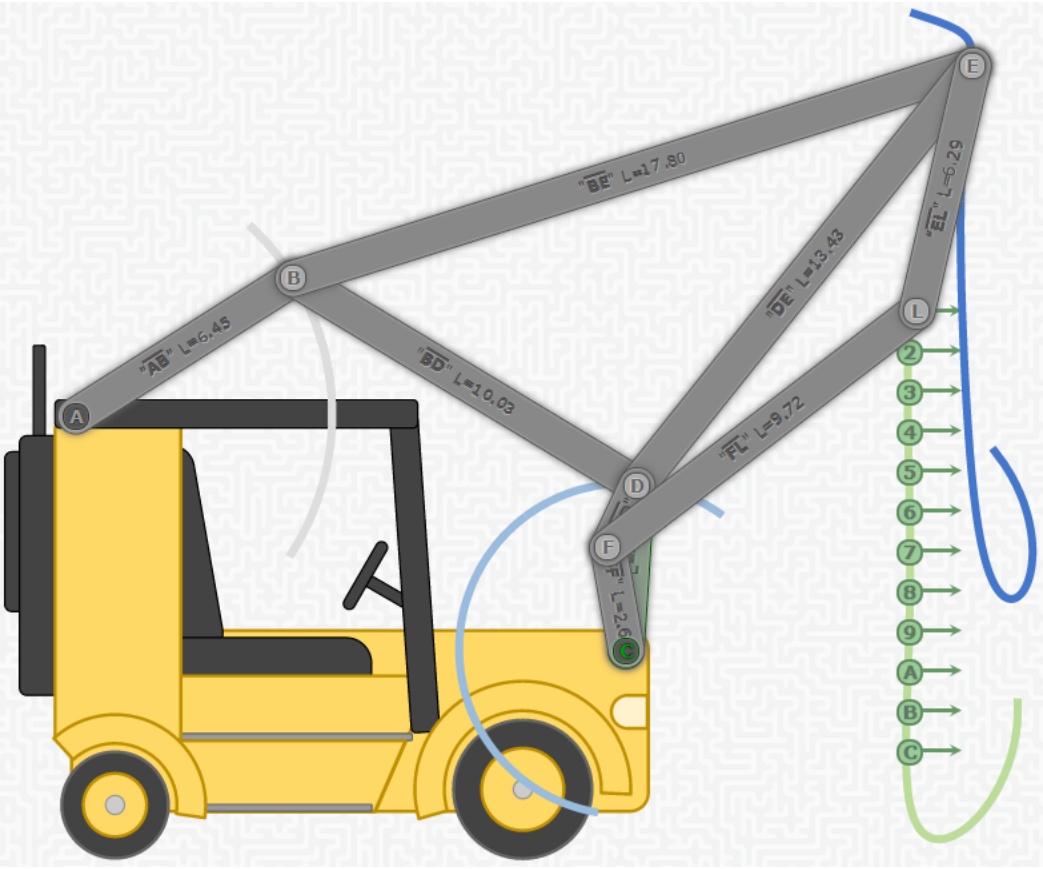


Figure 2.2: Example of an optimized forklift linkage with rectilinear motion.

2.2 Program Architecture

Object-oriented programming compartmentalizes and organizes an overall system. The compartmentalization allows the code to be reused for the multiples of a certain

object in a system as well as in different systems, which helps greatly in the system development. Since a linkage consists of many links and nodes, object-oriented design can greatly reduce the amount of writing required and increase the flexibility of the program once completed. Thus, an object-oriented language was chosen for the system of concern.

2.2.1 Language

There are many object-oriented languages available ranging, from C++ to Python. An often-overlooked language is JavaScript since it has not, until recently, been fast enough. Since it is an interpreted language that runs in a browser, it can be easily run on almost any device without the need for special permissions or compiling. Furthermore, the browser possesses robust graphics support such as layered graphics, highly customizable visual elements and rotations, and it also supports user inputs. To simplify graphics manipulations and maximize compatibility, the open source jQuery [7], jQuery UI [8], jQuery Rotate [9], and Raphael [10] libraries are used. Compatibility is important since functioning in Internet Explorer 9 is one of the design parameters for the project to allow the manipulator to be designed from a variety of computers.

2.2.2 Class Structures

The primary objects in a linkage are nodes and links. A node is a pin joint at a point in space. A link is connected to two nodes and has a length that changes when the linkage configuration is being altered, but while the linkage is being driven.

Both the links and nodes have a graphical component to them which is displayed on screen. There is a driving node, which is the center of rotation, and a driven link attached to the driving node and a driven node. Some nodes, generally two, may be fixed and will never move.

Once the linkage, node, and link classes are written, it is extremely easy to create a linkage. All that is required is to create a linkage instance, define its nodes, and then link those nodes together. Below is code that shows how to create the simple crank rocker shown in Figure 2.3. Note that the driven node is denoted by green text and that the link the node is driving is more green than the rest. Link names are defined by the two notes they are attached to and their length is automatically computed based on the initial position of each node. Additional rigid bodies and four bars may be attached to this simple crank rocker with no significant code rewrite required.

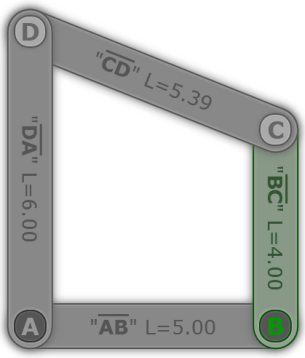


Figure 2.3: Crank rocker.

```
crankRocker = new Linkage();
crankRocker.addNodes([
  {id: "A", x: 0, y: 0, fixed: true},
  {id: "B", x: 5, y: 0, fixed: true, driven: true}, // driving link "B"
  {id: "C", x: 5, y: 4},
  {id: "D", x: 0, y: 6},
]);
crankRocker.linkNodes("A","B");
crankRocker.linkNodes("B","C",true); // driven link
crankRocker.linkNodes("C","D");
crankRocker.linkNodes("D","A");
crankRocker.updateComponents(); // show graphics
```

2.3 Linkage Decomposition

In order to solve the forward kinematics for an arbitrary linkage configuration, the linkage is broken into regions or smaller linkages. The distinction between the kinds of regions is important because each region type is solved a different way.

Three nodes connected to each other by a link form a rigid body region which requires two known node positions to solve for the third. Since all the link lengths are known, law of cosines is used to find the third node position. However, it is important to remember the orientation of the triangle when it is initialized because the mirror image of the triangle might also be solved when using law of cosines. Deciding which nodes belong in a three-bar region is done by searching through each node and checking every pair combination of nodes connected to it. If the pair is connected by a link, the current node being checked and the two nodes that also share a link form a three-bar region.

Four nodes connected by four links with no link connecting the opposite pairs of nodes is a four-bar region, also known as a crank rocker, which requires three known node positions to solve for the fourth. Since there is no physical link connecting the opposite corners of the quadrilateral, the distance can change and needs to be recalculated for each driving input position. The process for deciding what four nodes in a linkage are a four-bar region is similar to the procedure for finding three-bar regions except that it requires another node traversal for one of the pair combinations connected to the originating node. If a pair of nodes connected to the node being checked are not connected to each other, each node connected to one of the pair is checked to see if it is connected to the other of the pair. It is important to check

that the node that is two traversals away does not share a link with the originating node because that would form two three-bar rigid body regions.

2.4 Forward Kinematics

Solving the forward kinematics starts with an initialization phase where the configuration of the regions are found and remembered. Then all fixed nodes are said to be known, and all others are said to be unknown. Now the driven node is moved based on an angle from the driving node. The position of the driven node is now said to be known. From there, each region is checked to see if there are enough known node positions to solve for the remaining unknown node position. The solution for the unknown node position will vary based on whether it is a three-bar rigid body or a four-bar crank rocker. This process of searching for a solvable region may need to be repeated up to the number of regions there are in the linkage because the required sequence for solving regions is not known beforehand.

2.4.1 Three-Bar Forward Kinematics

First, the triangle is analyzed so that its interior angles and configuration can be remembered when it is translated and rotated later. The interior angles are found using law of cosines. Since it is not known beforehand which node position will be need to be solved for, the configuration is found for each "From" node, so that any "Target" node can be found as shown

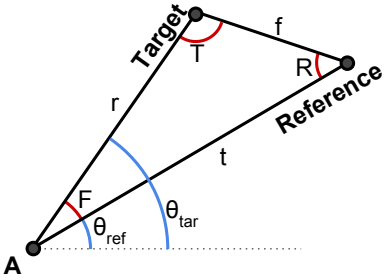


Figure 2.4: Three-Bar variables.

in Figure 2.4. Just knowing the interior angles is not enough to fully describe the rigid body configuration because the mirror image of the triangle also has the same interior angles - a problem known as chirality. The chirality of the triangle is checked by measuring against a fixed axis.

$$F = \cos^{-1}\left(\frac{r^2 + t^2 - f^2}{2 * r * t}\right) \tag{2.1}$$

$$PorM = \begin{cases} 1 & \text{if} \quad \theta_{tar} - \theta_{ref} + F \neq 0 \\ -1 & \text{if} \quad \theta_{tar} - \theta_{ref} + F = 0 \end{cases} \tag{2.2}$$

Once the triangle has been translated or rotated, the configuration above is the same, but the positions of the from and reference nodes have changed as well as the orientation from the fixed axis.

$$\theta_{ref} = \text{atan2}\left(y_R - y_F, x_R - x_F\right) \tag{2.3}$$

$$\theta_{tar} = \theta_{ref} + PorM * F \tag{2.4}$$

$$x_T = x_A + r * \cos(\theta_{tar}) \tag{2.5}$$

$$y_T = y_A + r * \sin(\theta_{tar}) \tag{2.6}$$

2.4.2 Four-Bar Forward Kinematics

The primary difference between the three-bar and four-bar forward kinematics is that the distance between opposite nodes can change. Therefore, a new length, r is computed.

$$r = \sqrt{\left(x_{known,a} - x_{known,b}\right)^2 + \left(y_{known,a} - y_{known,b}\right)^2} \tag{2.7}$$

$$A = \cos^{-1}\left(\frac{-a^2 + b^2 + r^2}{2 * b * r}\right) \tag{2.8}$$

$$\theta_{ref} = \text{atan2}\left(y_{known,a} - y_{known,b}, x_{known,a} - x_{known,b}\right) \tag{2.9}$$

$$x_{ref} = x_{known,b} + b * \cos(\theta_{ref}) \tag{2.10}$$

$$y_{ref} = y_{known,b} + b * \sin(\theta_{ref}) \tag{2.11}$$

Note that a configuration may be invalid if the result inside of the cosine of the law of cosines equation (Eq. 2.8) is not between -1 and 1. This should be checked and trigger the whole linkage solution to be unsolvable. Also, the chirality of a four-bar linkage also must be determined beforehand since there are two possible general configurations of crank rocker - parallelogram and antiparallelogram.

2.4.3 Traces

Now that the forward kinematics of a linkage of arbitrary configuration can be solved automatically, it is useful to trace the path that the end manipulator will take through the full range of motion of the driving input angle. To create a path, the forward kinematics are solved at some resolution, such as every 1°. Then the

position of the end manipulator is stored for each of those input angles. Since the orientation, or usually the change in orientation, of the end manipulator is important in an optimization problem, the angle between the end manipulator node and a reference node is also stored as shown in Figure 2.5.

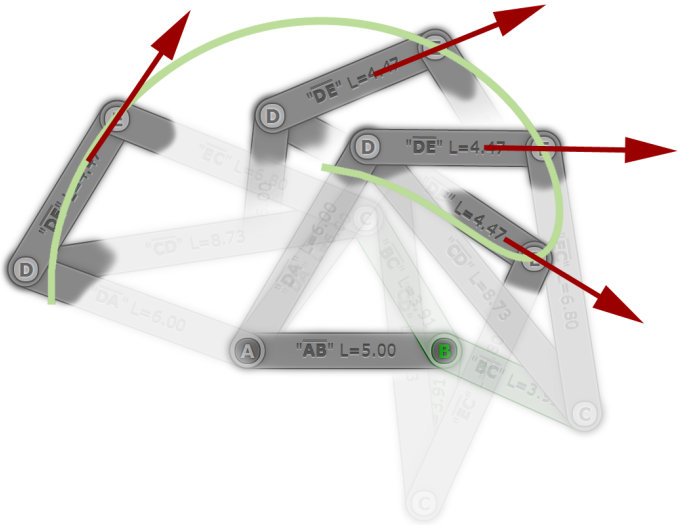


Figure 2.5: Angles on a crank rocker trace.

2.5 Genetic Algorithm

A genetic algorithm is a numerical solution technique for optimization problems. It iteratively changes parameters and compares their error by a fitness function. The fittest member of a population is duplicated and replaces the weakest member that has the most error. The bits of an integer can be thought of as genes. Once the fittest have thrived (or duplicated) and the weakest have died (or been replaced), the genes of the population are randomly swapped [11, pg. 196]. By repeating this intelligent trial and error process, the algorithm will reduce the error and find an improved set of parameters - though the result is not guaranteed to have the absolute minimum error. Over enough iterations the error should be significantly reduced.

Since there are limited computation abilities and the genetic algorithm requires boundaries, it is useful to map the integers used internally that go from 0 to 2^n to a different range that can be shifted and have more or less precision. The conversion can be seen below where v is the external value and g is the internal integer whose bits are used as genes.

$$v_{range} = v_{max} - v_{min} \tag{2.12}$$

$$n_{bits} = \text{ceiling}\left(\log_2\left(\frac{v_{range}}{\text{resolution}}\right)\right) \tag{2.13}$$

$$g_{range} = 2^{n_{bits}} \tag{2.14}$$

$$v = g/g_{range} * v_{range} + v_{min} \tag{2.15}$$

In the example below, the constants have the values:

$$v_{x,max} = 36\ , \quad v_{x,min} = 4\ , \quad \text{resolution}_x = 2 \tag{2.16}$$

$$v_{y,max} = 24\ , \quad v_{y,min} = 8\ , \quad \text{resolution}_y = 1 \tag{2.17}$$

A population of four members initially has random genes. There is a different set of genes for each parameter (x and y). Each set may have different bounds and resolution. The error is found through a fitness function, which will be discussed later. A given v_x and v_y value is said to be fit when it has least error and weak when it has the most error. The weakest member is replaced by a copy of the fittest member (Table 2.1). This ensures that bad genes are eliminated and that good genes propagate through the population faster. Propagation occurs when genes are randomly swapped (Table 2.2). The process of killing the weakest, duplicating the strongest, and randomly swapping genes is repeated numerous times. Note that

gene swapping does not necessarily occur each itteration because some genes may be identical.

Table 2.1: Genetic Algorithm Phase 1 - Finding Fittest and Weakest

Population	g_x (bin)	v_x (dec)	g_y (bin)	v_y (dec)	Error	Action
Member 2	1111	36	1111	24	20	weakest - delete
Member 3	0000	4	0000	8	10	
Member 1	1010	24	1010	18	15	
Member 4	0101	14	0101	13	5	fittest - duplicate

Table 2.2: Genetic Algorithm Phase 2 - Random Gene Swapping

Population	g_x (bin)	g_y (bin)
Member 4	01$\mathbf{0}$1	$\mathbf{0}$101
Member 3	00$\overline{\mathbf{0}}$0	$\overline{\mathbf{0}}$000
Member 1	101$\mathbf{0}$	1$\mathbf{0}$10
Member 4	010$\overline{\mathbf{1}}$	0$\overline{\mathbf{1}}$01

There is also an optional mutation phase that can be added. The infrequent random mutations of the genes ensures that the genes are diverse. Otherwise, given enough iterations, the population may become homogeneous and no further improvements can be made. However, if the mutations are too frequent, then good solutions may mutate undesirably and actually cause an increase in error. Another solution to ensure genetic diversity is starting with a larger population size. Since each member of the population is initially random, there is initially much greater diversity.

2.5.1 Linkage Parameters

Since the links and region configurations are decided by the spacial positions of the nodes, the node x and y coordinates are used as parameters to the genetic

algorithm. This also has the advantage of allowing different orientations and chirality to be tried. Even the different locations of fixed nodes can be tried within the design parameters.

2.5.2 Linkage Fitness Function

The most important part of any optimization problem is the definition of the fitness function which numerically defines a scalar error. A fitness function may measure one or many errors but must weigh each error carefully against the others to ensure each design goal has an appropriate influence on the solution. The optimization algorithm will find a set of system parameters that minimize the cost function. In this case, the x and y coordinates of all nodes marked as being movable are changed to find a trace path that follows the target points as closely as possible. To find the distance error, each distance from a target point to each point on the trace path is compared to find a minimum distance (Eq. 2.18). Once the closest point on a path to a target is found, the angle of the manipulator at that point is also an important component of the fitness function (Figure 2.6). Since an end manipulator can be attached to the end of the linkage at different angles, a direct comparison of the current angle to the target angle is not ideal. Instead, the change in angle throughout the path is important. Therefore, an initial angle is chosen and the error measured is the deviation from the target angle offset by the initial angle (Eq. 2.19).

In Figure 2.6, the $E_{distance}$ for each target is shown by the red line. Then the angle error (θ_{error}) is measured first by finding an initial angle and then by measuring the difference from the target offset by that initial orientation.

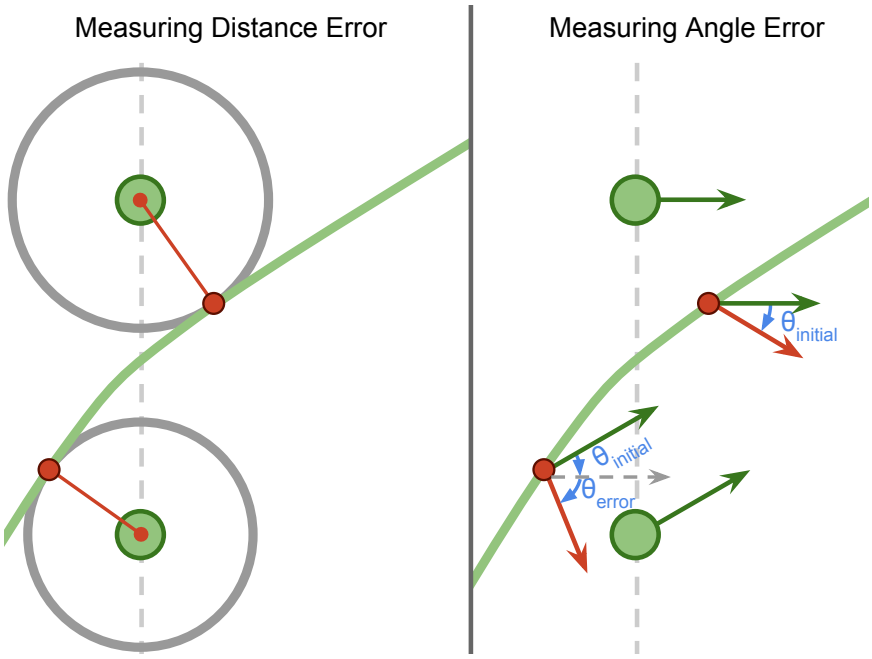


Figure 2.6: Measuring error from targets on a trace.

$$E_{distance} = \sum_{i=0}^{targets} \min(\ ||\vec{x}_{target,i} - \vec{x}_{traces}||\ ) \tag{2.18}$$

$$E_{angle} = \sum_{i=1}^{targets} |\theta_{target} + \theta_{initial} - \theta_{closest}| \tag{2.19}$$

$$E = E_{distance} + w_{angle} * E_{angle} \tag{2.20}$$

Once the distance error and angle error are known, they must be combined because only a single total error is used in the genetic optimizer. Depending on how important it is that the angles of the end of the linkage follow the targets in a specific design, the angle error may need to be weighted (w_{angle}) more or less heavily. If the angle of the end point is not important at all, the weight can be set to zero.

2.5.3 Shrinking Bounds

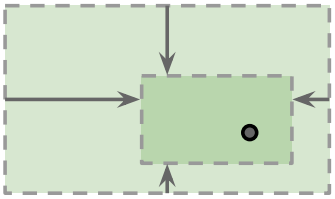


Figure 2.7: Bound shrink.

The genetic algorithm finds good solutions but due to its randomness will often not produce the best solution. Since the genetic algorithm can quickly find a good solution, and as a general rule a better solution will be close to a good solution, the genetic algorithm can be used for just a few iterations. Then the boundaries for the parameters are shrunk towards the good solution (Eq. 2.22, 2.23) and the search resolution is increased slightly (Eq. 2.24). In doing so, extreme parameters which were not found to be ideal can be ignored and a more refined search can begin (Figure 2.7). The search can be refined by shrinking the bounds multiple times to quickly converge towards a much more precise solution. Given the randomness of the genetic algorithm, and that there are often several very different solutions to an optimization problem, the same solution will not necessarily be found every time nor will it be guaranteed to be the best solution.

$$k = \frac{i}{i_{max} + 1} \tag{2.21}$$

$$x_{min} = x_{min} + k * (x_{solution} - x_{min}) \tag{2.22}$$

$$x_{max} = x_{max} - k * (x_{max} - x_{solution}) \tag{2.23}$$

$$\text{resolution} = \text{resolution} - k * \text{resolution} \tag{2.24}$$

2.6 Results

The genetic algorithm based linkage solving program designed is an easy-to-use interface for designing six bar linkages. In particular, it becomes easy to create highly specialized linkages that would be difficult to make with a three precision point inverse kinematics method. For example, the linkage shown in Figure 2.2 has highly rectilinear motion, which alone is difficult to achieve, but also has the unique ability to lean the forklift back at the top and bottom of the range of motion. It would be impossible to use a three precision point method to design a linkage that does both because it is limited to three precision points. Since the genetic algorithm can optimize for many points, all that is required is an additional set of points that are at an angle on either end of the range of motion. Beyond that fact, this is an important usability feature that cannot easily be quantified by experimentation.

However, it can be proven by experimentation that the genetic algorithm can achieve accurate results. As shown in Figure 2.8, the genetic algorithm keeps the distance to within .125 units of the target rectilinear motion, which is about 0.6% of the horizontal work area. The distance error can never be negative due to the fact that distance is measured by Pythagorean theorem, which is never negative. The end manipulator also does not change angle more than 1.75y from the target orientation, so it remains essentially upright through the whole range of motion. When the angle error is added to the fitness function, the absolute value is taken. The linkage formed to achieve the results in Figure 2.8 are shown in Table 2.3. The target path values were each at 15 for all x coordinates, 0 degrees for all angles, and integers from -2 to 9 for the y coordinates. Due to the randomness of the genetic algorithm, the linkage generated may be better or worse than the one described in this experiment.

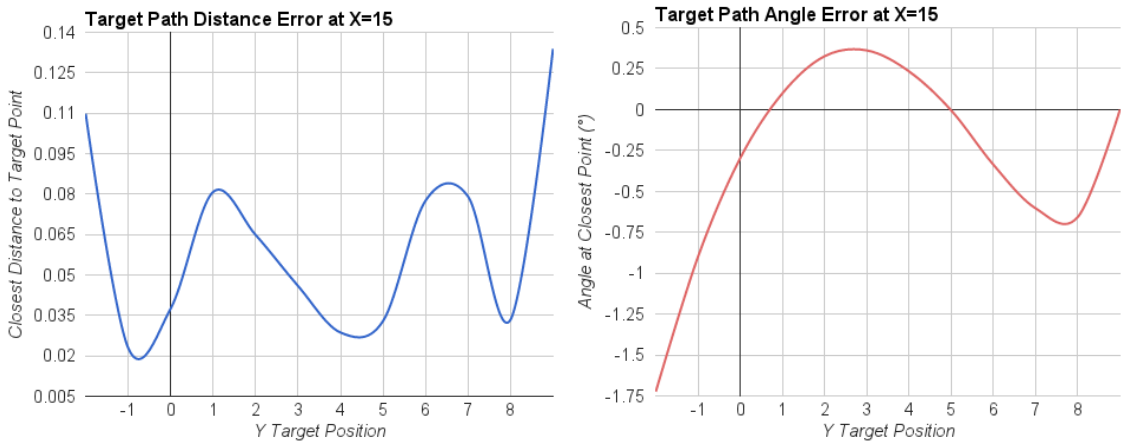


Figure 2.8: Plots of the errors from the desired path.

Table 2.3: The Coordinates of Each Node for the Test in Figure 2.8

Label	X	Y
A	-4.35	2.20
B	-0.26	9.89
C	9.63	0.43
D	6.25	5.06
E	15.27	12.06
F	6.07	2.61
L	14.93	6.28

2.7 Program Usage

Since the program is built with web standards, the functioning application and the source code behind it is available anywhere with an internet connection and a somewhat modern browser. The application is accessible online at:

`http://robotic-controls.com/static/linkage/`

The linkage may be modified by hand by clicking and dragging on the nodes of the linkage. The boundaries used by the genetic algorithm are revealed when a node is clicked and is represented by a green box. The boundary may be modified by dragging on the white handles on the outside of the bounding box. Additionally,

when a node is clicked it may be chosen to be not modified by the genetic algorithm at all, and a setting to alter the resolution used by the genetic number is also made available.

There are several settings that can be changed using the on-page panel shown in Figure 2.9. The input angle is the angle of the driven node from the driving node. The green targets may be moved by clicking and dragging. The target angle may be changed by scrolling the mousewheel while hovering over the green target. Sliding the slider brings the system through its range of motion. The remaining settings are related to the genetic algorithm.

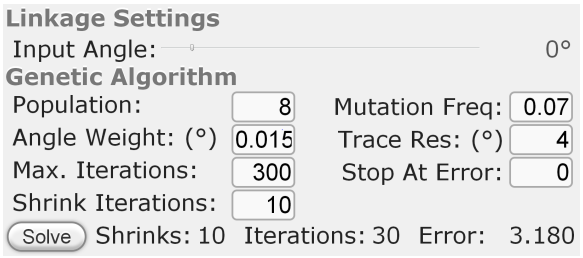


Figure 2.9: The settings panel for the linkage.

Population - The number of members in the genetic population. Each member has a genetic number correlating to the x and y coordinate of each node that is marked "solvable."

Mutation Freq - The probability that a mutation will occur in a parameter. For example, there is a 7% chance that the x coordinate of a node will mutate, but a separate 7% chance is applied to the y coordinate.

Angle Weight - The importance given to the angle error measurement with respect to the distance error as shown in Eq. 2.20. The angle error is measured in degrees.

Trace Res - The degrees of rotation of the input angle between each point on the traces. A higher resolution trace increases the accuracy of the error

measurement but requires more intensive calculations. Traces shown on screen are always 1 degree, but this setting influences the trace resolution used by the genetic algorithm.

Max Iterations - The total number of iterations the genetic algorithm will run. Multiple shrink iterations are enabled; each shrink will take an even share of this total number.

Stop At Error - The algorithm will stop on its search on the current bound size and continue to the next iteration if the total error is below this number.

Shrink Iterations - The number of times the bounds will be shrunk during the search. Increasing this number means each search phase for a given iteration will be divided.

Solve - Clicking this button will begin the genetic search. It may be stopped by clicking this button again.

Shrinks - The number of shrink iterations completed.

Iteration - The number of genetic iterations completed in the current shrink iteration.

Error - The minimum error found in the population in the current iteration. On the last iteration, this is the error of the solution found.

A solution usually takes around 2 seconds to be found given the default parameters. Once a solution is found, analyse the results and consider changing bounds or setting some nodes to not move while solving. Even without changing any of the parameters, the algorithm should be run several times because each time it will be different due to the fact that the genetic numbers are randomly initialized and genes are randomly swapped.

2.8 Research Comparison

To find the uniqueness of this approach, confirm its strengths, and potentially discover weaknesses, it will be compared to several other published methods.

2.8.1 Comparison to Inverse Kinematics Design Method

The inverse kinematics approach, published by Richard Hartenburg and Jacques Denavit, allows a linkage to be synthesized that will perfectly pass through either three [12] or four [13] precision points. While the inverse kinematics approach may be perfectly accurate for those precision points, the system being designed rarely has just a few points that are important. For example, to produce rectilinear motion where the end manipulator follows as straight of a line as possible, the displacement across the entire line is important - not just four points. Since the genetic algorithm discussed in this paper can measure error from an arbitrary number of points, it can produce linkage designs that follow the desired path more closely overall.

When designing a linkage, there are other important design factors, such as transmission angles, forces and stresses at joints, and range of motion. Since the genetic algorithm simply minimizes an error function, it can also optimize these parameters when designing the linkage. The undesirability of a parameter, like poor transmission angles, simply needs to be calculated and added to the existing error and weighted by importance over the path error (distance and angle) measurements.

Another major advantage of the genetic algorithm method is that it only uses the easily calculated forward kinematics. The forward kinematics can be easily modified for different linkage configurations like different numbers of links and joints and can

even be done at runtime without the need to solve simultaneous equations with a tool like Mathcad. As a result, links and joints can be added to and removed from the linkage graphically by the user without the need for programming.

2.8.2 Comparison to Other Genetic Design Methods

There have been other papers published about using a genetic algorithm to synthesize linkage designs. The most important differences between those papers and this one are how the linkage parameters, such as link length and initial configuration, are changed between iterations and how the error is measured between the path that the linkage traces and the desired path.

"Optimal Synthesis of Mechanisms with Genetic Algorithms" [14] was published in the journal *Mechanism and Machine Theory* in 2002. It uses a genetic algorithm to find an optimal synthesis of a planar four-bar mechanism. The fitness function that it uses is the distance error from target points (Figure 2.10). It has a penalty function for the Grashof condition but does not optimize the angle of the end point. The genetic algorithm for the single crank rocker is run only 100 times and the error for six target points is reduced by 99.99% and took 3-40 seconds on a very old Pentium III processor. A modern computer could be 10's to 1000's of times faster than that. However, since a five-bar system is significantly more complicated, one might guess that significantly more iterations are required - though that is not necessarily true due to improvements in the genetic algorithm. The paper concludes that the efficiency and accuracy are comparable to the final error of gradient based-methods.

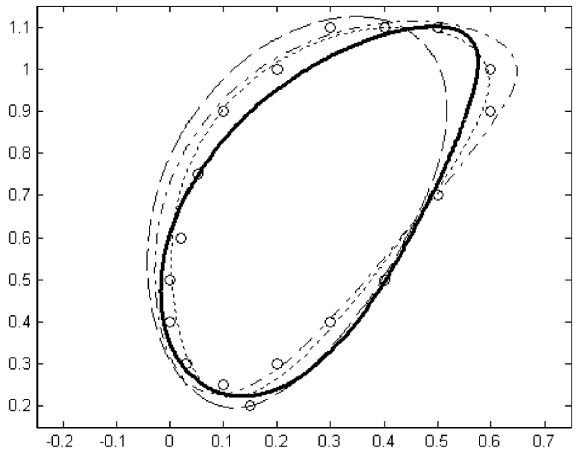


Figure 2.10: Genetic algorithm comparison [14].

$\cdots$: Gradient Method, $\circ$: Target, - -: Genetic Algorithm

The main difference between the optimization in "Application of Genetic Algorithm and Fourier Coefficients (GA-FC) in Mechanism Synthesis" [15] and the other papers cited is that it uses a Fourier series to represent the path of the four-bar mechanism. This approach adds significant complexity because the path generated must be translated, rotated, and scaled in order to compare it to the target path (Figure 2.11). There does not seem to be a major advantage despite this added complexity. The number of iterations used in the genetic algorithm is still 100. One possible benefit is that this method may require less computational power than other optimization algorithms; however, since this paper makes no mention of the programming language or hardware used in its test, it is hard to tell. Since it was published more recently it is safe to assume the hardware is significantly faster, so their algorithm may actually not be any faster at all. Furthermore, one of the papers specifically mentioned that they used MATLAB for their genetic algorithm, which is possibly less efficient than what the first GA paper used. While the Fourier series representing the path has the advantage of being continuous, it is still an approximation of the desired path and may not necessarily be more or less accurate

than a target point method. Furthermore, when actually designing a linkage, the desired output may not be a complete path, but a portion of a path. For example, to achieve a straight line motion over a small portion of a path, it is difficult to design a target path for the portion that is irrelevant.

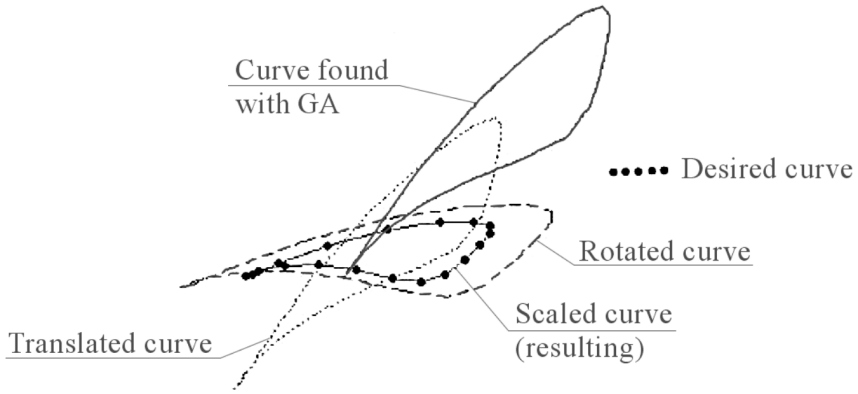


Figure 2.11: Fourier coefficient genetic algorithm [15].

The "Differential Evolution" [16] technique described in this paper is a specialized genetic algorithm shown in Figure 2.12. The target definitions are possibly less useful for general design purposes than either of the other papers. While it allows the user to define a target orientation for the end point, it requires that the target point and angle be paired with an input angle from the crank. When a desired path and orientation along a path are the design goals, it is completely irrelevant what of the crank is with respect to a specific target point. If a different design were to achieve the overall path more closely, but perhaps it follows the points faster or even in reverse order, it would still be more desirable. The advantage to this approach is that the forward kinematics only need to be generated for the given input angle and does not need to generate a full path, so it should be substantially computationally faster. The end of the article happens to mention a common form of rectilinear linkage - the Ackerman steering linkage, which is based on a six-bar, Watt's mechanism. The

design of this mechanism does not seem to employ the same differential evolution technique discussed earlier in the paper.

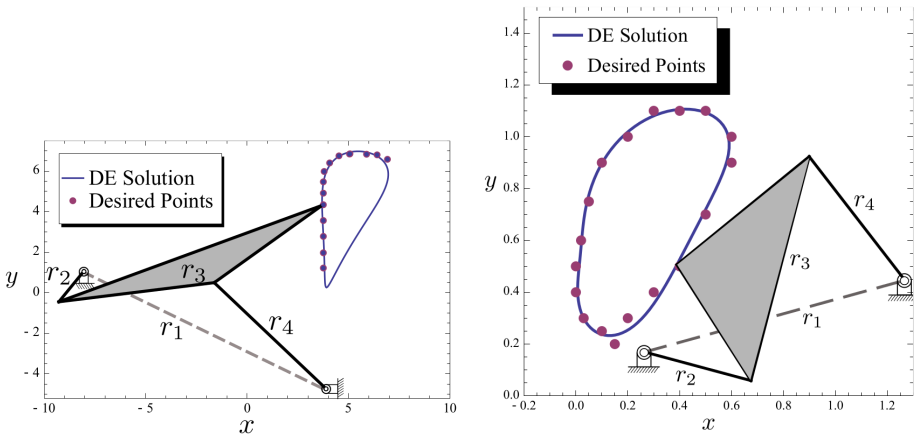


Figure 2.12: Differential evolution genetic algorithm [16].

"Multi-objective Optimization Using Genetic Algorithms: A Tutorial" [17] discusses genetic algorithms generally and their use for optimization in engineering. Potentially useful concepts are detailed such as elitism, where the fittest member will not be altered during gene swapping. Elitism would likely be a good feature to add to the general genetic algorithm used in this paper. It also shows some ways to solve for more than one objective. The paper lists 13 different kinds of multi-objective genetic algorithms. The weighted sum approach for fitness functions is the first approach for multi-objective optimization but also shows how alternating the objective function can be used. The paper also explores some complex ways to ensure diversity.

2.9 Genetic Algorithm Manipulator Design Conclusion

Once the forward kinematics for the linkage are defined, the design of the linkage for inverse kinematics becomes an optimization problem. A simple genetic algorithm

was then used to optimize the linkage configuration for a given target path. The algorithm itself is relatively unimportant compared to defining how the parameters of the algorithm vary and defining an efficient and accurate cost function. The algorithm design implemented in this paper appears to be more useful for the design of linkages than previously published algorithm implementations. The resulting application functions quickly enough to help design linkages interactively - even through a web browser.

An example of a design produced by the application is shown in Figure 2.13. Using only a single degree of freedom, a complex scooping motion can be designed. If the same linkage were mirrored and driven by the same actuator, it could hold and lift an object of a wide range of sizes. If the size of the object to be picked up is known beforehand, the target markers can simply be moved and the design would be optimized again.

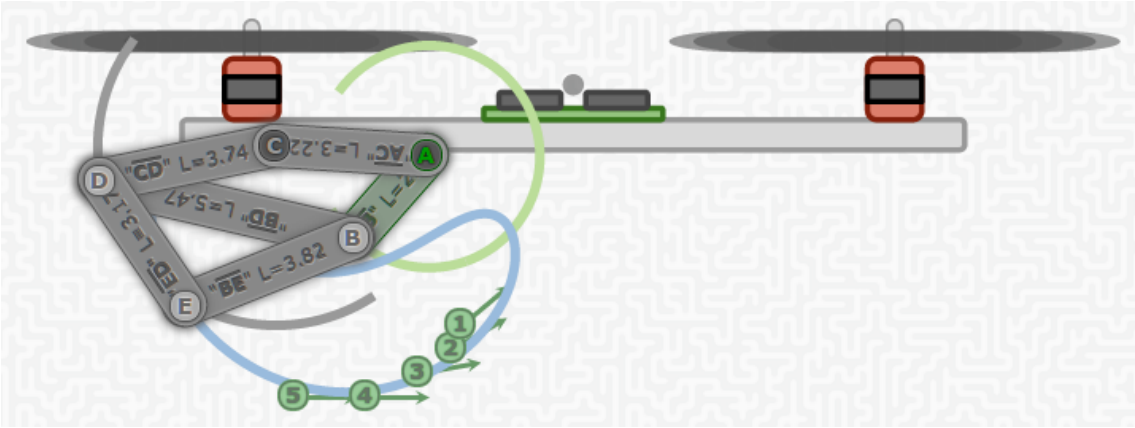


Figure 2.13: Example of a genetically optimized linkage for a quadrocopter.

CHAPTER 3

BLOCK SORTING AND SIMULATION

3.1 Introduction to Simulation

The simulation of a quadrocopter that sorts colored blocks can be decomposed into several independent parts, namely, the plant (i.e., the quadrocopter without intelligent control) that accurately models the dynamics of a quadrocopter; the proportional, integral and derivative (PID) controller that controls the movement of the quadrocopter; the artificial neural network (ANN) that is used to learn the color patterns of the blocks and decide which group a certain color block belongs to; as well as the path planning that tells the PID controller where to go based on the decision of the ANN (Figure 3.1). While a sufficiently complex ANN could fulfill both the tasks of controlling and sorting, a well-designed PID controller is robust, efficient, and it does not require training as an ANN does. Furthermore, due to the integral gain, it can even learn and compensate for unanticipated errors.

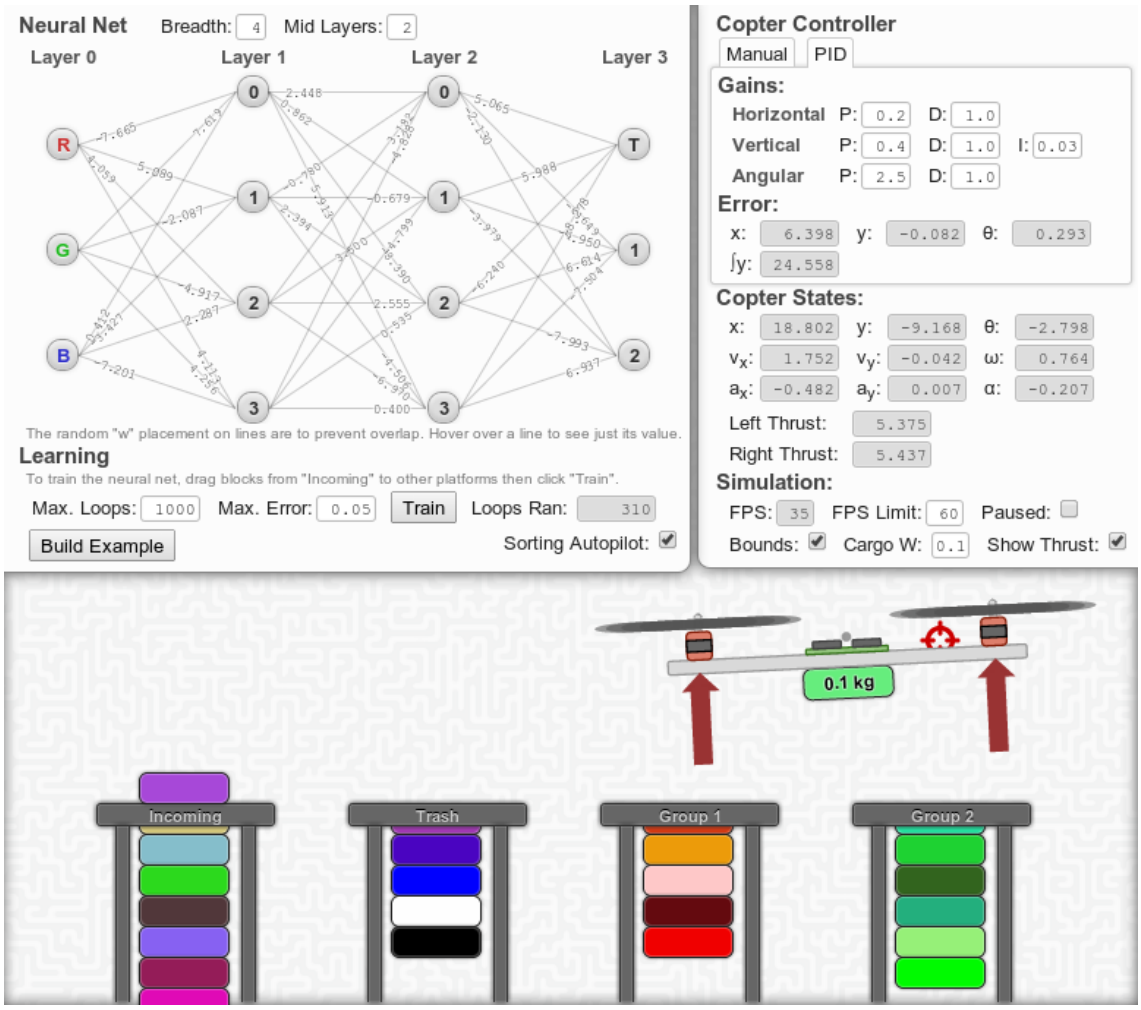


Figure 3.1: Simulation of the quadrocopter sorting blocks.

3.1.1 Programming Environment

As with the linkage solving program, the JavaScript object-oriented programming language will be used for the simulation program. The object-oriented feature of the language will allow components of the simulation to be compartmentalized and reused. Object-oriented programming is particularly advantageous for the ANN as well as for the graphics. The four main parts, namely, the simulated plant, controller, ANN and path planner (or autopilot) are decomposed into the objects

and relationships as shown in Figure 3.2. Though there are multiples of certain objects used in the system such as blocks, nodes, landing pads and links, it requires minimal effort to add the multiples, once the object is defined.

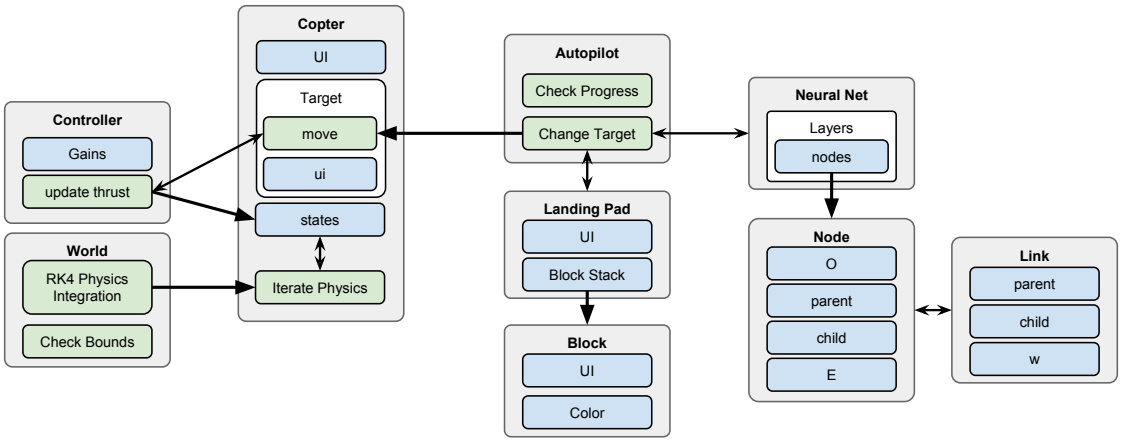


Figure 3.2: Simplified block diagram of overall object-oriented program design.

Again, JavaScript will allow the simulation to be run in-browser on a very wide variety of devices and platforms. Additionally, it provides very robust graphics support, which will be useful for the real-time 2D environment. 3D graphics representations are possible in-browser [18] but would add unnecessary complexity. To further simplify graphics manipulations and maximize compatibility, the open source jQuery [7], jQuery UI [8] and jQuery Rotate [9] libraries are used. If some of the components of this simulation are to be used on a physical quadrocopter, like the artificial neural network, microcontrollers exist that can be controlled by JavaScript [19], which means the code would not have to be rewritten.

3.2 Dynamics Simulation

The simulation environment provides the motion and behavior of the plant, that is the quadrocopter and its rotors without the controller. It is responsible for the dynamics of the plant such as position response to gravity and thrusts over time. The results of the simulated dynamics are used by the controller. If a comparable physical system is made, the states of the quadrocopter are measured instead of simulated by using numerical integration.

3.2.1 3D Dynamics of Quadrocopter

First, the dynamics of a real quadrocopter need to be established. A quadrocopter is a helicopter with four top-mounted propellers and no tail rotor, as shown in Figure 3.3. The x-y plane is parallel to the ground, and the z direction is the altitude. The yaw (z-axis rotation) control is achieved by having two of the four propellers rotating in opposite directions. To prevent yaw movement, the two sets of propellers must have the same speed. Otherwise, the copter rotates in the direction of the slower propeller pair, since this direction offers less air resistance and therefore imparts less torque on the craft than the faster pair.

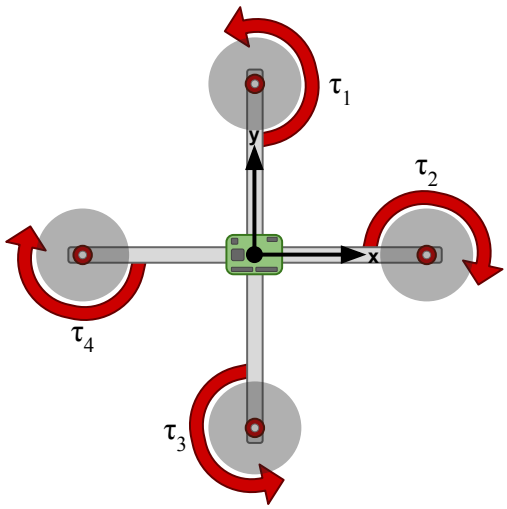


Figure 3.3: Top-down view of FBD.

Properly simulating the yaw effect, conserving angular momentum, and keeping track of 3D rotations in general are somewhat difficult. Additionally, it is difficult,

but not impossible, to represent 3D objects in a traditionally 2D browser environment. Either WebGL or matrix rotations and projections with 2D SVGs [18] can be used. So for simplicity and due to the limitations of the web browser graphics environment, the system is only simulated in two dimensions.

3.2.2 2D Dynamics of Quadrocopter

The quadrocopter is represented by a rigid body, shown in Figure 3.4. Since the quadrocopter is normally in 3D, the z axis is the altitude and the x axis is parallel to the ground. Thus, using Newtons Second Law, the dynamics are:

$$\Sigma F = ma \Rightarrow a = \frac{\Sigma F}{m} \tag{3.1}$$

$$a_x = \frac{\Sigma F_x}{m} = \frac{T_x}{m} \tag{3.2}$$

$$a_x = \frac{(T_L + T_R) * \sin(\theta)}{m} \tag{3.3}$$

$$a_z = \frac{\Sigma F_z}{m} = \frac{T_z + mg}{m} \tag{3.4}$$

$$a_z = \frac{(T_L + T_R) * \cos(\theta)}{m} + g \tag{3.5}$$

$$\Sigma \tau = I\alpha \Rightarrow \alpha = \frac{\Sigma \tau}{I} \tag{3.6}$$

$$\alpha = \frac{(L \times T_L) - (L \times T_R)}{I} \tag{3.7}$$

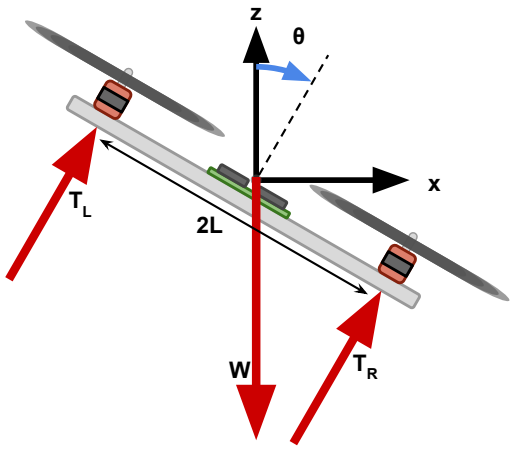


Figure 3.4: Side view of FBD.

The direction of positive rotation is clockwise, which is opposite of the convention that generally favors right-hand rule. This is chosen intentionally due to the fact that browsers perform rotations in degrees clockwise. Also note that g is negative. The dynamics Eqs. (3.3), (3.5) and (3.7) are translated to JavaScript, respectively:

```
ax = this.thrust * Math.sin( theta ) / (this.m + this.cargo);
az = this.thrust * Math.cos( theta ) / (this.m + this.cargo) + g;
alpha = (this.tl - this.tr) / this.I;
```

3.2.3 Physics Simulation Using Numerical Integration

Simulation requires a solution of the position states in Eqs. (3.3), (3.5) and (3.7). To relate the accelerations of the system to positions, each equation can be transformed from a single second-order differential equation to two (simultaneous) first-order differential equations.

$$\frac{d^2}{dt^2}\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \\ \alpha \end{bmatrix} \quad \Rightarrow \quad \frac{d}{dt}\begin{bmatrix} x \\ v_x \\ y \\ v_y \\ \theta \\ \omega \end{bmatrix} = \begin{bmatrix} v_x \\ a_x \\ v_y \\ a_y \\ \omega \\ \alpha \end{bmatrix} \tag{3.8}$$

Now, to simulate the physics of the plant, all that is needed is to integrate the right-hand side of Eq. (3.8). However, it is impossible to integrate the dynamics equations mathematically, since the thrusts are not known a priori. They are therefore integrated with a numerical approximation method. One example of such a method is multiplying the right-hand side by a small time step and adding it to the previous value, which is known as Eulers method [20, pg. 3] as seen in Figure 3.5.

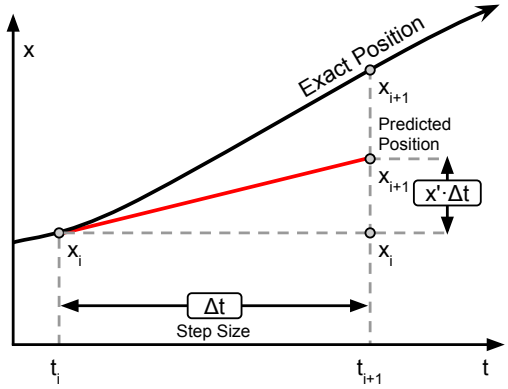


Figure 3.5: Euler numerical integration accumulates error with non-zero step sizes.

$$x_{i+1} = x_1 + f(x_i, t)\Delta t \quad \text{where} \quad f(x,t) = \frac{dx}{dt} \approx \frac{x_{i+1} - x_i}{\Delta t} \tag{3.9}$$

By substituting the numerical approximation of the derivative into Eq. (3.9), it is actually found that it simply says $x_{i+1} = x_{i+1}$. The problem is that the numerical approximation of the derivative is not necessarily a good guess since the time step is not infinitesimal. This becomes increasingly problematic for larger time steps, since the approximation is worse. Furthermore, the error can accumulate since each step is based on the previous step. To reduce the error, the time step is made as small as possible. However, smaller time steps require more integrations over the same amount of time. In a real-time simulation, computational power limits how small the time step can be. The real time it takes to process a time step cannot be greater than the time step. Otherwise, the simulation would be slower than real-time operations.

There are several more efficient numerical solutions. One is the Runge-Kutta fourth-order (RK4) method for solving ordinary differential equations using the intelligent combination of slopes from several steps to form the following equations [21, pg. 2-3]:

$$x_{i+1} = x_i + \frac{1}{6}\left(f_0 + 2f_1 + 2f_2 + f_3\right)\Delta t \tag{3.10}$$

$$f_0 = f\left(x_i, t_i\right) \tag{3.11}$$

$$f_1 = f\left(x_i + f_0\frac{1}{2}\Delta t, t_i + \frac{1}{2}\Delta t\right) \tag{3.12}$$

$$f_2 = f\left(x_i + f_1\frac{1}{2}\Delta t, t_i + \frac{1}{2}\Delta t\right) \tag{3.13}$$

$$f_3 = f\left(x_i + f_2\Delta t, t_i + \Delta t\right) \tag{3.14}$$

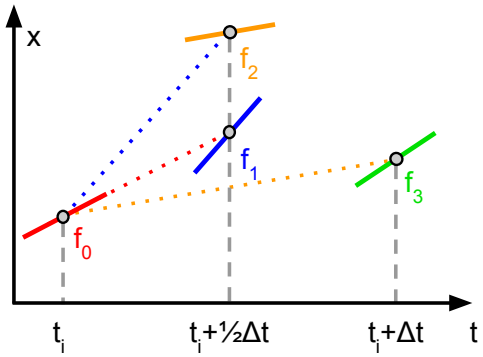


Figure 3.6: RK4 slope components.

Each RK4 step is actually a weighted average of four different Euler steps and half-steps, as seen in Figure 3.6. Each of the four steps uses the result of the position in the previous step to find a different slope. The middle two slopes are considered

to be more representative than the outer two. Therefore, they are weighted twice as much in the average as shown in Eq. (3.10). Due to this, RK4 integration is 16 times more accurate than Euler integration [21, pg. 7], but requires about four times more work. It is regarded as being more efficient and accurate due to the fact that it uses the slopes at intermediate time step guesses to correct itself before continuing. The result is that the numerical solution for Eq. (3.8) using RK4, given a fixed amount of processing power, is four times less prone to accumulating error than if it used Euler integration.

3.3 Translational Controllers

Proportional, integral, and derivative controllers are robust methods of controlling dynamic systems. Different controllers must be designed for different dimensions of control. For example, the horizontal and vertical position controllers function differently, primarily due to the force of gravity. Later, orientation controllers are also designed.

3.3.1 Horizontal and Vertical Control

Conventional PD and PID controllers are used to guide the quadrocopter towards its target in the horizontal and vertical directions, respectively. The horizontal controller takes the actual x position and the desired x position to decide a "pull" that is similar to the exertion of a spring. However, to prevent oscillation on a spring, a damper is needed. Similarly, to prevent any overshoot and oscillation for

positioning, the horizontal velocity is used to slow down the quadrocopter before it gets to the destination.

$$F_x = k_{h_P}(x_{target} - x) - k_{h_D} v_x \tag{3.15}$$

The vertical controller is similar to the horizontal controller. However, since gravity acts in the vertical direction, there are a few additions. First, since the mass of the craft is known, a feed-forward component can be used to eliminate the steady-state error [22]. When there is cargo, the additional mass of the cargo is not known by the controller. So, an integral gain must be added.

$$F_y = k_{v_P}(z_{target} - z) - k_{v_D} v_z + k_{v_I} I + mg \tag{3.16}$$

$$\text{Where: } I_{plain} = \int (z_{target} - z)\, dt \tag{3.17}$$

Principles of fuzzy logic can be used to enhance the controller as described hereafter. The integral gain is important to correct the steady-state error caused by the unknown additional mass of the block to the plant, but it takes a while to make corrections and it also causes more oscillation. To limit these drawbacks, a well-known procedure is to substantially reduce integral accumulation by reducing the influence of large errors and increasing the influence of small errors. Instead of passing the exact error directly to the integrator, the exact error is passed through a fuzzy logic membership function first. Since the integral controller is only needed to compensate for the unknown mass of the cargo, it should not be used at all when there is no cargo. The fuzzy logic inference table (Table 3.1) can be constructed intuitively:

Table 3.1: Integrand Error (z_{err}) Scaling Logic Table

	Down Far	Down Close	At Target	Up Close	Up Far
With Cargo	Small	Small	Zero	Small	Medium
No Cargo	Zero	Zero	Zero	Zero	Zero

Table 3.1 shows how the vertical error is fed to the integrator. This method has the major advantage of learning the weight of the cargo and remembering it for the next time the craft picks up cargo. Without the toggle feature, the integral controller would have to reduce to nearly zero after it drops off the cargo due to the removal of the additional mass, only to increase it when it picks up another. If the weight of the craft was not known by the controller in the feed-forward portion, an integrator would be necessary all the time. Alternatively, two separate integrators could be used: one for the craft and another for the cargo.

Table 3.1 also takes advantage of the fact that the integral correction should be faster in rising than in landing. The first time the craft picks up cargo, the integral controller needs to correct for the additional weight quickly while taking off. Once the craft has taken off, it has a while to fine-tune the integral correction while approaching the cruising altitude. It is not particularly important to achieve a perfect cruising altitude since, while the craft is up in the air, there is nothing to hit. Conversely, the integral correction should not need to be reduced as often since it only needs to reduce its own over-corrections and not steady-state errors. Overshoot during landing is extremely undesirable, since it might cause a disastrous collision. In addition to preventing too much integral correction, the integral gain should also be zero when there is no cargo. Otherwise, the integral correction of the controller lifts the craft too much and it is not able to come back down since the integrator is not collecting errors while there is no cargo.

Before the toggle adjustment to the integral correction is applied, first the continuous integrator, as shown in Eq. (3.17), must be transformed into a discrete numerical integrator, as shown in Eq. (3.19), by using the trapezoid rule. RK4 cannot be used because only the current and previous states are known. Conversely, RK4 could be used in the physics simulation because differential equations were known. In practice, a running total is kept and only the trapezoid from the current time step is added to the total.

$$z_{err} = z_{target} - z \tag{3.18}$$

$$I_{plain} = \int z_{err} dt = \sum_{i=\text{start}}^{\text{now}} \frac{\Delta t_i}{2} \left(z_{err,i} + z_{err,i-1} \right) \tag{3.19}$$

Now, a modification to the plain integrator in Eq. (3.19) is made by turning off the integral controller and stopping the integrator from collecting more data when the quadrocopter has no cargo, which is the feature defined by the bottom row in Table 3.1. The binary variable cargo is defined in Eq. (3.21). The `cargo` condition is a simple yes or no, which is multiplied in Eq. (3.22) to form an intersection relationship (also known as an "and" relationship.) The area of the trapezoid that is added to the integral when there is no cargo is zero. However, this would lead to steady-state error when there is no cargo since the integrator, which has compensated for the additional weight of the cargo, cannot be altered. To counter the steady-state error, an additional condition is added outside of the integrator to temporarily suppress it.

$$\texttt{cargo} = \begin{cases} \texttt{1} & \text{if craft has cargo} \\ \texttt{0} & \text{if craft does not have cargo} \end{cases} \tag{3.20}$$

$$I_{toggle} = \texttt{cargo}_{now} \cdot \sum_{i=\text{start}}^{\text{now}} \texttt{cargo} \cdot \frac{\Delta t_i}{2} \left(z_{err,i} + z_{err,i-1} \right) \tag{3.21}$$

To realize the top row in Table 3.1 as an equation, a fuzzy logic membership function is applied to the vertical error, as shown in Eq. (3.22).

$$I_{fuzzy} = \texttt{cargo}_{now} \cdot \sum_{i=\text{start}}^{\text{now}} \texttt{cargo} \cdot \frac{\Delta t_i}{2} \left(f\left(z_{err,i}\right) + f\left(z_{err,i-1}\right) \right) \tag{3.22}$$

The membership function in Eq. (3.22) can be constructed mathematically through a combination of the use of a square root and the limitation of the error by imposed conditions.

$$z_{err,adj} = \text{sign}(z_{err}) \sqrt{\text{abs}(z_{err})} \tag{3.23}$$

$$f(z_{err}) = \left\{ \begin{array}{ccccc} z_{err,adj} & \text{if} & -1 < & z_{err,adj} < & 5 \\ -1 & \text{if} & & z_{err,adj} \leq & -1 \\ 5 & \text{if} & & z_{err,adj} \geq & 5 \end{array} \right\} \tag{3.24}$$

As a result of the modification of I_{plain}, to I_{toggle}, and then to I_{fuzzy} as shown in Eqs. (3.19), (3.21) and (3.22) respectively, the vertical controller in Eq. (3.16) becomes substantially more stable and effective despite the presence of the integral controller.

3.3.2 Takeoff and Landing Experiment

An experiment is conducted to find the effects of the three kinds of the integrators, i.e., the plain, the toggle and the fuzzy. The quadrocopter completes 14 cycles, each of which consists of the four timed phases shown in Figure 3.7, i.e., cargo pickup, cargo dropoff, takeoff without cargo and landing without cargo, respectively. The procedure required for actually sorting blocks would be similar, but there is no horizontal movement in the experiment, since the integrator only acts vertically.

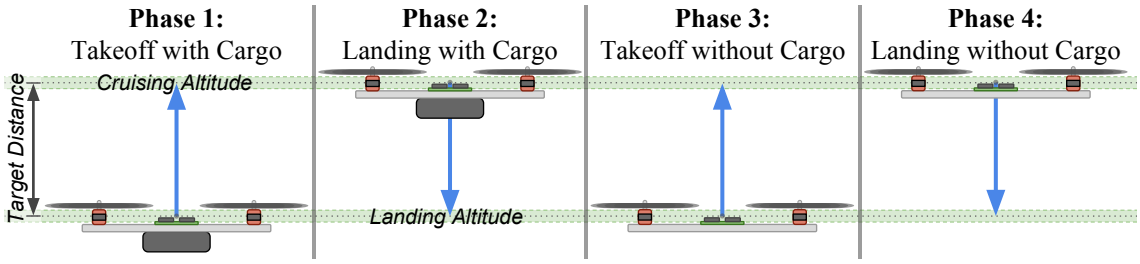


Figure 3.7: A single cycle of takeoff and landings.

The plain integrator is the commonly-used, always-on integrator and integral gain referred to in Eq. (3.19). The toggle integrator is the same as the plain integrator, except that it stops collecting the error in the integrator and eliminates integral gain when there is no cargo, as shown in Eq. (3.21). The fuzzy integrator refers to Eq. (3.22) which behaves like the toggle integrator but uses the fuzzy membership function to achieve more stable control. The integral gain is cut in half for the plain and toggle integrators so that they match more closely the scale of the fuzzy integrator because it adjusts the integrand with a square root. Since changes to the integrand will change how the integral controller behaves, the integral gain is changed in an attempt to make the controllers behave similarly.

The settle time is the time it takes to reach the target and stay there. It is measured in this experiment by checking to see if the speed is less than 0.075 meters per second, or essentially stopped, and the position is within 0.75 meters of the target altitude. A tolerance is necessary because a controller only converges towards a target and does not ever arrive at the exact target. The target position depicted in the two green lines in Figure 3.7 is similar to the green region in Figure 3.8. When the target altitude is approached, the vertical error is

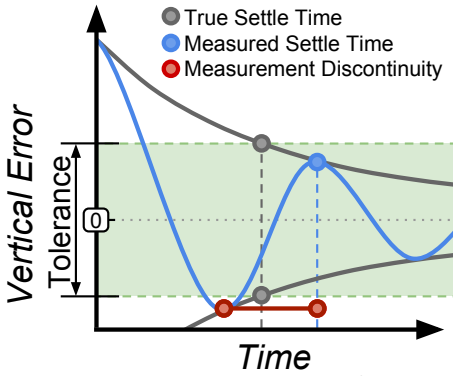


Figure 3.8: Measuring settle time.

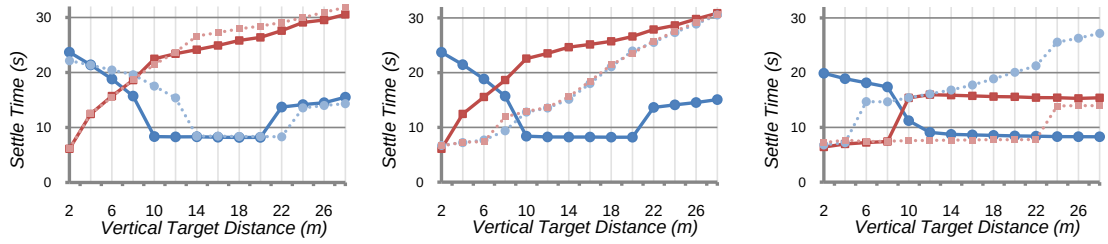
reduced. When the vertical error (shown by the blue line in Figure 3.8) is reduced enough to be within the tolerance, it is said to be at its target. The velocity of the craft is checked to see if the craft is staying at its target altitude and not overshooting it. However, the velocity condition also restricts the settle times to peaks in vertical error. As a result, there are large jumps in settle time when a peak exceeds the error tolerance. The settle time has to wait for the next peak to occur within the error tolerance region. This results in the discontinuity shown by the red line in Figure 3.8. The true measurement of the settle time is to intersect the maximum error tolerance with the bounds of the oscillation shown by the dark grey curve in Figure 3.8, but it is considerably more practical to use the numerical approach and only check the peaks.

The largest error in measuring the settle time numerically (and therefore the largest discontinuity) is half a period of the oscillation, which occurs immediately after the overshoot exceeds the error tolerance. In the vertical controller, overshoot is particularly undesirable due to the possibility of hitting the landing pad too fast and causing damage. Therefore, these large discontinuous increases in settle time should be observed carefully, especially in the landing phase.

Aside from the overshoot, the integral controller impacts the settle time due to the windup time, which is the time it takes for the integral component to change because it is integrating the error with respect to time. So, time has to pass before the error, or the current distance to the target, accumulates area in the integrator. Windup time is one source of a more continuous change in settle time. Windup time also exacerbates how much overshoot increases settle time since the windup time takes longer for close distances.

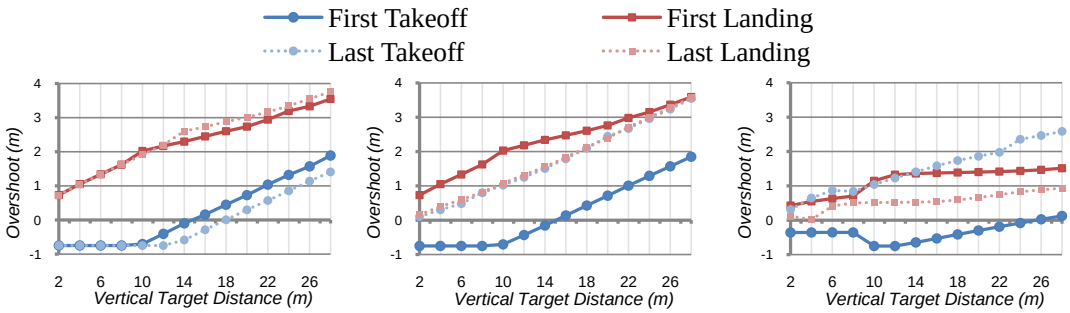
Windup time is the root cause of both overshoot and undershoot. Undershoot is when the integral controller does not compensate enough for steady-state error before

settling close to the target. For takeoff, the craft lifts off towards its cruising altitude but slows down below the target altitude as the integrator slowly compensates for the mass of the cargo. Overshoot, in this experiment, is measured by the maximum distance past the target; however, undershoot occurs before the craft reaches the target. Thus, no undershoot measurement is lower than the tolerance distance of -0.75 meters, which is where the craft stops moving towards the target and moves onto the next phase.



(a) Plain Integrator (b) Toggle Integrator (c) Fuzzy Integrator

Figure 3.9: The first and 14th takeoff and landing settle times with cargo.



(a) Plain Integrator (b) Toggle Integrator (c) Fuzzy Integrator

Figure 3.10: The first and 14th takeoff and landing overshoot with cargo.

The experiment results shown in Figure 3.9 and Figure 3.10 reveal the different behaviors of the three integral controllers while there is cargo, as depicted by phases 1 and 2 in Figure 3.7. Of course, the craft also completes phases 3 and 4 in between cycles.

The experiment results from the plain integral controller defined in Eq. (3.19), are presented in Figure 3.9a and Figure 3.10a. Figure 3.9a shows the settle time for takeoffs and landings. In the first takeoff, when the altitude change is small (2-10 meters), there is a long windup time and it undershoots (or briefly stops below) the cruising altitude until a target distance of 10 meters is tested, as seen in Figure 3.10a. The reduction of undershoot to be within tolerance causes the discontinuous jump in settle time in Figure 3.9a that occurs at 10 meters. Conversely, the craft begins to overshoot after the target distance is more than 20 meters in the first takeoff. In the first landing phase for any target distance, there is overshoot below the landing altitude that is greater than the tolerance, which is extremely undesirable due to the possible high-speed collision with the landing platform. It can overshoot the landing altitude by nearly 2 meters, which would likely be a catastrophic collision. The first and last takeoffs, as well as the first and last landings, have very similar settle times, respectively. The plain integral controller has to wind down when it has no cargo, as shown in phases 3 and 4 in Figure 3.7. The plain integral becomes close to zero while there is no cargo, but it must wind back up when there is cargo again in the phases 1 and 2 in the next cycle.

The experiment results from the toggle integral controller, defined in Eq. (3.21), are presented in Figure 3.9b and Figure 3.10b. The curves of the first takeoff and first landing are identical to those presented in Figure 3.9a and Figure 3.10a. This is because the first takeoff and the first landing are both with cargo, and the integral controller has not switched off yet. Since the toggle feature is what is different between the two kinds of integral controllers, the two should perform the same at this point. Interestingly, the last takeoff and the last landing curves become identical to each other. By the 14th takeoff and the 14th landing with cargo, as shown in phases 1 and 2 in Figure 3.7, the integral controller has compensated for the mass

of the cargo. By then, the integral controller is winding up and down due to the error caused by the change in target altitude alone, which is the same distance for going both up and down. Before this point, the integral controller had not yet fully compensated for the mass of the cargo.

The experiment results from the fuzzy integral controller defined in Eq. (3.22) are shown in Figure 3.9c and Figure 3.10c. Generally, the curves for the fuzzy integral controller in Figure 3.9c and Figure 3.10c are substantially lower than the curves for the other two controllers. As seen in Figure 3.10c, in the first landing phase the overshoot is reduced enough so that it is within tolerance until 6 meters, as opposed to merely 2 meters, and in the last landing it does not occur at all until a 22-meter target distance, which is a massive improvement over even the toggle integral controller. This is likely the result of the limit imposed on the negative corrections as defined in Eq. (3.24). The first takeoff takes less time, according to Figure 3.9c, at either very small (2-4 meters) or large distances (20-26 meters), likely due to the square root function that is increasing the influence of very small changes and reducing the influence of large changes. The first takeoff also never experiences significant overshoot, but does experience undershoot. The fuzzy integral controller is, however, more prone to experiencing overshoot on the last takeoff than the plain integral controller. As described in the design of the fuzzy integrator in Table 3.1, overshoot is more acceptable in the takeoffs than in the landings due to the risk of a crash while landing. Also, long settle time is less important on takeoff due to the relatively long time spent moving horizontally at the cruising altitude to move between blocks.

A very important situation, not shown in Figure 3.9 and Figure 3.10 at all, is that the toggle and fuzzy integral controllers disabled completely when there is no cargo, as shown in phases 3 and 4 in Figure 3.7. The craft behaves much more efficiently

when there is no cargo, as a result of the toggle feature. Since the proportional and derivative controllers are the only ones acting, there is no overshoot caused by the integral windup. Thus, there is consistently minimal settle time with no overshoot.

3.4 Angular Control

Even with perfectly functioning horizontal and vertical controllers that describe the desired motion, the quadrocopter only has the ability to apply thrust in its current direction and to exert torque on itself for rotation. For example, it is physically impossible to exert force horizontally while the propellers are pointed vertically, as shown in Figure 3.11. So, the current angle of the craft needs to be controlled as well by applying torque. Torque is created when the thrust applied by the propellers on one side of the craft is greater than the thrust applied on the other side.

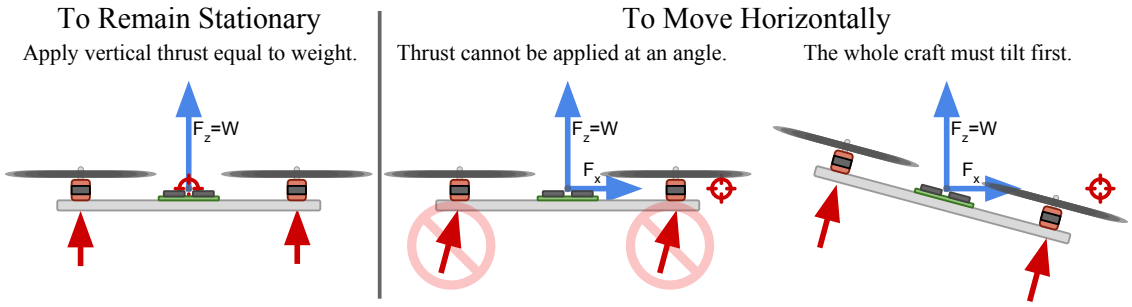


Figure 3.11: Rotation is required for horizontal quadrocopter motion.

To demonstrate the issue, consider Figure 3.11. To remain stationary in the air, thrust must be applied to counter the gravitational force, which is an entirely vertical force. Therefore, the propellers must be pointed upwards while stationary. If the target were to move to the right, the desired force could not be immediately applied because there would be no way to apply thrust horizontally. To accelerate to the right, the craft would first need to lean towards the right while maintaining

the vertical component of the thrust equivalent to the force of gravity (see Eq. 3.5). Since the tilt causes acceleration, as shown in Eq (3.3), the craft must tilt towards the left to reduce speed before it gets to the target. This type of anticipation is handled by the derivative component of both the horizontal and vertical controllers.

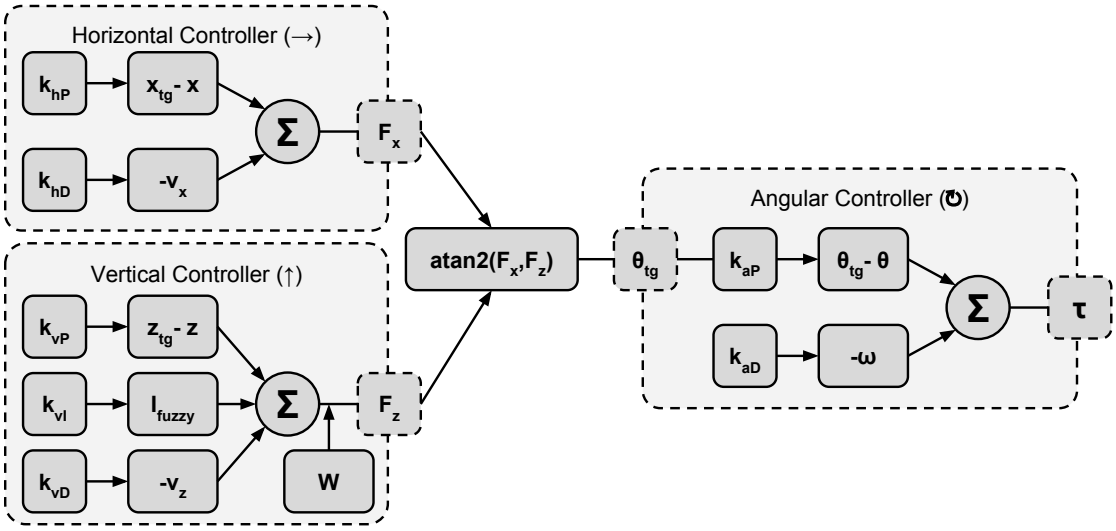


Figure 3.12: Partial block diagram for the angular controller.

An ideal controller, as shown in Figure 3.12, is robust and stable for a wide range of angles so that it can perform well in extreme situations. For example, if the craft is flipped due to collision, a robust controller could attempt to right the craft. If a craft needs very rapid horizontal motion, it can turn nearly sideways in order to move even faster. Similarly, if desired, the craft can forcefully descend faster than gravity by flipping upside down.

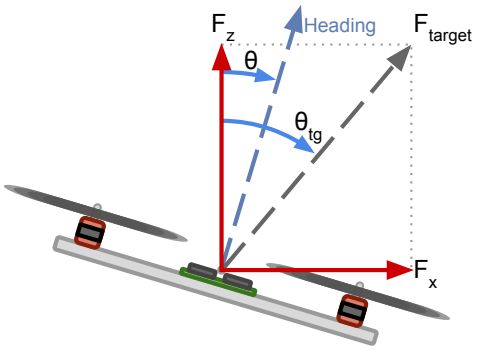


Figure 3.13: Angular control variables.

To achieve the desired horizontal and vertical motion, F_x and F_z are used, which are already designed as shown in Eq. (3.15) and Eq. (3.16). These desired resultant forces form a vector that then describes the desired angle given by Eq. (3.25) and as shown in Figure 3.13. By using the `atan2` function on the two component forces, the quadrant of the desired angle can be retained, thereby allowing the craft to even flip upside down if the vertical controller happens to tell it to. The design for the angular controller is then a simple PD control as shown in Eq. (3.26).

$$\theta_{tg} = \text{atan2}(F_x, F_y) \tag{3.25}$$

$$\tau = k_{a_P}(\theta_{tg} - \theta) - k_{a_D}\omega \tag{3.26}$$

The main challenge in optimizing the new angular controller is deciding what total thrust to use. In the majority of cases, θ is close enough to θ_{tg} that $F_{target} \approx F_{current}$ because the horizontal and vertical controllers tend to change continuously, so the angle input changes continuously. However, when the target changes drastically, the two angles may be nowhere near each other. For example, if the quadrocopter is stationary and needs to move horizontally, the F_{target} force will be the combination of the desired F_z and F_x forces, even though it is only currently able to apply force in the z direction (Figure 3.14a). If the F_{target} force is exerted, the quadrocopter would then drift above its previously established vertical position until the angle is near zero again. In another example, as shown in Figure 3.14b, if the current orientation and the desired orientation are at a right angle, then no total thrust can be applied. However, torque can still be applied without changing the total thrust.

In a more extreme example, if the craft is pointed up, but the new angular target tells the craft to point down (Figure 3.14c), simply using F_{target} would cause the craft to apply upward thrust while attempting to flip. Instead, it should try to pull itself downward while flipping.

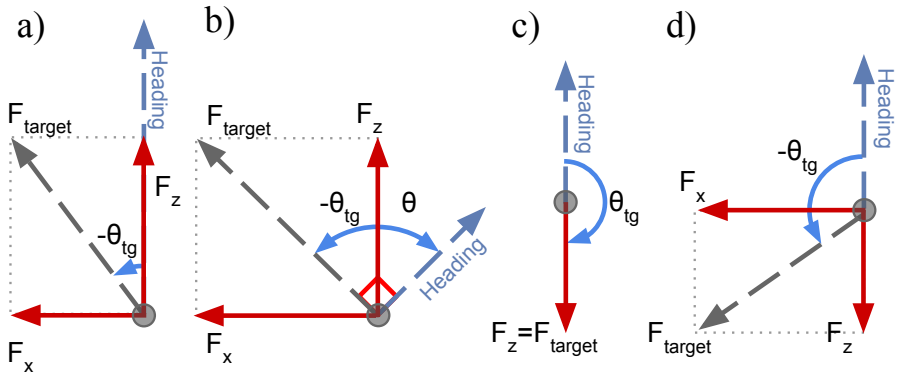


Figure 3.14: Extreme configurations where total thrust needs to be chosen carefully.

Since the craft at its current orientation may be unable to fully exert the desired force, the total thrust output should be modified by projecting the F_{target} vector onto the current heading vector like a dot product, which is most easily seen in Figure 3.14d. The projection can be achieved using the cosine of the angle error. The modification solves the problems in the examples in Figure 3.14 whereby simply using the magnitude of F_{target} would produce undesirable results. If the angle error is close to zero, then nearly the full target force is applied. If the force should be applied in the direction opposite to the current orientation, then negative thrust is used.

$$||\vec{F}_{target}|| = \sqrt{F_z^2 + F_x^2} \tag{3.27}$$

$$F_{output} := ||\vec{F}_{target}|| \cdot \cos(\theta_{tg} - \theta) \tag{3.28}$$

To combine all of the controllers, use Newton's Second Law to apply the total force and torque in terms of the thrust applied by each propeller:

$$\Sigma F = F_{output} = T_L + T_R \tag{3.29}$$

$$\Sigma \tau = \tau = (T_L \times 1) - (T_R \times 1) \tag{3.30}$$

$$T_L = \frac{1}{2}(F_{output} + \tau) \tag{3.31}$$

$$T_R = \frac{1}{2}(F_{output} - \tau) \tag{3.32}$$

In conclusion, the individual thrusts applied by the motors are controlled by the total thrust and the torque applied on the craft. The torque is the difference between the left and right thrusts. The total thrust is dictated in Eq. (3.28) by a combination of the horizontal controller from Eq. (3.15) and vertical controller from Eq. (3.16). The torque is dictated by the angular controller in Eq. (3.26), which uses the horizontal and vertical controllers as inputs to decide a target angle. The total thrust might be negative, or the torque might cause just one propeller to exert negative thrust. A real quadrocopter typically cannot reverse the direction the propeller is spinning, often due to electrical limitations. So, a real quadrocopter might not be able to exert negative thrust. Additionally, a motor has a maximum limit to how much thrust it can exert. These issues are unavoidable regardless of the controller used, so gains and constraints should be designed to avoid these limits. The controller can now guide the craft appropriately to externally defined target positions, but the target positions must be dictated in such a way for the quadrocopter to sort colored blocks.

3.5 Color Grouping Using Artificial Neural Network

An artificial neural network (ANN) can adaptively learn how to sort blocks based on color. An ANN is similar to polynomial regression fitting. By first finding a fitting function of a data set of inputs and outputs, additional outputs can be inferred when there are new inputs. The fitting process is known as training or learning. An ANN, however, is capable of fitting multiple inputs to multiple outputs (MIMO), unlike a polynomial fitting function which is single input and single output (SISO). Since an ANN is MIMO, it has a wider variety of applications, such as grouping or pattern recognition. Theoretically, a sufficiently complex ANN might be able to emulate any relationship function with adequate training.

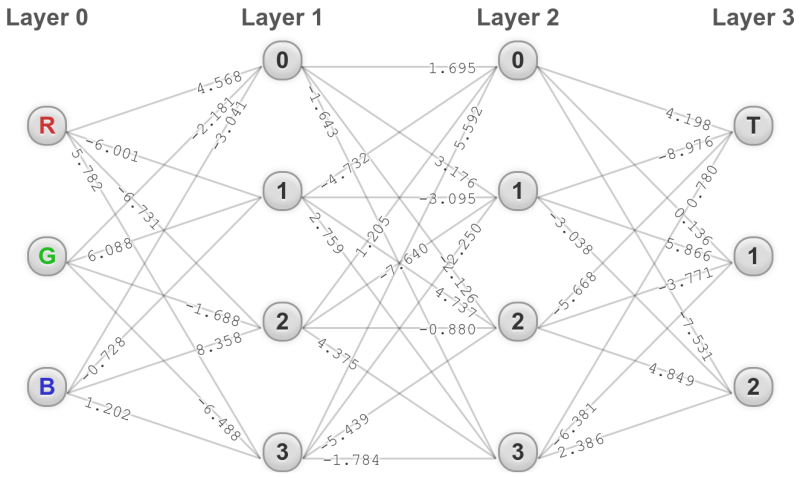


Figure 3.15: A trained artificial neural network for grouping.

Essentially, an artificial neural network consists of several layers of nodes (Figure 3.15). Many of the nodes are interwoven by weighted relationships. The ANN weights (w) start out with random values. Training a neural network is essentially a repetitive process of "guess and check," where the weights are tuned to more closely match the training data.

3.5.1 Mathematics

Before a generic ANN can be built, how it works must first be described. To find the outputs, each node, except for the first layer, has a value that is a function of the weighted sum of its parent nodes, where the weight is the weight of the relationship connecting the two nodes (Figure 3.16) [11, pg. 180]. The nodes in the first layer are the inputs and have their values defined accordingly.

$$s = \sum_i w_i O_i \quad , \quad O = \frac{1}{1 + e^{-s}} \tag{3.33}$$

Where:
- w_i is the weight of a previous relationship
- O is the current node
- O_i is the previous node in the w_i direction

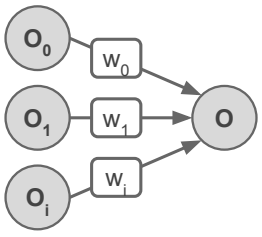


Figure 3.16: Node variables.

To train a neural network, first, the output is found through the method described in Eq. (3.33). Then the error is calculated on the final outputs by comparing them to the known data. Then the error is distributed, starting at the output and going towards the input, in a process called back propagation [11, pg. 187].

$$E_{output} = O_{actual} - O \tag{3.34}$$

$$E = O(1 - O) \sum_i w_i E_i \tag{3.35}$$

Where:
- w_i is the weight of a next relationship
- O is the value of the current node
- E is the error at the current node
- E_i is the error in next node in the w_i direction

The second phase of a training iteration for a neural network is to apply the error to each of the relation weights. Reducing the influence that the error from a single

sample has on a weight can help balance the neural net faster, but reducing it too much can make it take longer to compensate.

$$w_{new} = w_{old} + \alpha EO \tag{3.36}$$

Where:
- w is the weight of a
- O is the value of its input node
- E is the error at its input node
- α is a learning constant. Ex: 0.2 or 0.3

Finally, to fully train an artificial neural network, a training iteration is performed for each sample input and output. The samples used to train must be large and diverse enough to be a representative sample of all possible inputs and outputs. Also, each sample should be trained multiple times, since each training iteration will throw off the tuning from the other samples. Each sample could be trained a predetermined number of times, but the number of times an ANN needs to be trained will vary based on its shape, the random values the weights are initialized with, and even the training samples. By keeping track of the errors in each iteration, the training process can be stopped whenever the error is minimized. However, it is not guaranteed that the error will disappear completely. A more complex neural net might be required to accurately represent the relationship between input and output or conflicting data may be in the training data set. So, it is best to stop training whenever either an iteration limit is reached or the error reaches a sufficient minimum.

It may be important to test the neural net against a different set of data and outputs to validate the training. It might be possible that the neural net could be tuned for the training data, but not all possible data if the training data does not sufficiently represent the system. However, in practice it is often best to give the ANN the additional training data so it can have a more complete picture instead of

withholding data for a validation phase. In this color sorting example, it is critical that all of the required color group associations be trained on the neural network. The interpolation between these critical color groupings is, to some extent, subjective. Some methods have been developed to rotate the training and validation data, but they are unnecessary for this usage.

3.5.2 General ANN Implementation

To build a generic implementation of an ANN, objects should be used. By observing the index notation used in the mathematics behind ANNs it might be tempting to use a three-dimensional array to store the w variable for each link. Further observation shows that the O and E variables are associated with the nodes, though, which could be represented with two-dimensional arrays. Such a data structure might be efficient, but it is not flexible. It requires that the number of nodes in each layer be the same and that each node has links to each of the nodes in the next layer - neither of which are valid assumptions for all neural networks. In fact, there are some ANN designs where links can skip layers. Object-oriented programming allows the creation of more complex data structures which can be much more flexible. By using dynamically resizable arrays, a node object, and a link object (Figure 3.17), any conceivable neural net layout can be implemented easily. It can represent

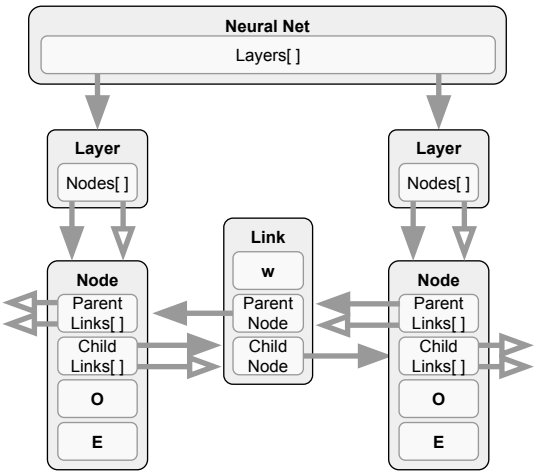


Figure 3.17: ANN data structure.

networks that have differently sized intermediate layers, or even networks that have nodes that can bypass layers. Once the data structures are built and the methods are implemented, the traversal patterns used in the mathematics are automatically followed for any graph configuration.

It is still useful to have a two-dimensional array to store the nodes in order to perform traversals. While it would be possible to traverse the entire graph without them, it avoids traversing the same node twice. The advantage to having the link data type is that it can be added and removed between nodes regardless of layer. So, the functions to perform operations become universal regardless of size or link configurations, like the output calculations from Eq. (3.33):

```
1 for (var l=1; l<this.layers.length; l++) {
2     for(var n=0; n<this.layers[l].length; n++) {
3       // each node in each layer
4         var node = this.layers[l][n];
5
6         //calculate sum
7         var sum = 0;
8         // all preceeding nodes
9         for (var p=0; p<node.parents.length; p++) {
10            sum -= node.parents[p].w * node.parents[p].p.O;
11        }
12
13        node.O = 1 / ( 1 + Math.exp( sum ) );
14    }
15 }
```

3.5.3 Color Grouping Input and Output Design

To use a general ANN to sort colors into groups, the inputs and outputs have to be established. Computers generally represent colors with three channels (red, green, and blue), so the input to the neural network will have three dimensions. A

high number indicates that the color is bright, and low means dark. White would be the maximum of each channel, and black would be the minimum. To interface the colors with the ANN, each input and output is normalized to have a minimum and maximum of 0 and 1 respectively. The number of outputs will depend on the number of groups that should be made, which in this example is three. The output node values represent the strength of the input matching a group. The group is chosen from the output by choosing the node with the maximum value.

Tables 3.2 and 3.3 show the neural net converting color channel inputs to group outputs. Table 3.2 shows a pink color firmly belonging in the group with reds and pinks. Conversely, Table 3.3 shows a dark green being close to belonging in either a group with black or a group with green, but slightly closer to belonging in the black group. If this behavior were not desirable, a dark green block could be added to the green group before training. It is also possible to retrain the neural net by correctly placing the misplaced element and then applying additional training iterations to the existing neural net.

Table 3.2: Grouping a Pink Input

Input	Output
R - .9608	.0085 - blue, white, black
G - .4275	**.9929** - red, orange, pink
B - .5059	.0011 - green

Table 3.3: Grouping a Dark Green Input

Input	Output
R - .1273	**.4969** - blue, white, black
G - .2232	.0015 - red, orange, pink
B - .1388	.4672 - green

3.5.4 Shape vs. Efficiency and Accuracy Experimentation

To sort the colors efficiently and accurately, a properly shaped ANN must be used. To find the best shape for the ANN to sort colors, two experiments were

performed. First, the training efficiency was tested, then the accuracy of the trained shape was tested.

To test the training efficiency, 18 ANNs were randomly generated for each shape. The number of training iterations required to achieve 5% error from placing each of the samples (Figure 3.15) completely in their respective groups was measured and averaged (Table 3.4). Instances where the ANN never eliminated the error completely were thrown out, rebuilt, and retested. Shapes where this occurred are denoted by an asterisk (*) because the true average work done would have been much higher if they were included. Failure rates were later measured in a separate experiment. ANNs with breadth of two were also tested, but none were successfully trained.

Table 3.4: Training Loops to Reach 0.05 Error for Differently Shaped ANNs

n=18	Breadth: 3	Breadth: 4	Breadth: 5	Breadth: 6
Depth: 1	1145	833	669	651
Depth: 2	373	263	264	212
Depth: 3	1494*	1274*	1154*	1453

Less training iterations does not necessarily mean that it requires less computing power to train the neural network. The number of links in the neural network increases the amount of work a single training iteration requires. The equation describing the number of links in the ANN is:

$$L = 6B + (D-1)B^2 \tag{3.37}$$

Where: L is the number of links in the ANN
B is the breadth of the ANN
D is the depth or layers in the ANN, not including input or output

Multiplying the number of links in the ANN by the number of loops required to train the ANN gives the number of times a link was adjusted and approximately the amount of work required to train the ANN (Table 3.5).

Table 3.5: Work to Train ANN

n=18	Breadth: 3	Breadth: 4	Breadth: 5	Breadth: 6
Depth: 1	20,606	19,993	20,064	23,425
Depth: 2	10,075	10,509	14,503	15,275
Depth: 3	53,797*	71,347*	92,312*	156,901

It can be seen in Table 3.5 that 3x2 and 4x2 ANNs require the least amount of work. More generally, a depth of two is shown to be the most efficient regardless of breadth. Really, the difference between the depth of two and even the depth of one is somewhat negligible, so accuracy should be used to decide which one is the best.

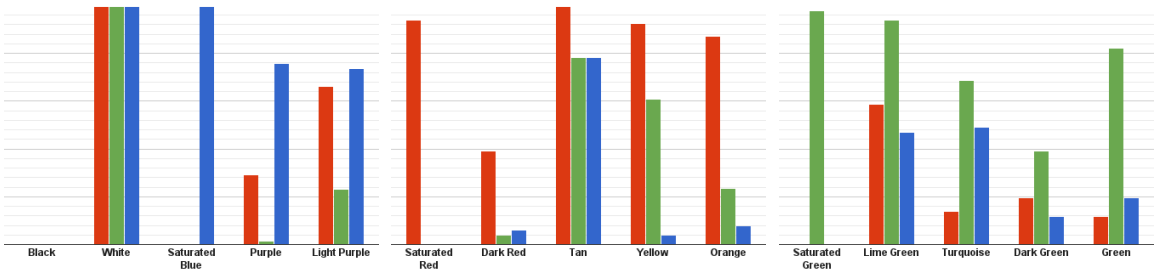
In some cases, if the randomly generated ANN would never converge, then these were not included in the average. Instead, a separate set of ANNs was randomly generated and trained and the number of failures to converge were recorded (Table 3.6).

Table 3.6: Failures to Converge to 0.05 Error

n=22	B:3,D:3	B:4,D:3	B:5,D:3	B:6,D:3
Failures:	11	4	3	0

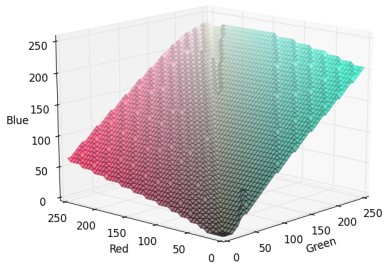
To test the ability of the ANN shape to accurately group colors, a single sample of that shape was trained (Figure 3.18). Then an evenly spaced grid of red, green, and blue inputs was fed into the controller to essentially form a three dimensional coordinate system with each point on the coordinate system having a group value. To generate a 3D surface graph of this data, a loop went from the bottom of each

blue column and found the top-most transition between groups. The color that is shown on the graphs in Figure 3.19 is therefore the color on the boundary and nearly indecisively between two groups. Unfortunately, other potential group transitions were not captured due to the constraint of the graphing surface.

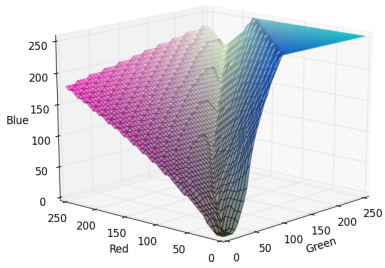


(a) Group 0 (mostly blue) (b) Group 1 (mostly red) (c) Group 2 (mostly green)

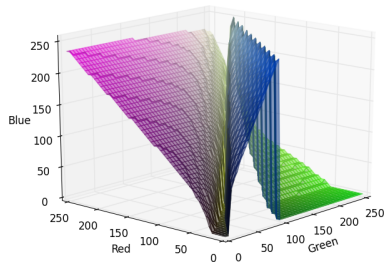
Figure 3.18: Grouping data samples used to train the ANN in experiments.



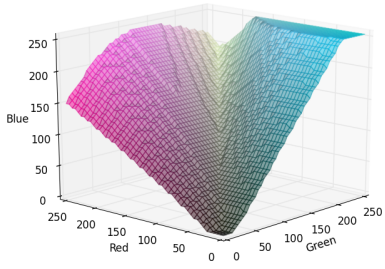
(a) Breadth: 3, Depth: 1



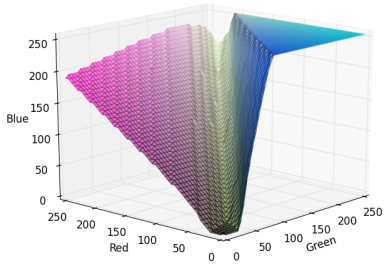
(b) Breadth: 3, Depth: 2



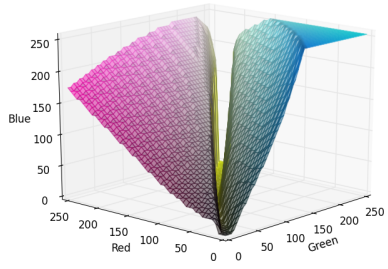
(c) Breadth: 3, Depth: 3



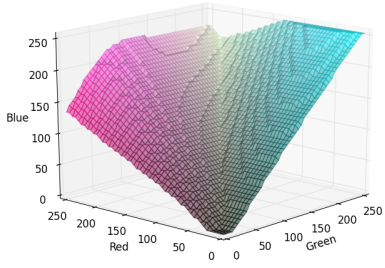
(d) Breadth: 4, Depth: 1



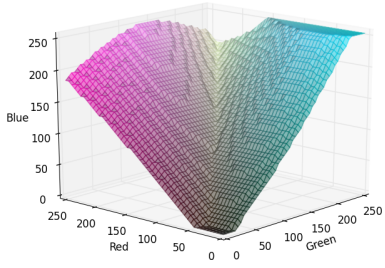
(e) Breadth: 4, Depth: 2



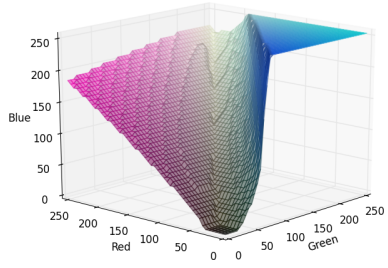
(f) Breadth: 4, Depth: 3



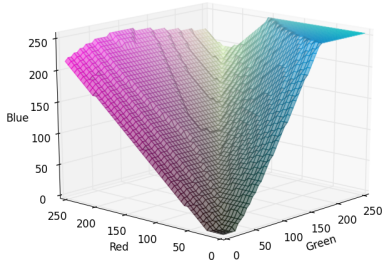
(g) Breadth: 5, Depth: 1



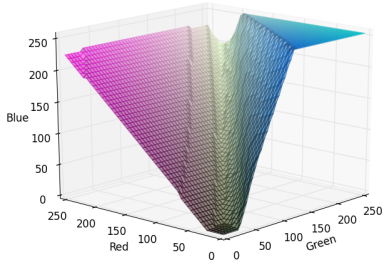
(h) Breadth: 5, Depth: 2



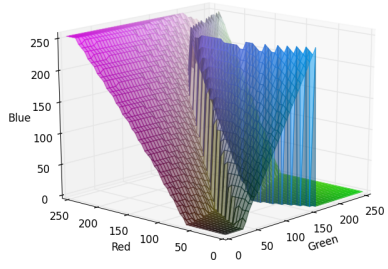
(i) Breadth: 5, Depth: 3



(j) Breadth: 6, Depth: 1



(k) Breadth: 6, Depth: 2



(l) Breadth: 6, Depth: 3

Figure 3.19: Valid grouping boundaries (Figure 3.18) for various ANN shapes.

The results of the accuracy experiment are somewhat subjective, since the desirable grouping pattern is also subjective. However, the grid of graphs in Figure 3.18 shows an important trend. The more complex a neural network is, the more it is able to contour to more complex shapes. For example, the 3x1 ANN (Figure 3.19a) created essentially a plane to divide the groups. In comparison, the 4x2 ANN in Figure 3.19e shows that the mostly blue group has claimed a small valley of yellow. It should be noted that, since these are each only a single sample of a trained neural net for a relatively small set of data, the boundaries could easily look very different depending on the initially random link weights. One example of this is the 5x3 ANN in Figure 3.19i, which does not look very different from the relatively simple 6x1 ANN (Figure 3.19j). Since only the top-most boundaries are shown, when the top boundary goes over the edge on the 3x3 and 6x3 (Figure 3.19c and 3.19l), the boundary beneath it is revealed. It should be assumed that there are similar boundaries on the other graphs separating the mostly red and mostly green sections. Due to the way the boundary is selected, the boundary between either red and blue or green and blue is generally favored in the visualization.

3.6 Combining Controller and ANN Using Path Planning

Since the controller and neural network are tuned to function optimally in this system, getting the two to work together to solve the problem is actually extremely simple. The only missing element is path planning, which is the changing of the targets of the controller over time. Path planning tells the controller where to go, but the controller decides how to get there. So the path planner simply needs to utilize the ANN's ability to group colors to decide which group to go to.

To integrate the artificial neural network with the quadrocopter controller, an additional object is created called Autopilot (Figure 3.2). Autopilot is responsible for feeding the block colors to the ANN and using its output to decide which landing platform to go to. To prevent collision with landing platforms, an additional landing and takeoff process must be added between each trip. The resulting path planning procedure generates changes in target position that are jarring and discontinuous, which might not be desirable. It is assumed, however, that the controller should be able to handle the discontinuous changes. So, while ideally the path planner should provide smooth changes in target parameters, it is not necessary. The resulting procedure for the path planning is:

1. `"home"`: Approach the Incoming platform to get a new block by setting the target position at a safe cruising altitude directly above the platform.
2. `"land"`: Descend towards the platform by setting the target to the same horizontal position, but at an altitude close enough to the block to pick it up.
3. `"takeoff"`: Ascend from the platform by setting the target to the same horizontal position, but at a safe cruising altitude far enough that a tilt will not hit a platform.
4. `"sort"`: Use the ANN to decide which platform the block should be sorted to. Set the target at a safe cruising altitude directly above that platform.
5. `"land"`: Same as 2
6. `"takeoff"`: Same as 3

The final result of the project is a webpage, shown in Figure 3.1, which features a two-dimensional quadrocopter that can sort blocks based on color. There are two panels that can change settings for the ANN, controllers, or the simulation environment.

Since the simulation is built with web standards, the code is available by viewing the page source from any browser running it. The simulation is available online at:

`http://robotic-controls.com/static/control-demo/`

By appending `#autostart` to the address, a sorting demo will automatically be trained and autopilot will begin as soon as the page is loaded with no interaction required. Alternatively, clicking the "Build Example" button does the same thing as the address parameter. To set up a custom grouping, the procedure is:

1. Click and drag blocks to grouping platform.
2. Configure neural net parameters, like breadth, layers, and the maximum error and iterations.
3. Click the "Train" button. If the "Loops Ran" is the same as the "Max. Loops," the error has not been reduced enough and more training is needed or, more likely, the ANN shape is incapable of grouping the pattern of colors given. A more complex ANN or less complex color pattern should be used.
4. Check "Sorting Autopilot" for the quadrocopter to begin sorting blocks.

Once the sorting has begun, no interaction is necessary. However, by clicking anywhere in the simulation area, a user can change the target position used by the copter controller manually. The target can be changed regardless of whether sorting autopilot is enabled or not.

Additionally, the gain constants for the controller may be modified in the top right corner. The default gains were chosen by experimentation and there is likely room for performance improvement by further refining them. The cargo weight is also adjustable, so the effect of the unknown weights and integral gains can be more thoroughly explored.

Alternatively, a user can attempt to control the thrusts of the quadrocopter manually. The quadrocopter is extremely difficult to control, partially due to the virtual input interface. A physical thumbstick or joystick would likely provide a substantially easier way to control the craft because of its tactile nature. In manual mode, the virtual thumbstick is at rest at the bottom center where no thrust or toque is applied. Pushing the virtual thumbstick up will increase the thrust, and pushing it to either side will cause it to exert a torque in the same direction.

Force arrows, similar to those used in free body diagrams, are shown on the craft by default. This is to help visually expose the underlying physics in the simulation. It can help with learning about controls, but it also can expose problems when designing the controller. For example, the arrows change color when the thrust is negative, which is often impossible in practice.

On modern hardware with a modern browser like Google Chrome or Firefox, the simulation can simulate up to 50-100 frames per second. With less modern hardware, or in a browser like Internet Explorer 9, the frame rate is still around 25 frames per second. Even a tablet like the iPad 2 can perform 10 frames per second. Unfortunately, the simulation needs a time step (Δt) slightly smaller than 0.1 seconds to accurately represent the quadrocopter enough for the controller to be stable. Also, most browsers limit how often a background tab can run the `setInterval` event to once every second to improve performance for the focused tab, so the simulation falls apart if it is run in a tab that is not focused. Similarly, when in the foreground, a browser may only fire the `setInterval` event every 4 to 15 milliseconds, so there is actually an upper limit to the frame rate too. To prevent excessive hardware usage, an adjustable frame rate limit is imposed at 60 frames per second. Also, the simulation may be paused by either checking the paused box or by pressing the spacebar.

3.7 Sorting Simulation Conclusion

To build a simulation from scratch that simulates a two dimensional quadrocopter that has the ability to sort blocks, the components were compartmentalized and completed individually. First, a functioning physics simulator was built. To do so, the dynamics of the system were defined, then the dynamics equations were solved numerically using the fourth-order Runge-Kutta differential equation solution method to reduce error and increase efficiency. At this point, the quadrocopter that would just fall unless manual motor control were used. It needs to know how to move. A series of PD and PID controllers were used to give the quadrocopter the ability to move automatically and maintain its position despite gravity. A unique approach was used that allows the craft to be stable at any angle through the use of an angular controller. The controller was further enhanced by using fuzzy logic principles to refine the ability to compensate for cargo mass in the vertical controller. By then, the quadrocopter could hold a position or go to a destination, but it did not know how to group blocks. Grouping was accomplished through the use of an artificial neural network, which was constructed in a flexible way by taking advantage of object-oriented programming. Then the neural network was interfaced to the controller through simple path planning. With this addition, the quadrocopter is finally able to sort blocks based on color. By applying the principles explored in this simulation, many things can be achieved. Other browser-based simulations could be built, real quadrocopters can be controlled, and grouping based on any kind of measurable attribute can be performed.

CHAPTER 4

INDOOR ABSOLUTE POSITIONING WITH BLUETOOTH AND ANN

Indoor radio-based positioning has several potential uses, that currently cannot be filled. In particular, autonomous robots need to know their position accurately. Another potential use, however, might be to find the exact position of a person in a building. Human positioning could be based on a cell phone and could have applications such as home automation or gaming. There are not currently solutions that are readily available and accurate enough for either task. Consumer-grade GPS devices in the United States (or SPS devices) are inadequate for indoor positioning. Outdoors, they are accurate to 307 inches (7.8 meters) on a 95% confidence interval [23]. So, if consumer GPS devices worked indoors, they might not even be accurate enough to reliably differentiate between rooms. However, most consumer devices cannot acquire enough satellite signals indoors to work at all. Many modern cell phones work around this issue by augmenting the GPS positioning with position data from nearby IEEE 802.11 WiFi access points and cellular towers. This technology is called aGPS, or assisted GPS, which was developed to give 911 emergency dispatchers location information. The SSID and MAC address of the router are taken from the beacon frame and checked against a database of known router positions. The beacon frame is typically sent on an interval of around 100 milliseconds. However, the received signal strength indication (RSSI) for WiFi is known to be very prone to

noise, so a higher sample interval might be desirable so that some signal processing or averaging can be done to improve accuracy.

In this paper, the Bluetooth Low Energy (Bluetooth LE or BLE) subset of the Bluetooth 4.0 specification is used. Receivers are readily available in smartphones and USB dongles. The transmitters, which can be used as beacons are cheap, require little energy, and can be connected to quickly. The RSSI is then processed and put into a multilayer perceptron, a type of artificial neural network, so that the signal strength at certain locations can be learned and remembered.

4.0.1 Bluetooth LE

Bluetooth LE operates on the typical 2400 to 2484 MHz range, much like WiFi and other common short-range radio devices. There are two major kinds of Bluetooth devices. Bluetooth LE devices have 38 2MHz-wide channels that can be used for communication. The frequencies have wavelengths between 4.7525 to 4.918 inches. The other kind of device is often referred to as classic Bluetooth because it does not support the new LE specification added in version 4.0 of the Bluetooth standard. Both LE and classic devices operate in the same range of frequencies, but each has a different number of channels within the range. Both types of devices also feature adaptive frequency hopping which automatically avoids noisy channels. Due to this feature, the channel cannot be specified by the host computer [24].

There are several advantages to using a BLE device rather than a classic Bluetooth device. BLE uses 10 to 20 times less power than a classic Bluetooth device [25]. In addition to the significantly lower power consumption, a BLE device can connect and send data in under 6 ms, whereas classic Bluetooth devices can take up to 100

ms. The tradeoffs for the faster connection time and lower power consumption are decreased throughput and decreased range. The throughput, for the purpose of positioning, is unimportant since the devices are just pinged for signal strengths. The decreased range could actually be beneficial since the resolution and accuracy in the RSSI measurement is limited. If one unit of RSSI correlates to less distance, the distance measurement should be more accurate. However, more beacons would be required in order to achieve full coverage. The cost of the additional beacons to compensate for their limited range should be minimized by the reduced cost of BLE technology. BLE chipsets are cheaper compared to classic Bluetooth devices. BLE devices also consume less power, which allows the use of smaller and cheaper batteries and reduces the recurring cost of power consumption.

4.0.2 Trilateration

Trilateration is a process often confused with triangulation. Triangulation uses angles from two known positions to compute an unknown position. Trilateration uses the distance from two or more known positions to compute an unknown position. In 2D space, there are two possible locations for a pair of distances from known positions because two intersecting circles will intersect at two points. A third known distance and position is required in order to solve for just one position to eliminate one of the two possible positions. However, there is typically error in the distance measurement. To reduce the error, the combination of more than three known distances and positions can be used (Figure 4.1). The mathematics

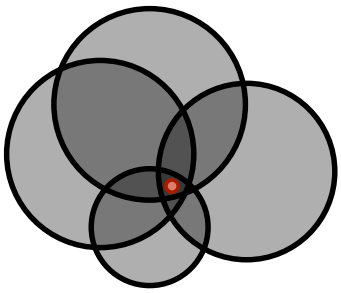


Figure 4.1: Error correction.

to calculate the position is made somewhat irrelevant because the signal strength drop-off of BLE devices over distance is highly irregular and because a neural network is used instead of trilateration.

4.1 Getting BLE Signal Strengths

Due to the fact that Bluetooth 4.0 is a relatively new specification, there are not an abundance of easy-to-use software libraries available. One easy way to interface with a USB Bluetooth 4.0 adapter from a Linux-based computer is the Noble [26] library for Node.js. The basic procedure to collect the signal strengths is to first scan for all LE-enabled Bluetooth devices. A device can respond to this sort of request up to approximately five times per second. The response of the device also has an RSSI, so basic positioning could be done without even connecting. This could be useful, particularly for a device that is slightly out of range, since a device is more likely to respond to a scan than to establish a connection from a long distance. Before connecting, scanning must be stopped. Once connected, the updateRSSI function can be called repeatedly. With this method, the RSSI can be updated up to 1000 times per second. Multiple connections are not necessarily possible under Bluetooth Low Energy, so to use the updateRSSI function, the active connection must cycle through the available BLE devices.

4.2 1D Experiment

Before attempting positioning, first the basic relationship between RSSI and distance for a BLE device should be understood. To map this relationship, an

experiment was conducted where a Kensington Proximo K39567US was placed at incrementally increasing distances from a Bluetooth USB receiver. The update RSSI function was used at a maximum rate of about 500 times per second until 50,000 samples was reached. The full experiment results are shown in Figure 4.2.

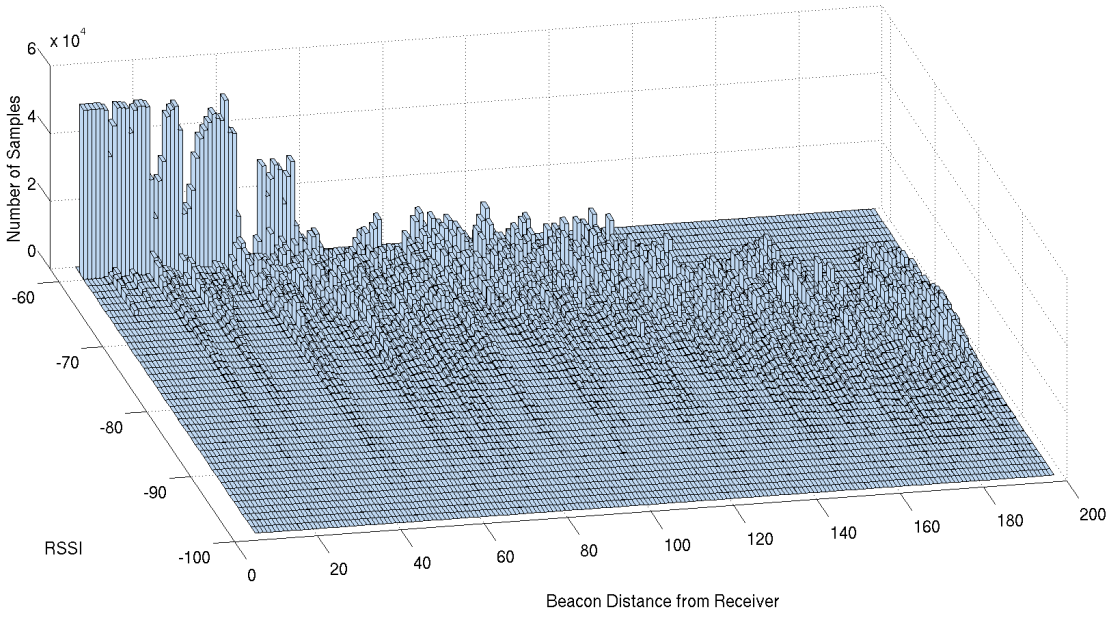


Figure 4.2: 3D histogram of RSSI samples at distances.

The experiment results show that signal strength measurement is significantly noisy. However, the signal strength does tend to decrease as distance increases, as expected. The close distances from 0 to 40 inches reveals important information. There is a significant drop-off in signal strength every four or five inches, probably correlating to the wavelength of Bluetooth at 2.4 GHz, which is about 4.8 inches. During this test, there was another BLE device at a fixed position off to the side about 60 inches away, while the Kensington Proximo was moved (Figure 4.3). It was discovered after the

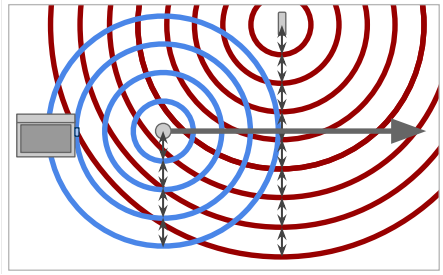


Figure 4.3: Interdevice interference.

test that these dropoffs disappeared when the second BLE device was not present and reappeared when it was brought back. While the test could be repeated to avoid this unexpected source of error, not much of importance would be revealed. Since positioning based on BLE devices will require multiple BLE devices to be within range, cross-device interference will have to be dealt with. As long as the dropoffs are repeatable, a neural network should be able to compensate.

Several methods of processing the RSSI into more manageable components are shown in Figure 4.4. The mean and standard deviation are probably the most interesting. However, the minimum and maximum could potentially be useful as well. The median is not significantly different from the mean in this case but may be less accurate since it is restricted to whole numbers (as RSSI is) as opposed to the mean which is not restricted. By visual inspection, the standard deviation may not contain different information than the mean or minimum, but just inverted.

Also on the top plot in Figure 4.4 are multiples of the minimum and maximum wavelength, respectively. While the offset of these multiples may not be accurate, it does reveal several potential effects. First, the dropoffs are more drastic when the Kensington is close to the receiver, and likely most severe when the wavelength correlates exactly on a distance that was measured. Second, the maximum signal strength drops a step almost exactly when the distance between the minimum and maximum wavelength multiples is zero. In other words, after about 30 wavelengths, the difference between the minimum and maximum wavelength (about .15 inches) becomes about the wavelength 4.8 inches. Therefore, the adaptive channel hopping can avoid any phase cancellation at long-enough distances. However, the adaptive channel hopping is somewhat random, so when this happens the noise in the signal strength will increase.

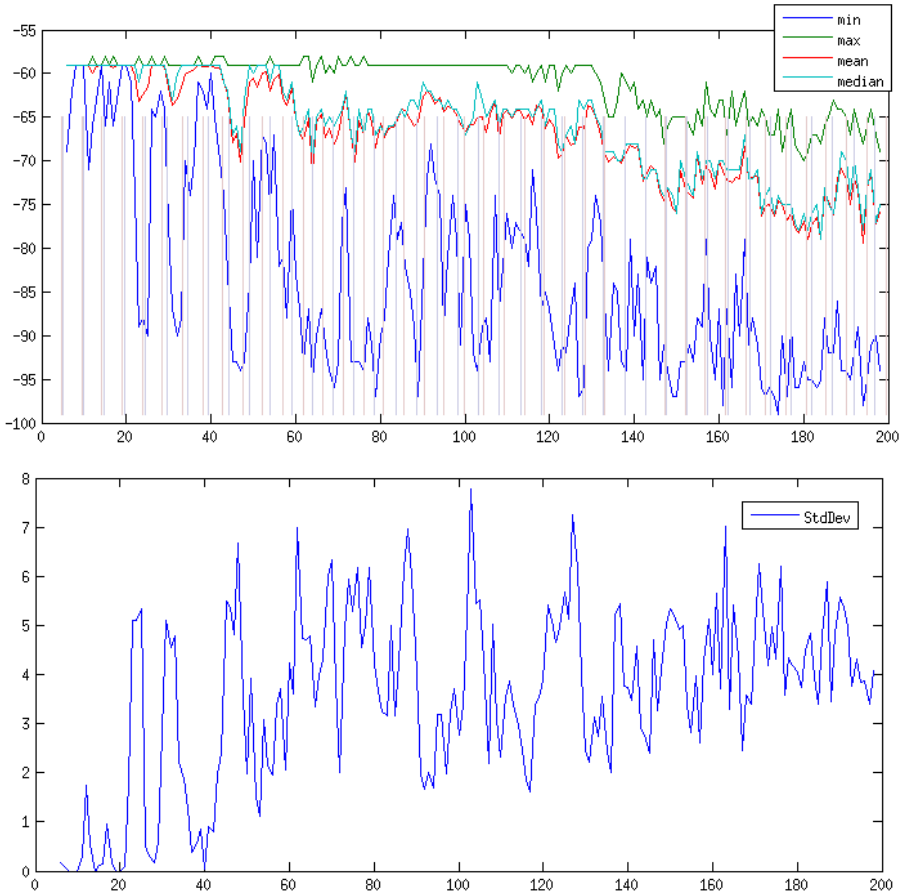


Figure 4.4: Plots RSSI samples after simple statistical post-processing.

Knowing that the phase cancellation and wavelength are so important to signal strength, a better solution to find the distance might be to measure the signal strength at each of the 48 channels that the BLE devices can communicate on. However, due to the adaptive channel hopping, which would generally be a useful feature, it does not seem to be possible to override a channel. It is worth noting that the scan responses occur on one of several channels different from the channels used to communicate on. Again, there does not seem to be a way to force scanning to take place on a channel. Extracting the communication channel from the receiver might be possible, but there seems to be no obvious software method for finding this information. Identification of communication channels is not generally necessary

for simple Bluetooth serial communication, which is the typical usage. Although the channel is selected adaptively, the channel selected is not guaranteed to be the strongest of those available, so the information may not be very useful information.

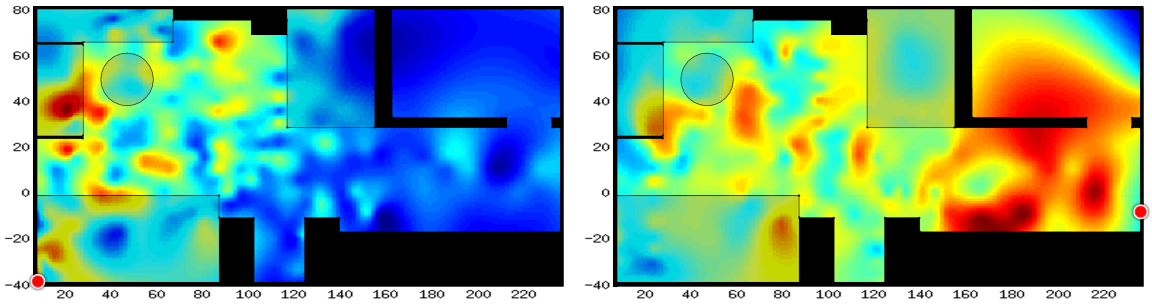
4.3 2D Experiment

Fitbit One, Kensington Proximo, USB
Figure 4.5: Devices used.

Now that the behavior of a single BLE signal is somewhat understood, a test for 2D positioning is performed. The procedure for this 2D experiment is different from the 1D one. Instead of moving the beacon, which was easier, the receiver is moved. Also, the number of samples at each position is substantially less - from 50,000 to 4,000. In addition to the Kensington Proximo K39567US from the previous experiment, a Fitbit One is also used - both are shown in Figure 4.5. Both beacons have extremely similar transmission strengths. Both beacons are placed approximately 55 inches above the floor on opposite ends of the room. Then a laptop with the BLE dongle is placed at varying coordinates on the floor and the signal strength is measured. The samples are taken at uneven spacing, but most are on a 5-inch grid. To ensure repeatability, the laptop is not rotated between tests to ensure that the unrecorded direction of the small antenna in the USB dongle is not introducing seemingly random errors to the data. The significance of the orientation of the small antenna is unknown. If it is found to be important, the angle will somehow have to be taken into account - perhaps by recording the

angle and adding the value as a node to the neural network. However, this would require significantly more training.

Figure 4.6 shows the data from the experiment plotted on a heatmap in MATLAB using 'v4' interpolation due to the uneven spacing of the samples. Then a map of the room and the layout of some of the large furniture is overlaid onto the heatmap. Solid objects like walls or bookcases are shown in black, and furniture where samples were taken under are shown with an outline and a slightly darker translucent fill. The left map belongs to the Fitbit One, and the right map belongs to the Kensington Proximo. The strongest signal is shown in dark red; the weakest is shown in dark blue.



(a) Fitbit One (b) Kensington Proximo

Figure 4.6: Heatmap showing the RSSI throughout room.

Both heatmaps show one important reason why an analytic approach to trilateration cannot work indoors. Due to various effects like attenuation and reflection from obstacles, the strongest signal measured for each beacon was a full 60 to 80 inches away from the respective beacon. Additionally, the signal strength does not form a circle around the beacon. Instead, there is a clear wave pattern that is distorted by the obstacles in the room like walls and furniture. As a result, there can be many positions with high signal strength far away depending on how the furniture and walls reflect and distort the waves.

One problem that arose during testing was that in the small room shown in the top right corner of Figure 4.6, there were significant delays in connecting to the Fitbit One. Collecting samples for the signal strength for both devices in this room became difficult or impossible, which is why that room does not appear to have the same wave effects. In the future, the collection process should record delays in connecting and limit them. Failure to connect then could be assumed to be weaker than the weakest signal strength.

4.4 Designing the 2D Multilayer Perceptron

Once the RSSI and position data is gathered, it can be used to train a neural network. The neural network used is a common multilayer perceptron with a sigmoidal activation function, but with specialized inputs and outputs, as shown in Figure 4.7. Each beacon is given two inputs, the mean and standard deviation for a 5000-sample position. The mean RSSI is normalized linearly from -95 to -58 to 0 to 1. The standard deviation is normalized from 0 to 8. The x and y positions, based on the boundaries of the map, are normalized from 0 to 230 and -40 to 80 respectively to 0 to 1. The minimum and maximum RSSI could also be used as inputs from each beacon but were not in the following results because they do not seem to be very reproducible between tests. Logically, the standard deviation and mean are a combination of all of the samples and would therefore be less prone to error than the minimum or maximum, which is just a single outlier sample. The positions of the beacons are never measured or added to the neural network, just the positions of its signal strengths.

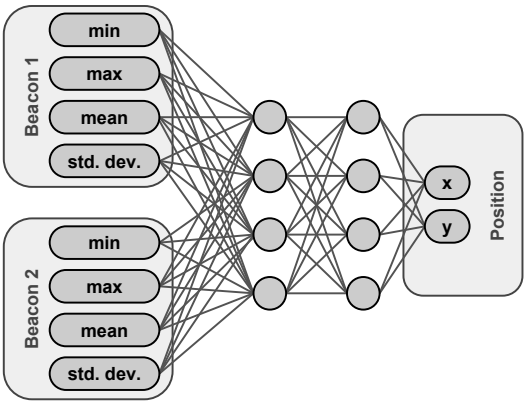
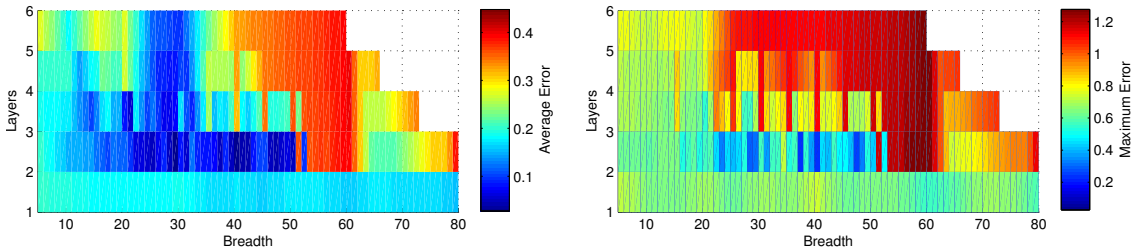


Figure 4.7: Mapping of inputs and outputs to the MLP ANN.

To discover the number of layers and number of nodes in each layer (or breadth) that would produce the most accurate results, 121 size combinations were tried. The results of those trials are shown in Figure 4.8. Each graph shows the Pythagorean magnitude of the combination of the x and y errors as a percentage of the range of the map. This may be somewhat accurate for the average graph, but since the maximum error in each direction can refer to different points, it can actually make the error appear much worse than it really is. That is how the maximum error gets to 120%.



(a) Average Error (b) Maximum Error

Figure 4.8: Accuracy of neural net shapes for 50,000 training iterations.

A single layer of neuron nodes is apparently insufficient to fit the data, since it never significantly changes no matter how many nodes wide it is. The most accurate shapes are mostly just two layers. Neural nets that have a breadth of 60 neurons are extremely inaccurate regardless of how many layers are in the neural network,

which is somewhat unpredictable behavior. Similarly, a breadth of slightly less than 30 seems relatively accurate for each layer size. This is a very unexpected behavior because another layer should drastically change the ability of a neural network to fit data, and a more complex one should work better, not worse.

The most accurate breadth is either 30 or 40 nodes wide. The two-layer neural nets that are 30 or 40 nodes wide achieved 1.8% average error in the x and y directions, with about a 20% error maximum. Given that the map is 240 and 120 inches in the x and y directions respectively, the neural net should be accurate on average to 4.829 inches. By coincidence or not, this happens to be almost exactly the average wavelength of the 2.4 to 2.48 GHz radio waves that were measured, which is 4.837 inches. The neural network was never trained with any explicit data on the wavelength, yet the wavelength arose as the average error with extreme accuracy.

The accuracy of the neural network could easily be enhanced by adding more beacons. If the shape of the obstacles in the room had not created such irregular shapes, and the beacons were not placed in corners on the same side of the map, trilateration might require a third beacon at a minimum. It might be reasonable to supplement the BLE signals with WiFi-based signal strengths. Additional measurements, if reasonably accurate, should help the accuracy as a general rule. However, in this case it may be particularly helpful because the two may interfere with each other. In this experiment, it does not seem that any interference was caused by WiFi, since there was a router in the room at coordinates (50,70) and several others outside of the room at unknown positions.

4.5 ANN Positioning Results

According to Figure 4.8, the best shape of MLP to use has two layers and is 40 nodes wide. Then signal strength collected in the previous experiment was run through the neural network and the distance between the prediction and the actual value was calculated to form Figure 4.9. If a new neural net were trained, the heatmap would likely look different due to the randomness of the initial link weight values.

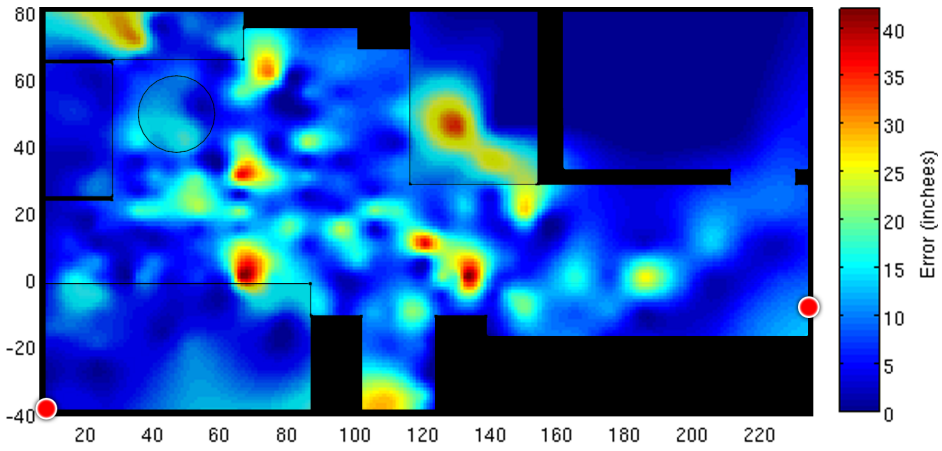


Figure 4.9: Heatmap of the distance the trained neural network is wrong by.

While there is no correlation to the points of high error and the placement of furniture and other obstacles on the map in Figure 4.6, there is a reason for the error. The issue is that the data cannot be fitted because the RSSI to position conversion function would need to be 1 to N, and the decision between which possible to choose position cannot be made. There are coordinates that are far apart but have essentially the same signal strengths. The most extreme distance example is at inch coordinates (35,70) and (110,-35). These two points are extremely far apart - about 93% normalized distance units - the difference between the beacon signals is less than 10% normalized RSSI units. So, the neural network instead will average

the two output positions. When a trained neural network sees the RSSI for one of those two points, it cannot distinguish between them and will end up averaging somewhere in between. In the most extreme error example, the neural net is wrong by 40 inches.

The solution is to find a way to ensure the uniqueness of the signal strength inputs for a given position. The easiest way to do this would be to add a third beacon at a different position. Alternatively, the usage of other data, such as the previous position calculated by the neural net, or other sensors could be used.

4.6 Radio Positioning Conclusion

A neural network was trained to fit signal strength data for one room to be accurate to about 5 inches on average and 40 inches in the worst cases. This performance is significantly better than GPS-based positioning, which is accurate to 35 inches under ideal conditions - i.e., not indoors. To realize the full potential of neural network-based positioning, at least three beacons should be used to ensure the uniqueness of the combination of signals corresponding to just one position. The downsides to this approach are that it has to be trained ahead of time and that the system requires that conditions be identical between the training and during operations. Moving the beacons an inch or new forms of interference could completely throw off the accuracy of the neural network. To combat this, and to further refine the accuracy of the network, it could be trained during operation with another sensor. This technology could be useful for autonomous robots because positioning methods are needed that work indoors and do not accumulate error over time due to integration like a wheel encoder or accelerometer.

CHAPTER 5

RELATIVE POSITIONING SENSORS

Absolute positioning is useful for navigation and maintaining positions. Its main advantage is that it does not accumulate error, but it is typically not very precise. Relative positioning is useful for tracking small changes. For example, a camera pointed towards the floor can track changes in position. Another sensor that tracks changes is an inertial measurement unit (IMU) which typically measures acceleration and changes in orientation.

In addition to positioning, distance measurements are also useful for navigation. A downward-facing distance sensor can reliably determine altitude. Other distance sensors pointed in the horizontal plane can detect obstacles to avoid.

5.1 Distance Sensors

One of the most important distances to measure is the downward direction because it is critical for maintaining altitude. Altitude cannot easily be maintained due to the varying of the voltage level of the battery, inaccurate relation between output signal, and the potentially unknown mass of the craft. Since the quadrocopter is designed to be used indoors, it can be expected that the floor or other obstacle should be between 1 and 9 feet away vertically at any given time. Given this range, an ultrasonic sensor is an ideal, low-cost solution. The MaxBotix HRLV-MaxSonar EZ can detect distances from a range of 1 foot to 16 feet with 1mm resolution. This

higher resolution sensor was chosen due to the importance of maintaining a steady altitude. It is possible that noise from the propellers could produce interference, or that the air turbulence the ultrasonic waves must pass through might distort the signal.

The wiring of the sensor is simple. The signal processing and unit conversion is done entirely within the sensor itself. After soldering the TTL jumper on the bottom, the sensor outputs easily readable numerical characters in millimeters over a serial connection. The enable and disable feature is connected to a GPIO pin on the Arduino Due by the orange wire in Figure 5.1 so that, if necessary, the sensor can be temporarily disabled to prevent crosstalk. Crosstalk is when multiple ultrasonic sensors are sending and receiving ultrasonic chirps, but one hears the echo of another before its own.

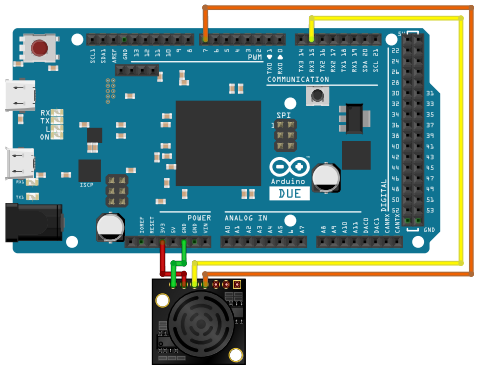


Figure 5.1: Maxbotix wiring.

The HC-SR04, shown in Figure 5.2 is significantly less expensive but is a more analog ultrasonic sensor. They do not provide serial output like the Maxbotix ones do. Instead, a ping must be triggered by a pulse. Then a microcontroller must wait for a pulse when an echo is heard. The distance can then be calculated based on the known speed of sound in Eq. 5.1 [27]. The HC-SR04 sensors are inexpensive and manually triggered, which makes them ideal for object detection in the horizontal directions because four of them are required.

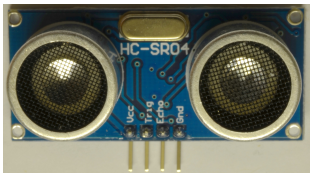


Figure 5.2: HC-SR04.

$$d = \frac{t}{2}\,\mu s \;\cdot\; 0.3432\,\frac{mm}{\mu s} \tag{5.1}$$

To potentially overcome some of the limitations of ultrasonic sensors, other distance-measuring methods should be considered. Infrared (IR) sensors are effective at much shorter ranges. They work by shining IR light and measuring the brightness of the same IR frequency to gauge how much light was bounced back. A sensor like the Sharp 2Y0A02F26 is commonly available through many hobby robotics retailers. It is simple to wire, as shown in Figure 5.3, since it has an analog voltage output. However, measurement in terms of a length unit requires additional work. First an experiment should be conducted where voltage is measured at various known distances, shown in Figure 5.4. The data is then used to create a best fit curve that determines distance as a function of the measured voltage. Given the data, a fourth-order polynomial was found to be the best fit, shown in Eq. 5.2.

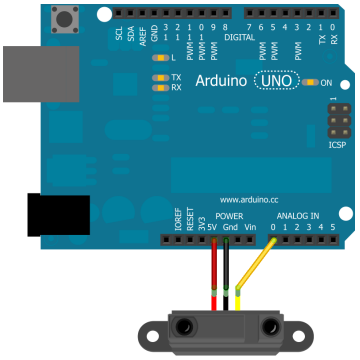


Figure 5.3: IR sensor wiring.

$$d = -0.0000398399385V^3 + 0.045689977V^2 - 17.4853558V + 2571.05272 \quad (5.2)$$

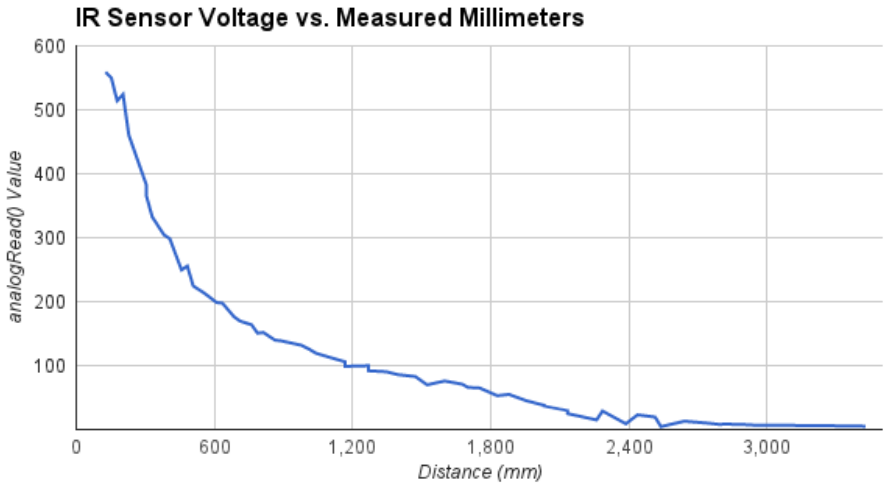


Figure 5.4: The relation between voltage and distance.

5.2 Inertial Measurement Unit

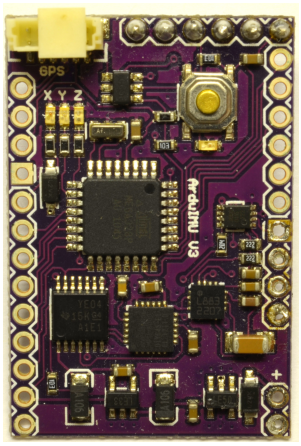


Figure 5.5: ArduIMU.

An inertial measurement unit (IMU) gathers orientation, acceleration, or sometimes position information from several different kinds of sensors. An ArduIMU V3 (shown in Figure 5.5) was selected because it is a relatively inexpensive IMU, with many of the same sensors as more expensive ones, and is reprogrammable using the Arduino IDE and an FTDI cable. It features a gyroscopic sensor that measures angular velocity, an accelerometer, a magnetometer that functions as a compass, and an Arduino-compatible Atmega328 running at 16MHz. The gyroscopic sensor provides very responsive information on the orientation, but because it is measuring the change in orientation, it is prone to drift error. The accelerometer predominantly measures the direction of gravity as long as the craft is stationary or maintaining a speed. Knowing which direction is down helps to correct the roll and pitch drifts, but not the yaw. To correct yaw drift, the magnetometer is used to find magnetic north.

Since the ArduIMU can be reprogrammed, the output sensor information and format can be easily altered. Additionally, different methods for combining the sensor information can be applied. Since the ArduIMU will be used by the much more powerful ARM-based Arduino Due processor, the ArduIMU should simply read the sensors as quickly as possible and send all of the information to the Due over serial. Not only does the Due processor have a higher clock rate, it also is capable of doing arithmetic with true “double” decimal data types, whereas the Atmega can only deal with “float” data types which have half of the precision.

5.3 Camera Motion Tracking

USB cameras can act as an inexpensive sensor that can provide a critical new dimension for the control system. While distance sensors can detect the altitude of the quadrocopter, relative to the floor, there are not many ways to gauge position in the X-Y plane precisely. Absolute positioning systems, like GPS or the previously discussed Bluetooth trilateration method, are not accurate on a small scale - only accurate to several feet or 5 inches respectively. A downward facing camera can detect changes in position as small as an eighth of an inch, depending on the resolution of the camera, the height from the floor, and the texture of the floor.

The visualization of the data from the motion tracking program is shown in Figure 5.6. It uses OpenCV to detect hundreds of high-contrast points like corners throughout an image by using the `goodFeaturesToTrack` function. The high-contrast points are shown as blue circles in Figure 5.6. Then it looks to see where each of those high contrast points moved using `calcOpticalFlowPyrLK`. The overall change in position is calculated based on summation of the change in position of all points on the screen, is shown by the green line in Figure 5.6 and calculated using Eq. 5.4. Changes in orientation can be taken into account using the IMU, as shown in Eq. 5.3.

$$\Delta X = (\Delta R_{imu} X_{cur}) - X_{prev} \tag{5.3}$$

$$\Delta x = \frac{\sum_{p=0}^{n} \Delta X_p}{n} \tag{5.4}$$

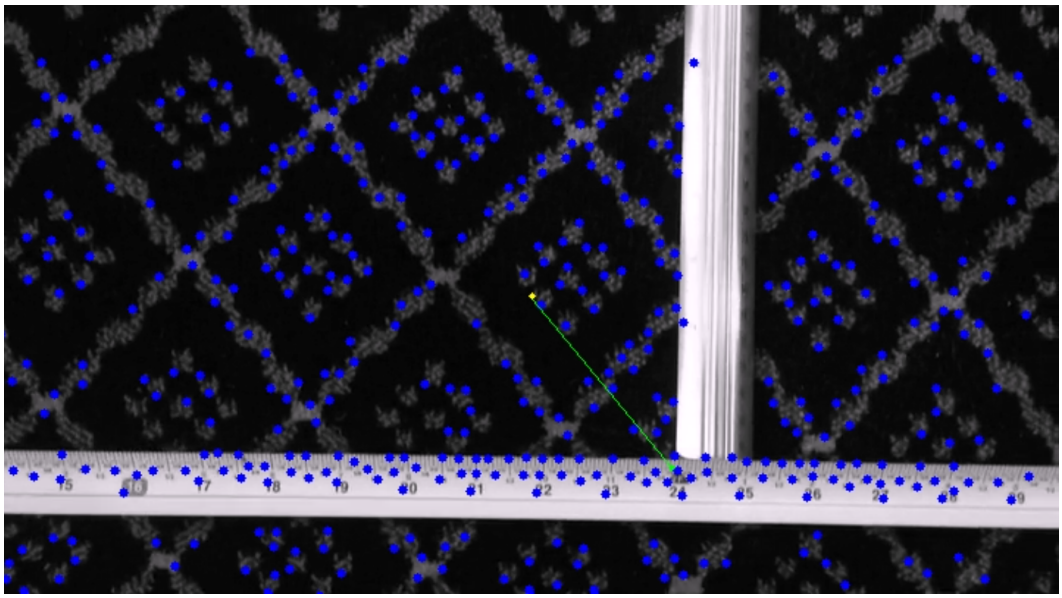


Figure 5.6: Camera-based position tracking using OpenCV.

As long as there are enough high-contrast points still in the view of the camera, the same points will be searched for each time. As a result, the total change in position is not as prone to accumulating error due to numerical integration like the gyroscope in the IMU is. While they both add changes in position together, the camera sensor can use the same point of reference between each addition. With the gyroscope, errors will accumulate because errors in measurements are not corrected.

There are still several downsides to this sensor method. It will accumulate errors as it moves over larger distances because new high-contrast points must be found, and former ones are forgotten. Additionally, it does not necessarily compensate for motion in certain parts of the screen that are different from the rest. For example, if an object in view were to move, that would be interpreted as if part of the floor were moving. A more common problem is the reflection of light from overhead sources. As the camera moves, the reflection of a light source seems to move slower than the floor. As a result of these issues, this sensor is primarily useful for helping a quadrocopter maintain a position.

CHAPTER 6

OBSTACLE TRACKING

Storing obstacle data in a way that it can be retrieved and analyzed quickly is critical for real-time operations. As data accumulates, it becomes impractical to check every data point collected. If the data collected about potential obstacles is stored in an organized way, less computationally intensive searching will be required.

6.1 Data Structure and Point Searching

The first way to simplify the obstacle storage is to force the data into voxels - or volumetric pixels. Instead of trying to store a mesh shape and having to check if collisions between meshes are occurring, the shapes are broken into non-overlapping cube particles. While a mesh is able to represent large surfaces with minimal memory usage, storage in voxels allows fast search algorithms to be used.

This spatial storage and search algorithm is similar to a binary search tree. A binary search tree is searched starting at a root node and a value to search for. An example of the algorithm in practice is shown in Figure 6.1. The value to search for, 13, is compared to the value of the node, which is 11. Since the value is greater than the current node's value, it moves down a level towards the larger nodes. On the next layer, the value of 13 is less than the new current node value of 17, so the traversal

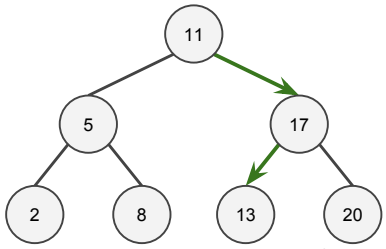


Figure 6.1: Binary search tree.

happens in the opposite direction. The algorithm consists of these two steps repeated until the value is found, or there is no node in the direction traversal should happen.

The first major difference of the spatial searching algorithm from a binary search tree is that each node can have up to four or eight child nodes depending on whether it is for 2D or 3D space respectively. Each traversal is navigated using two or three simultaneous "more or less" decisions - one for each dimension. The second major difference between the spatial search algorithm and a simple binary search tree is that most of the true obstacle data is on the bottom layer.

For the purpose of simplicity of visualization, the search algorithm can be seen in 2D space in Figure 6.2. The search begins at the center of a large square that encompasses all obstacles. Then the direction the target is in is determined by whether it is above or below the horizontal center line and right or left of the vertical center line. The quadrant that contains the target is then chosen and the direction of the target is checked again based on the center of this quadrant. The process repeats until the exact target is found.

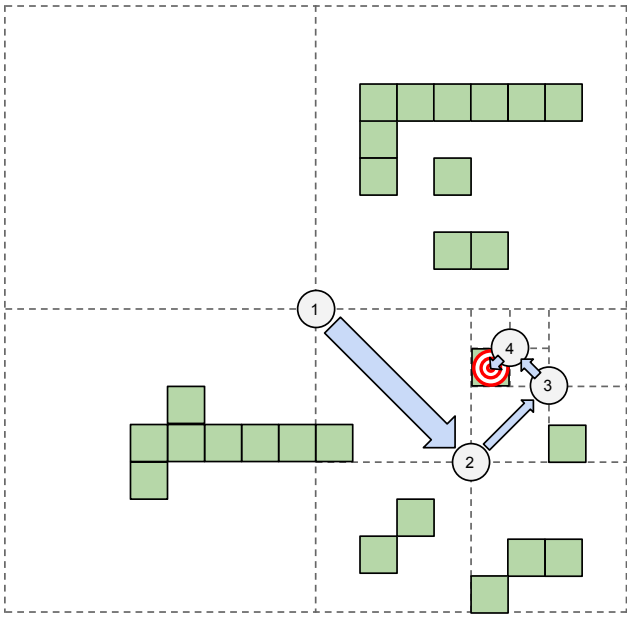


Figure 6.2: Searching for a point in space using subsquares.

The data structure that drives the algorithm shown in Figure 6.2 is shown in Figure 6.3. Each square has an array of variables to store the coordinates of the center of the square, a variable for the size of the square, and pointers to each of the four squares that make up its own quadrants. Each of those squares has its own data and its own subquadrants. A minimal version of code to traverse the tree to find 3D spatial data is shown below.

```
current = root;
for( int i = 0; i < maxDepth; ++i ) {
    x = target[0] >= current.center[0];
    y = target[1] >= current.center[1];
    z = target[2] >= current.center[2];
    current = current.quadrants[x][y][z];
}
```

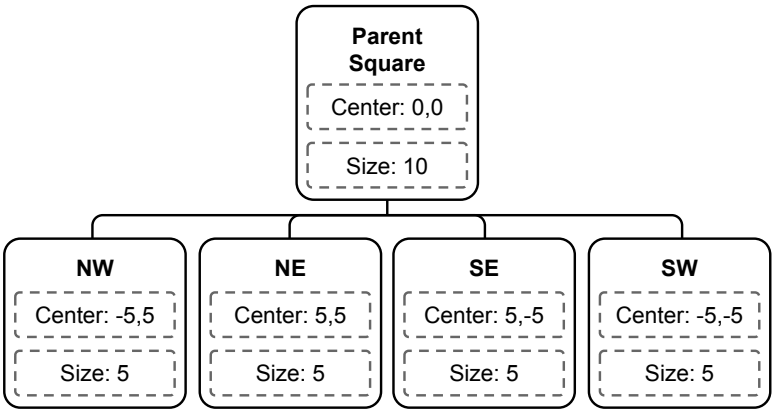


Figure 6.3: 2D spatial data organization structure.

There are some further optimizations. To save memory, the tree should not be fully initialized. subcubes should only be added once they contain an obstacle particle. Ensuring each parent cube contains an obstacle particle can allow faster searching and allow bulk operations. For example, to find out whether there is an obstacle anywhere in the top left quadrant of Figure 6.2, it could be determined immediately that there are none because the quadrant would not exist yet. Additionally, to select

or delete large regions of spatial data, any parent cube can be selected where each of the four corners are fully contained within the region.

6.2 Performance

The performance of the spatial data organization is shown in Eq. (6.3), which is a vast improvement over unorganized data storage that would have $O(n)$ time complexity. The resolution is a potentially limiting factor, but it can be set to a reasonable value based on the known resolution of the sensors that will be creating the obstacle data. The maximum number of particles in Eq. (6.1) is not a limiting factor because a new and larger root can be created when an obstacle is stored outside of the current boundaries. However, if too fine of a resolution is chosen, then it will use too much memory and likely be slower to search than using mesh objects.

$$\text{Maximum obstacle particles: } n = 4^l \tag{6.1}$$

$$\text{Resolution: } r = \frac{s}{2 \cdot l} \tag{6.2}$$

$$\text{Point search time complexity: } O(l) = O(\log_2(n)) \tag{6.3}$$

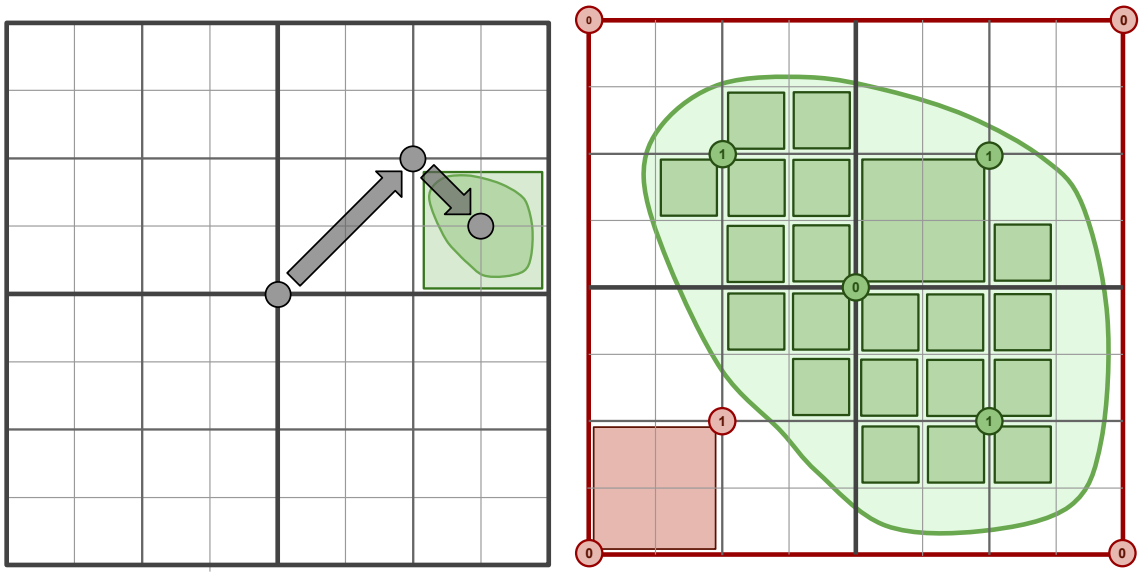
Where:
- s: the size of the root cube.
- l: the number of layers.
- n: the number of particles.
- r: the resolution or the size of an obstacle particle.

6.3 Region Searching

Selecting a region of obstacles is much more resource intensive but can still benefit greatly from the data structure organization. Searching for a region allows a large section of obstacle data to be deleted if a sensor covering the area does not find anything. It also allows the quick creation of large shapes. Finally, it can be used to determine if there are any obstacles in a region, which can be used for path planning. The region search algorithm does not require a specific shape for selection but has the major restriction that the shape must be convex. Cubes, spheres, and cones are acceptable shapes, but toroids or rings would not be. Since many shape types may be used, the search function should not rely internally on a specific shape definition. Instead, it relies on two types of condition checks, which can be defined externally and are used in each of the two phases respectively.

The first phase requires a function that determines whether the shape being searched for is in a cube fully, partially, or not at all. It can be implemented either by using a bounding box with little regard to the precise shape or by using the exact geometry. The procedure for a 2D spatial region search is shown in Figure 6.4. The first phase, shown in Figure 6.4a narrows down the selection. It starts in the center of the root square; each of the quadrants is tested with the function to determine if the target shape, shown in green, is contained within them. If the shape is fully within the quadrant, the previous step is repeated with the new quadrant. If the shape is partially within several quadrants, then the search proceeds to the next phase.

Phase 2, shown in Figure 6.4b, builds the actual selection of the subsquares. It requires a typically simpler function which tests to see if a single point is within



(a) Phase 1 (b) Phase 2

Figure 6.4: Procedure to select an arbitrary region in space.

the target region. First, a point within the region must be found. Ideally, the point within the region is the center of the current square. Then, by using the point test, along with the fact that the shape must be convex, the search can branch out and find the boundaries of the shape. This is why the shape must be convex - the point found is not known beforehand, and a line must be able to be drawn from that point to any point on an edge. Using this fact, large regions can be ruled out or included without testing the entirety of their contents. For example, during the first expansion iteration, the top right quadrant can be entirely selected because the top left, top right, and bottom right corners are within the selection region. Since these points form a triangle that includes the larger subquadrant, its contents must be within the region too due to the convex shape. Similarly, during the second expansion iteration, the bottom left subquadrant on the bottom left quadrant can be ruled out because the top left corner, bottom left, and bottom right corners of the bottom left quadrant are not within the region.

This algorithm is by necessity computationally intensive due to the way a region is defined. The functions to test whether a point is within a search region and to test if a search region is contained in a quadrant both likely require dot product, division, and cosine operations which are significantly slower than simple condition check operations. Additionally, many tests must be conducted on many of the points in each layer in order to fully construct the selection region. Due to the likely greater number of voxels required to represent a shape than the number of vertices required, this method may not be faster than a mesh intersection algorithm.

6.4 Nearest Obstacle Search

Since the region selection algorithm is computationally expensive by nature, a different search type is required for the "trial and error" probing method common to many path-planning algorithms. Finding the closest obstacle to a point is a much faster operation. This information can be used to quickly create obstacle avoidance gradients for attractive and repulsive fields. A point can be quickly found using the procedure defined in Section 6.1. Using the memory optimization that quadrants without obstacles do not exist, a search for a non-zero distance will stop before it finds the obstacle particle. Then one of the siblings of the non-existent quadrant likely has the closest point. The closest particle should be found in one of those. However, neighboring non-sibling quadrants also must be checked. To do this, the search must revert to one of the larger layers to find the quadrants on the same level that also touch the non-existent one.

CHAPTER 7

PROTOTYPE CONSTRUCTION

The purpose of the prototype is to test an actual implementation of some of the design concepts discussed in previous sections. It will focus primarily on the controller and not path planning, since it is a first prototype. As a result, it will need to be human controlled. However, software and components should be chosen that would allow it to be easily modified to be autonomous with minimal alteration later.

7.1 Chassis Design

The design of this quadrocopter will focus on safe indoor flight. Therefore, unique consideration should be taken to ensure that the propellers are protected from potential obstacles. Propeller collision would be catastrophic since it would likely break the propeller and damage what it hit - which could be a person. Once a propeller is broken, flight cannot be stably controlled and further collision could occur. Reducing the possibility of propeller collision is therefore one of the most important functions of the chassis design. This design will also rely on readily available components so it can be reproduced easily.

7.1.1 Stock Components

The design of the chassis depends on the performance and specifications of stock components. Since there is limited variety and availability of specialized quadrocopter components, some readily available components were first selected so the chassis could be designed around them. Two kinds of propellers were picked so the most effective could be chosen by experimentation.

Table 7.1 shows that the battery is by far the heaviest of the stock components. A comparison of Tables 7.2, 7.3, and 7.4 shows that the component with the most restrictive current limitation is the electronic speed controller (ESC) at 18 amperes.

Table 7.1: List of Stock Quadrocopter Parts

Part Type	Model	Weight (oz)	Dimension (in)	Qty.	$ ea.
Motor	DJI 2212	1.980	$1\frac{1}{4}$ D x $1\frac{15}{16}$ H	4	25.99
Propeller	DJI 8045	0.235	$8\ \frac{1}{16} \times \frac{7}{8}$	4	1.62
Propeller	DJI 1045	0.280	$10\ \frac{1}{16} \times 1\frac{7}{8}$	4	3.50
ESC	DJI ESC18A	0.815	$2\frac{1}{2} \times \frac{5}{16} \times 1$	4	17.99
Battery	Traxxas 2849	11.100	$5\frac{5}{16} \times \frac{15}{16} \times 1\frac{10}{16}$	1	75.99

Table 7.2: Traxxas Battery #2849 Specifications

Type	Capacity	Cells	Voltage	Wiring	Discharge Rate	Charge Rate
LiPo	4000 mAh	3	11.1	3s1p	25C (100A) 50C (200A) max	1C (4A) 2C (8A) max

Table 7.3: DJI Opto-coupled Electronic Speed Controller (ESC) Specifications

Current	Voltage	Battery Compatibility	Frequency Response
18 A	11.1-14.8V	3S-4S LiPo	30-450 Hz

Table 7.4: DJI Brushless Outrunner 2212/920KV Motor F330-550 Specifications

rpm/V	Shaft	3S Battery/Prop	4S Battery/Prop	Current	Max Current
920 KV	8mm	11.1V: 10×4.5	14.7V: 8×4.5	15-25 A	30 A

7.1.2 Performance Testing of Stock Components

To further define design constraints, the performance of the components should be tested. The most important constraint is the maximum lift each motor can impart, since this will dictate the maximum weight of the craft. Additionally, determining the current consumption of the motor at various speeds can be used to predict time of flight.

The first step in testing a single motor is finding a suitable power supply. While the batteries are powerful enough and available, an AC to DC power supply was used for convenience and reproducible testing. For the lift testing, an ATX power supply was used, which is typically used in a computer [28].

The electronic speed controller (ESC), motor, and propeller combination were wired to the ATX power supply and an Arduino shown in Figure 7.1. The motor and propeller were screwed to a heavy block with a handle so it could safely rest on a weighing scale. While the motor was not spinning, the weight on the scale was measured. Finally, sketches were uploaded to the Arduino such as the following C++ code so a servo signal could be sent to the ESC.

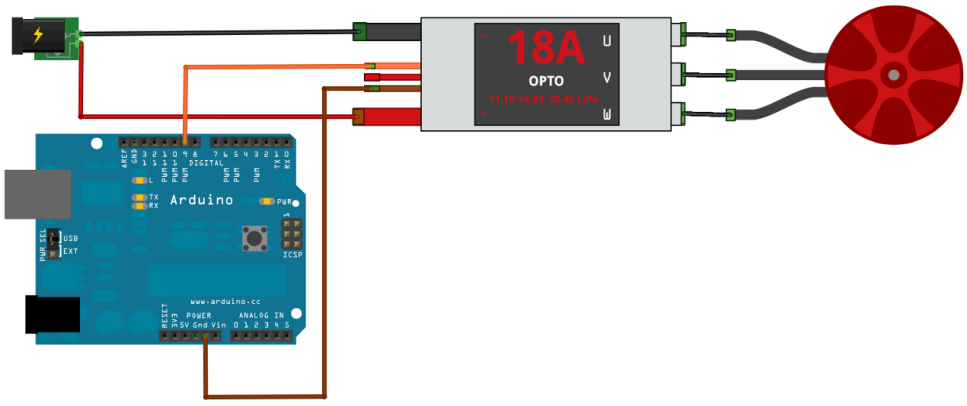


Figure 7.1: Motor test wiring diagram.

```
#include <Servo.h>
Servo myservo;  // create servo object to control a servo

void setup() {
  myservo.attach(9); // Servo object signals to the ESC on pin 9
}

void loop() {
  int val = 1350; // speed in microseconds
  myservo.writeMicroseconds(val);
}
```

$$F_{scale} = W + F_{thrust} \tag{7.1}$$

$$F_{thrust} = 5.5\text{lbs} - F_{scale} \tag{7.2}$$

The lift is measured as the weight on the scale with thrust less than the weight at rest, shown in Eq. (7.2). The procedure is repeated for the full range of the servo signal, and for both the 8-inch and 10-inch propellers - from 1000 microseconds to 2000 microseconds. The results, shown in Figure 7.2, show that the 10-inch propeller is capable of roughly 1 pound of lift. The smaller propeller is not capable of as much lift at only about 0.75 pounds. The lack of increased lift from a 1900-microsecond servo signal to a 2000-microsecond servo signal suggests that the motor with the 10-inch propeller may need more current than the motor, ESC, or power supply is capable of. Since the battery is capable of 200 amps, much greater than the DC power supply, the current usage is mainly a concern for flight time. Being maneuverable and able to lift cargo are of greater concern than flight time for this quadrocopter design, so the 10-inch propellers were chosen.

To explore the current limitations of the propeller system, an experiment was conducted to show how much current the motor draws at varying speeds. For this test, a variable 10V-14V 30A DC power supply was used with the model number KY-360W-12-L, as well as a panel current meter capable of measuring up to 30A.

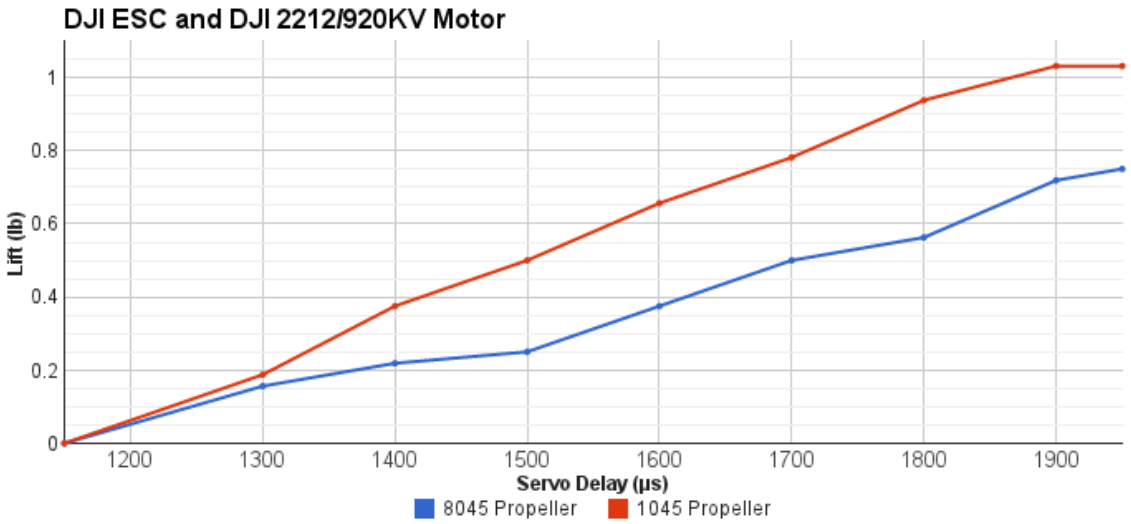


Figure 7.2: The lift each motor provides given a servo signal input.

The results in Figure 7.3 show that the maximum current draw is up around 21A when at the 11.1V that the lithium polymer battery can provide. Increasing the voltage by 10% to 12.2V shows that the current draw increases roughly 10% as well. Presumably the speed the motor goes and the thrust it applies also follow the same trend, but it is somewhat irrelevant due to the lack of availability of batteries at this voltage.

When maintaining a constant speed, the motor draws only 9A at full speed. It is the drastic changes from stop to full speed that could be problematic for the 18A limit of the ESC. However, the ESC does not state if the rating is for a continuous or maximum current. If all four motors were providing their full thrust, it would be able to fly for 7.36 minutes according to Eq. (7.3).

$$\frac{5400 \text{ mA} \cdot \text{h}}{4 \cdot 11000 \text{ mA}} = 0.12272727 \text{ h} \cdot 60 \frac{\text{min}}{\text{h}} = 7.36 \text{ minutes} \tag{7.3}$$

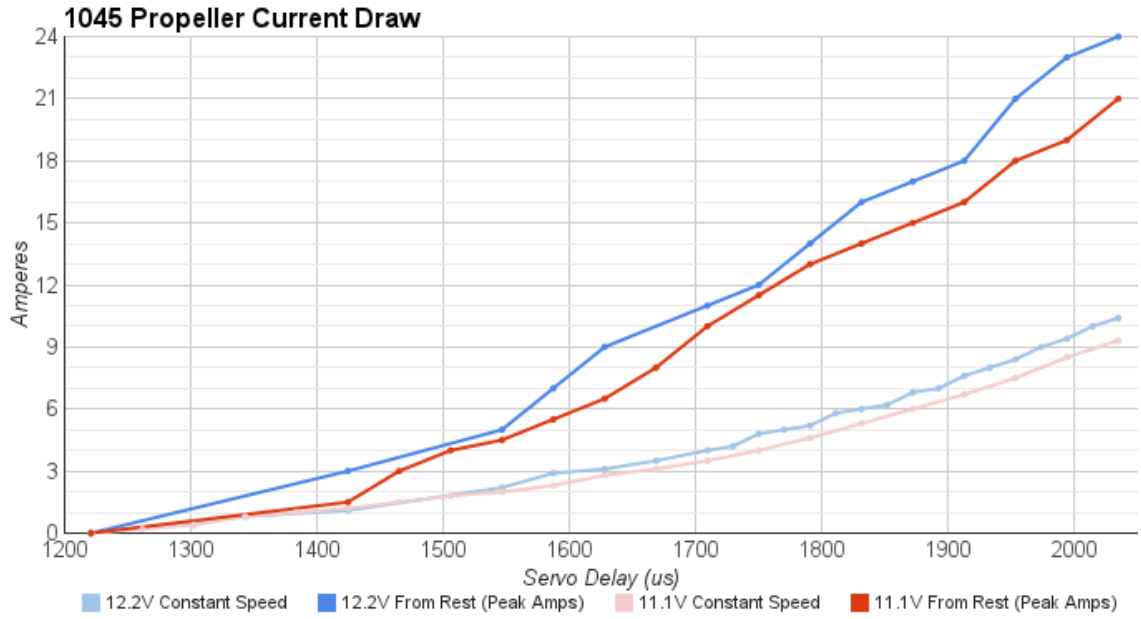


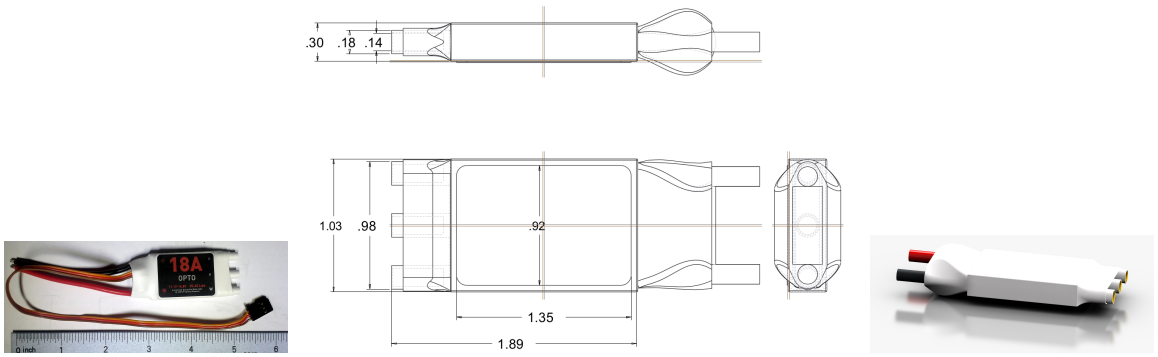
Figure 7.3: The current each motor draws at various speeds.

7.1.3 Computer-Aided Design

First, each of the stock components were accurately modeled using a caliper and Creo Parametric so that the dimensions of the design could be carefully tuned. The parts were weighed and the density of the models were modified so that the performance criteria, like the mass and inertia matrix, could be accurately measured.

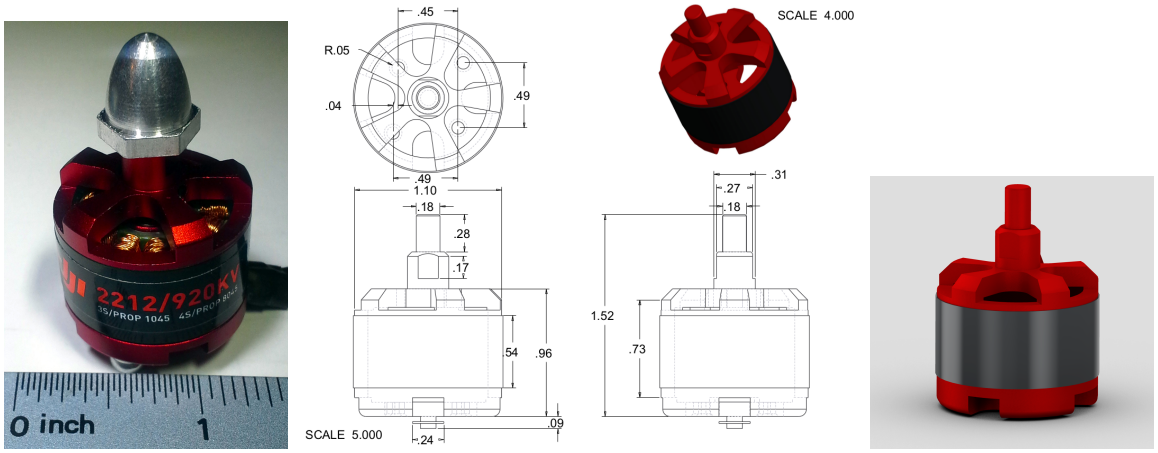
Figures 7.4 and 7.5 show the detailed models of the ESC and motor respectively. The other components are relatively trivial to model. The battery has very simple geometry. While propeller geometry is complex, for the purposes of designing the chassis, they are treated as a circular prism.

Conventionally, quadrocopters have a minimal frame that connects the motors to the controller and battery. Since most are designed for outdoor flight, the propellers are usually left completely unprotected to reduce weight, which increases flight time and maneuverability. On some quadrocopters, detachable semi-circular guards



(a) Photo (in) (b) Drawing (in) (c) Render

Figure 7.4: A photo compared to the model of the electronic speed controller.



(a) Photo (in) (b) Drawing (in) (c) Render

Figure 7.5: A photo compared to the model of the motor.

are affixed under each motor to protect the propellers from side impacts. Since the specifications of this design state that the quadrocopter will primarily be used indoors, it is not necessary that the propeller protection be removable.

The resulting design, shown in Figure 7.6, uses the standard consumer-available aluminum U-channel for the bulk of the chassis. The frame consists of two bent U-channel sections that interlock at the center. The frame acts as the landing legs and as protection for the propellers. In a more typical design, the landing legs would still be required for operation, so their placement at the farthest possible point from the center of mass does not increase the weight, but does increase the

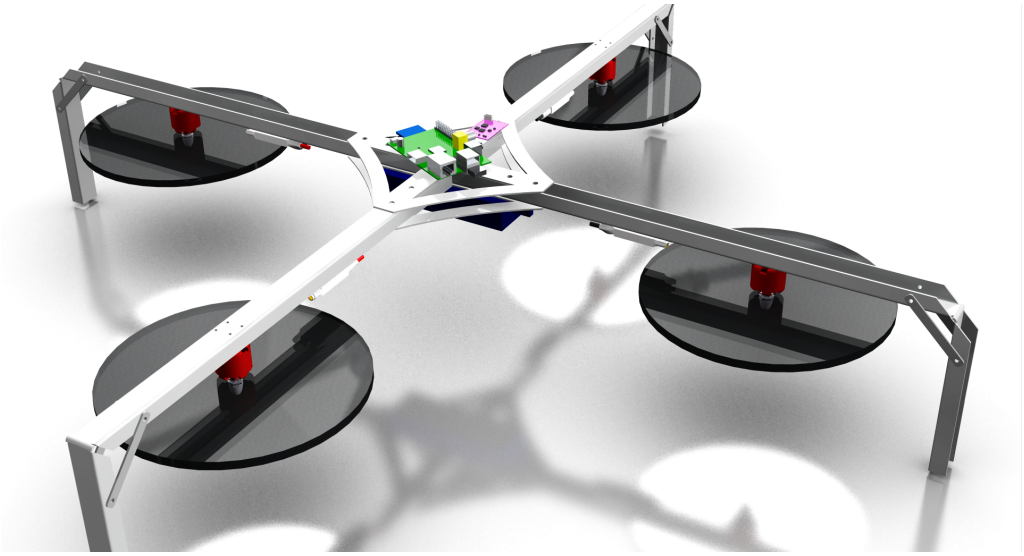


Figure 7.6: A 3D rendering of the quadrocopter design

moment of inertia. The increased moment of inertia means that the motors will have to exert more force to cause the same angular acceleration, so some maneuverability is sacrificed. In exchange, they provide additional safety for the propellers and more stable landing stance in the new position. According to the Creo model with appropriate densities, the expected mass of the fully assembled quadrocopter is 2.9 pounds with an inertia tensor shown in Table 7.5. The center of gravity is 0.9 inches below the center of the of the intersecting U-channel sections, which is about where the battery is located.

Table 7.5: Inertia Tensor at Center of Gravity in $lb * in^2$

$I_{xx} = 107.989$	$I_{yy} = 112.393$	$I_{zz} = 213.666$
$I_{xz} = 0.059$	$I_{zy} = -0.008$	$I_{xy} = -0.009$

The leg length is designed to stand the quadrocopter high enough off the ground to prevent the propellers from touching a typical blade of grass on a lawn, which was measured to be around 3 to 4 inches. While the quadrocopter is designed to be used indoors, it is conceivable that the craft would be started outdoors and then flown indoors. Furthermore, this is a reasonable factor of safety to prevent collisions

with small objects which could be thrown at dangerous speeds or allow landing on uneven surfaces.

The encapsulated design of the chassis requires the motors be mounted on the underside of the legs and point downward. This is an uncommon design, although some motors do point downward in octocopter designs where there is one pointed up and down on each leg. There is no particular reason why the motors should point upward. The brushless motors can spin in either direction depending on the order the three wires receive current, and two motors must be spinning in opposite directions anyway. A typical motor and propeller combination for a quadrocopter would not have issues flipping upside down, but the DJI motor and popellers have a small notch that locks the propellers into place which is larger on one side than the other. Even worse, the notch on the propellers is angled so it is difficult to attach upside down. This is likely to ensure that a consumer would not put a propeller on in such a way that it would be spinning in the opposite direction than the propeller is designed for. While the unusual orientation could potentially affect the cooling rate for the motors, it was not found to be an issue.

One of the most important features of the encapsulated design is that if a line is drawn from leg to leg in a top-down view, the propellers will not extend outside of the containing square, as shown in Figure 7.7. As a result, it is impossible for a wall collision to impact the propellers. If the craft were to collide into a less wide obstacle, like a person, there wouldn't be any protection unless a leg were in the way. Therefore strong, lightweight wire should be strung between each of the legs to further increase safety.

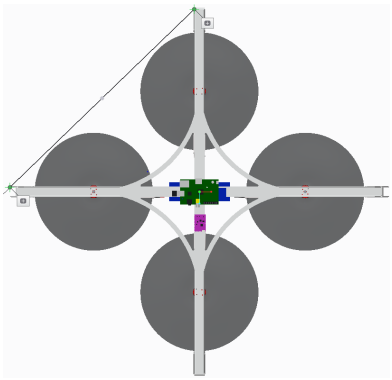


Figure 7.7: Encapsulation.

7.2 Chassis Construction

The chassis design requires minimal work to construct using parts available from a hardware store - 1/8-inch thick aluminum U-channel that is 7/8 inch wide and has 5/8-inch tall walls as well as an additional small plate of 1/8-inch thick aluminum.

A power miter saw was used to create the required 45-degree cuts for the bends on the legs shown in Figure 7.8. The cuts must be carefully made so that the bend radius is at least 1.5 times larger than the thickness of the aluminum at that point, or 1/8 inch to ensure structural integrity. So, the cuts were made so they left a gap 1/4 inch wide at the intersection. The remainder could then be easily filed down. When finished, a single leg pair section weighs slightly more than one pound.

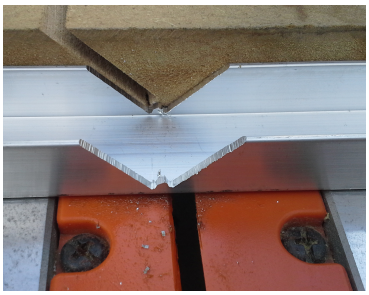


Figure 7.8: Miter cut.

Since the structural rigidity provided by the U shape has been removed at the leg bends, supports should be added in these areas. The support should connect the two loose sections diagonally, like those shown in Figure 7.9. The size of the triangle is determined by the radius of the propellers. The propellers can potentially touch at exactly one inch below the bottom of the horizontal support, so this point on the outer edge of the diagonal support should be more than 10.3 inches away. Pieces forming right triangles with a hypotenuse of 3.45 inches from screw center to screw center were found to provide this clearance. The supports are aluminum and can be

Figure 7.9: Support.

cut using the same power miter saw. The eight small supports in total weigh less than 3 ounces.

Perhaps the most challenging procedure when creating the aluminum frame is making screw holes for the motors. A drill press with clamps improves the accuracy when drilling, but marking the precise positions of each of the screw holes proved challenging since the holes could not be seen from the other side of either the motor or aluminum. To improve accuracy, a simple transparent plastic jig was made as shown in Figure 7.10.

Figure 7.10: Motor jig.

The constructed chassis in Figure 7.11 with main components, including the battery, weighs 2.93 pounds and the Creo model was estimated to weigh 2.91 pounds. While the use of wood instead of aluminum for the connector plates at the center may be lighter, they are solid pieces and the model also did not take into account the weight of machine screws. The machine screws and nuts for the leg supports alone weigh almost 0.25 pounds. They are likely steel and could potentially be replaced with a aluminum rivets if weight becomes a significant issue.

Figure 7.11: The construction of the basic chassis frame.

7.3 Selection and Attachment of Components

Nylon, plastic, #4 threaded risers and machine screws were used for attaching sensors to the wood panel in the center because they are lightweight and a compatible size for most components. A tap was used to eliminate some of the need for nuts, further reducing the weight. The plastic risers also electrically isolate the component from the conductive parts of the chassis, preventing short circuits. Additionally, they ensure that parts like the microcontroller, which need to dissipate heat quickly, are well ventilated and have exposed surface area. Finally, they help to keep the wiring organized because wires can pass underneath the components and are not left free to move.

7.3.1 Processing Components

There are two kinds of microcontrollers that will be considered. First, there is a traditional microcontroller which is ideal for time-sensitive input/output and other low-level controls. Second there is a newer form of device referred to as a microcomputer. A microcomputer has a similar size to a microcontroller and provides exposed access to digital I/O and serial connections but runs a full operating system and has traditional interfaces seen on typical computers, like USB, HDMI, and audio ports. Microcomputers have much more powerful processors than microcontrollers, can run a larger variety of programming languages, and can utilize existing drivers and hardware abstraction layers that a modern operating system provides. For example, ethernet and wireless internet adapters can be used for communication between the device and internet, just as on a typical computer. As a result of the added complexity of the multi-tasking operating system, microcomputers are less ideal than microcontrollers for time-sensitive operations.

A microcomputer will be required because of the access to high-throughput 802.11N WiFi networking and the USB interface that will allow the use of inexpensive cameras. Since real-time processing of the video feed from the downward-facing camera is desired, a fast microcomputer should be chosen. The microcomputer will be responsible for five cameras, network communications, path planning, and the spatial searching presented in Chapter 6. The demand on the processor could be significant, and it could benefit from the use of multiple cores since the tasks can be split into different processes.

To ensure that the data from the IMU is used immediately and to ensure that the servo signal to each of the ESCs is uninterrupted, a microcontroller should be

used too. The additional component will add some programming complexity because information must be relayed between components. To accomplish the responsibilities of the controller, not much computational speed is necessary. A faster processor will ensure that the controller can output signals at the full speed of once every 2 milliseconds. Another factor to consider is the number of digital general-purpose input/output (GPIO) pins available, which would allow RGB LEDs to be controlled and used as navigation lights. Twelve GPIO pins would be required to control the lights in a way that would allow the navigation lights to be changed. In addition to the twelve GPIO pins, four PWM-capable digital pins are necessary for connection to the ESCs. Another restriction is that there are three TTL serial inputs and one TTL serial output that the microcontroller will need to process: the IMU, ultrasonic, and microcomputer. The Arduino Due and Teensy are the only options listed that have more than one TTL serial connection. Another factor to consider is that the microcontrollers with ARM processors can use a true "double" data type and therefore have the advantage of greater precision for decimal arithmetic operations. Therefore, the Teensy 3.1 option might be ideal due to its minimal cost and weight, as shown in Table 7.6.

To satisfy the intense multi-processor requirements, the networking, and the ARM microcontroller requirements all at once, the Udoo Quad is chosen. It is one of the fastest microcomputers available in this form factor and includes the same microcontroller that is in the Arduino Due embedded in the device. Additionally, it also has an 802.11N USB adapter embedded. The most significant downside to the Udoo Quad, as seen in Table 7.7, is that it is by far the heaviest of the options considered. Its powerful processor requires a large aluminum heat sink to dissipate the heat, which makes up the majority of the weight. If the software requirements are proven to be less intense, a lighter weight option would be better. If this is

Table 7.6: Weights and Speeds of Microcontrollers

Microcontroller	MSRP (USD)	Weight (oz)	Clock (MHz)	RAM (KB)	GPIO Pins
Arduino Pro Micro	20	0.1	16	2	12
Arduino Uno	29	1.0	16	2	16
ArduIMU	79	0.1	16	2	2
Teensy 3.1 (ARM)	20	0.1	72	64	34
Arduino Due (ARM)	51	1.3	84	96	54

Table 7.7: Weights and Speeds of Microcomputers

Microcomputer	MSRP (USD)	Weight (oz)	Clock (MHz)	RAM (MB)	Dhrystone (MIPS)	GPIO Pins
Raspberry Pi B	35	1.1	700	512	0.81	12
BeagleBone Black	45	1.3	1000	512	3.32	65
Udoo Dual	115	3.5	2x1000	1024	5.62	54
Udoo Quad	135	5.0	4x1000	1024	11.24	54

done at a later time, the code written for the Arduino Due-compatible processor should function identically on other ARM-based microcontrollers. Additionally, the code written for the microcomputer should be compatible, for the most part, with any Linux-compatible microcomputer, with the exception of some device-specific bus addresses. The heavy weight of this microcomputer is acceptable because this quadrocopter prototype is not designed to pick up and manipulate cargo but will focus primarily on the control system and sensors instead.

7.3.2 USB Cameras

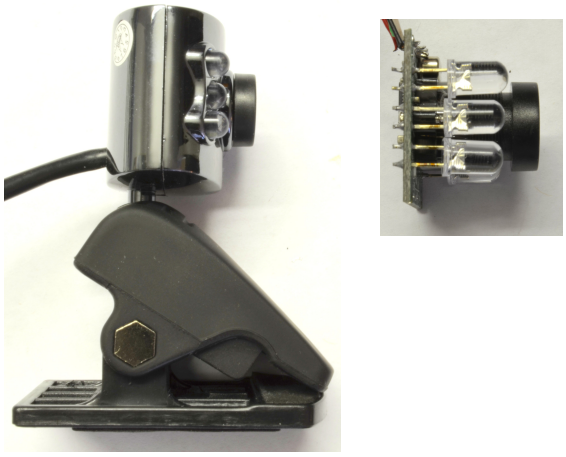
When purchasing USB cameras for the quadrocopter, the most important factor is USB Video Class (UVC) Linux compatibility. Without it, the drivers on the microcomputer may not be able to read images from the camera at all. Image quality

is important too, but increased resolution will come at the cost of higher bandwidth usage on the USB controller and wireless network. Therefore, inexpensive cameras are ideal, since the high resolution will not likely to be utilized.

Weight can be considered too when purchasing the cameras, but the weight can be significantly reduced by removing the plastic casing around the cameras, which is necessary for attachment to the quadrocopter anyway. The minimal device consists of a printed circuit board (PCB) and a plastic lens, which are not likely to change between cameras. In fact, some are simply the same PCB with a different case, as was discovered while testing some candidates. The cameras chosen do not have a discernible brand or model number, but have the USB ID `1e4e:0102` and were purchased for $4.84 each.

Originally, a single camera shown in 7.12a weighs 1.6 ounces. Since the chassis, battery, and motors weighs 3 pounds, and the motors can lift 4 pounds in total, the cameras would use up more than 50% of the available cargo weight. This does not leave enough lift capacity for the microcomputer or other components. If the casing is removed from the camera, shown in Figure 7.12b, then the weight is reduced to 0.47 ounces. Shedding 70% of the original weight means the camers will use only 17% of the available cargo capacity.

Further reducing the weight by shortening the wire and removing the outer insulation requires significant soldering work on very small delicate wires. USB has very strict requirements for the data lines being twisted close together, and removing the shielding may further increase the risk of data loss. Ideally, the original USB casing should be kept. However, a single section of USB cable long enough to reach the legs with the outer insulation and shielding intact weighs 0.375 oz - 10% of the available cargo mass. Removing the insulation reduces the weight to 0.09 oz, or 2.5% of the available cargo mass. The spacing between the wires was found to be a



(a) Original (b) Lightened

Figure 7.12: Reducing the weight on the USB cameras.

significant issue. A camera would not work at all if the wires were more than a half inch apart anywhere along the wire.

The webcams are slightly too wide to attach to the chassis using the original screw holes. Instead, they were be attached by using the fact that the lens is essentially a large machine screw. This is supposedly to allow the lens to be focused, but the only focus that works is when it is fully tightened. A hole the size of the threaded base of the lens can be drilled into the chassis. Then the lens can be detached from the camera, placed behind the hole, and the lens can be re-attached. The result is shown in Figure 7.13. Some filing or rubber spacers may be necessary to ensure firm attachment at the proper focus.

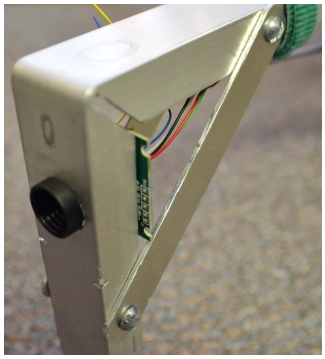
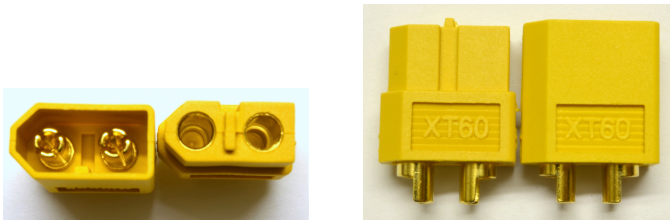


Figure 7.13: Lens mount.

7.3.3 Wiring

A wiring system for the ESCs should be designed so the battery can be unplugged easily and so they can be powered by two separate power supplies. Therefore, the connectors must be broken into two separate plug components: one that has a single connector and one that has two connectors. The single connector must be one compatible with the Traxxas connector on the battery. The remaining connections, including those on the ESCs, can be any other kind provided they have a high-enough current rating. XT60 connectors, shown in Figure 7.14, are an inexpensive and lightweight connector that can be used to ensure polarity. The angled end, on the left of Figure 7.14a and 7.14b, connect the grounds and the flat end connects the +12V. While there is no requirement that the connections be made this way, the convention is noted on the sides of the connectors. They have a 60A current rating, which is where the 60 in the name comes from.



(a) Top (b) Front

Figure 7.14: The XT60 connector.

Two parallel wiring harnesses were made, one of which is shown in Figure 7.15. The positive ends are connected to the positive ends, unlike in a series connection. High-current, flexible, 14AWG silicone wire was used. The harness pictured in Figure 7.15 has an additional wire for the 5mm barrel plug for the microcomputer. The

Udoo requires 6V to 15V, and 12V is recommended. Therefore, the Udoo is able to be powered directly from the lithium polymer battery or the power supply.

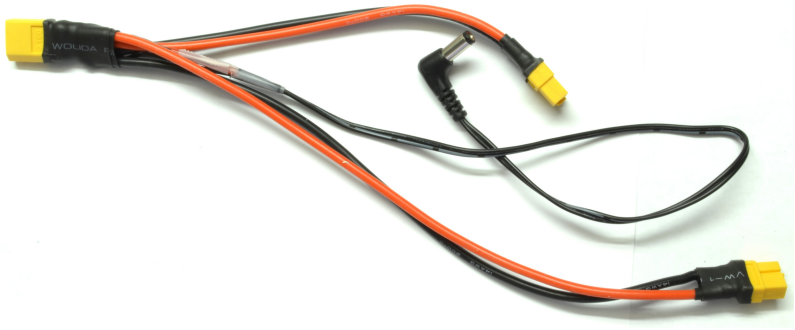


Figure 7.15: A parallel wiring harness.

Additionally, the sensors and control signals must be wired to the microcontroller. The wiring required for the entire system is shown in Figure 7.16. The battery or power supply is shown by the barrel connector at the top right. Both it and the one on the Due are 12V. The ArduIMU is symbolized by the black box above the Due. It is connected to 3.3V power, ground, and its serial output (TX) is connected to the (RX1) pin on the Due. The ultrasonic sensor has similar requirements but also uses a digital pin to enable or disable active sensing. Finally, the four ESCs are wired in parallel to the 12V power, which means they each have 12V power and the total current drawn is a summation of all four, in accordance with Kirchoff's voltage and current laws. As the Arduino Due is compatible with the Arduino Mega, up to 48 servos can be attached on any digital output, but the default pin numbers are 9-12.

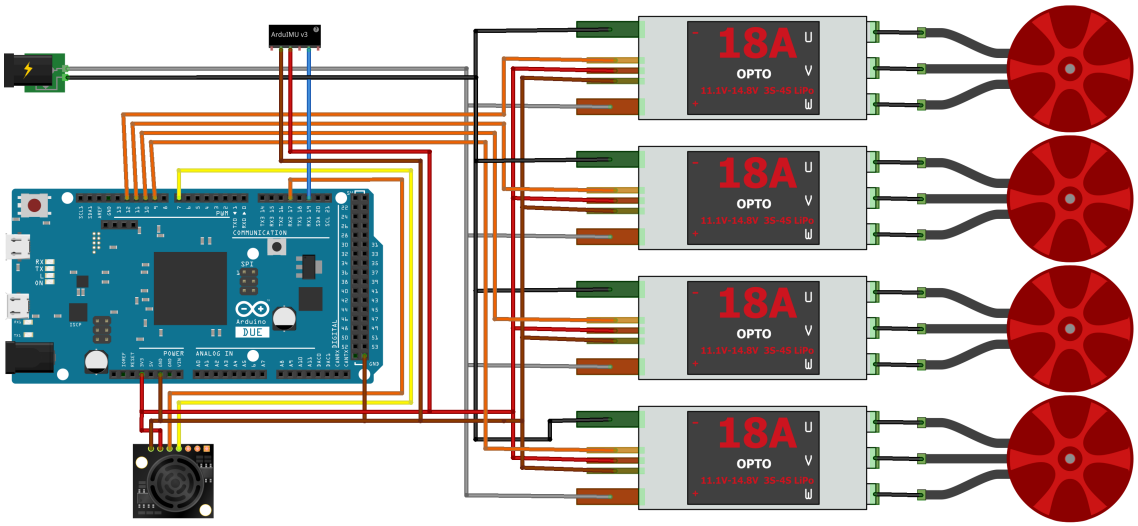


Figure 7.16: The wiring diagram with all of the components connected.

7.4 External Power Supply

While developing the prototype, many tests must be run. The tests need to be quickly terminated remotely and performed in rapid succession. The use of a wired power supply ensures consistent tests, remove the need to wait for the battery to charge, eliminate unnecessary wear on the battery, and create a method to quickly and reliably terminate a test flight remotely. With an external power supply severed, the motors are guaranteed to stop immediately. Using the main wireless communication to terminate flight is potentially unreliable as the software is unstable during the development phase. Finally, an external power supply allows the current usage to be monitored.

A capable power supply is not easily available due to the very high current consumption of the motors, each of which can draw a maximum of 20 amps in bursts or up to 9 amps continuously according to Figure 7.3. Expensive lab power supplies

do not usually deliver more than about 5 amps. Cheaper, adjustable AC to DC power supplies which are designed to drive large quantities of LEDs or a ham radio are available that deliver up to 30 amps. ATX power supplies for desktop computers are readily available for consumers, but only high-end ones would be capable of supplying enough 12 volt power, are somewhat difficult to adapt, and are not easily adjustable. The LED power supplies were chosen because they are variable, easier to use, and cost about the same per amp.

Alternatively, a corded power supply could be made using a lead-acid battery, which is the kind of battery typically used in cars. Lead-acid batteries are much cheaper than lithium-polymer batteries, have a significantly higher amp-hour capacity, and are very heavy. Again, the voltage cannot be finely tuned with the battery option, and maintaining the capacity of the battery was seen as a potential source of testing error, so it was ruled out as an external power supply option.

Figure 7.17: The PSU.

The power supplies used are 12V 30A DC regulated switching power supplies. They can be varied 10.8V to 13.2V using a small potentiometer next to the connectors. Two power supplies are used, each of which drive two motors. Since two motors share the same 30A limit, the power supply is potentially inadequate to supply the maximum current. The maximum current drawn in the test in Figure 7.3 is a 20A burst when the motor goes from stop to full speed. While flying, none of the propellers should be completely stopped. The peak current draw should be closer to the sustained rate of 9A. A 30A power supply is also the largest power supply easily available, and a third cannot be added to evenly distribute between four motors because power supplies cannot be connected to each other due to the difficulty of distributing the load, since minor differences in voltage could cause a power supply

to feed into another. Thus, the power supplies power a separate set of motors and are not connected. So the only option is to add another two power supplies, allowing each power supply to be dedicated to a separate motor, which would add significant cost with potentially no benefit.

Figure 7.18: The relays.

Since DC power supplies have large capacitors, they can remain powered for a few seconds after being disconnected from AC power. Since one of the purposes of the external power supply is to quickly terminate a test flight, the switch should be placed between the DC end of the power supply and the motors. Most switches are not designed to handle 30 amps, and both supplies should be shut down at the same time. Therefore, a low power switch should be used to control an electromechanical relay. When the switch is set to "ON," the relay coil is powered and creates a magnetic field that pulls the contacts in the casing to complete the connection. The relays chosen are shown in Figure 7.18 and are designed for automotive use, so they operate at 12V. They can handle 30A, which is exactly the same limit as the power supplies.

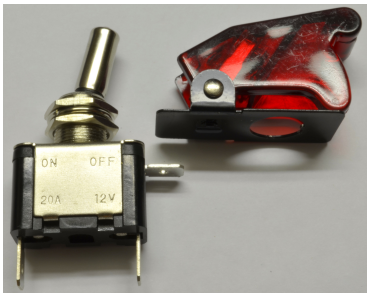


Figure 7.19: Missile switch.

The switch that interfaces with the relays, shown in Figure 7.19, is referred to as a missile switch due to the protective cover that adds another step in order to turn it on and makes it easier to switch off. Due to the dangerous high current of the power supplies, the extra step requiring the cover to be lifted before turning on is a good safety feature.

The system is wired together according to Figure 7.20. To prevent damage to either the power supply or the relays, 30A car fuses are added due to their availability

and because they operate at 12V. Quadrocopter-compatible XT-60 plugs are used on the output for quick connection and disconnection. Large-gauge wires should be used to prevent high resistance and voltage losses due to heat. Additionally, the cables connecting the quadrocopter should be flexible, which means large-gauge fine-strand wire with silicone-insulated wire is ideal. 16AWG wire was chosen because it was on hand and was proven to be adequate at avoiding heat buildup. Higher gauge wire would be ideal due to the very high DC current. With the exception of the XT60 connectors, all of the connections are mechanical, using wire nuts or crimping connections, and not soldered. Additionally, the system should be rigidly mounted due to the complexity of the wiring to allow transportation of the power supply. A plastic case is chosen to reduce the possibility of shorts. Meters, for voltage and amperage, are also be added to the enclosure. The result is shown in Figure 7.21 with two 30A current meters and a volt meter.

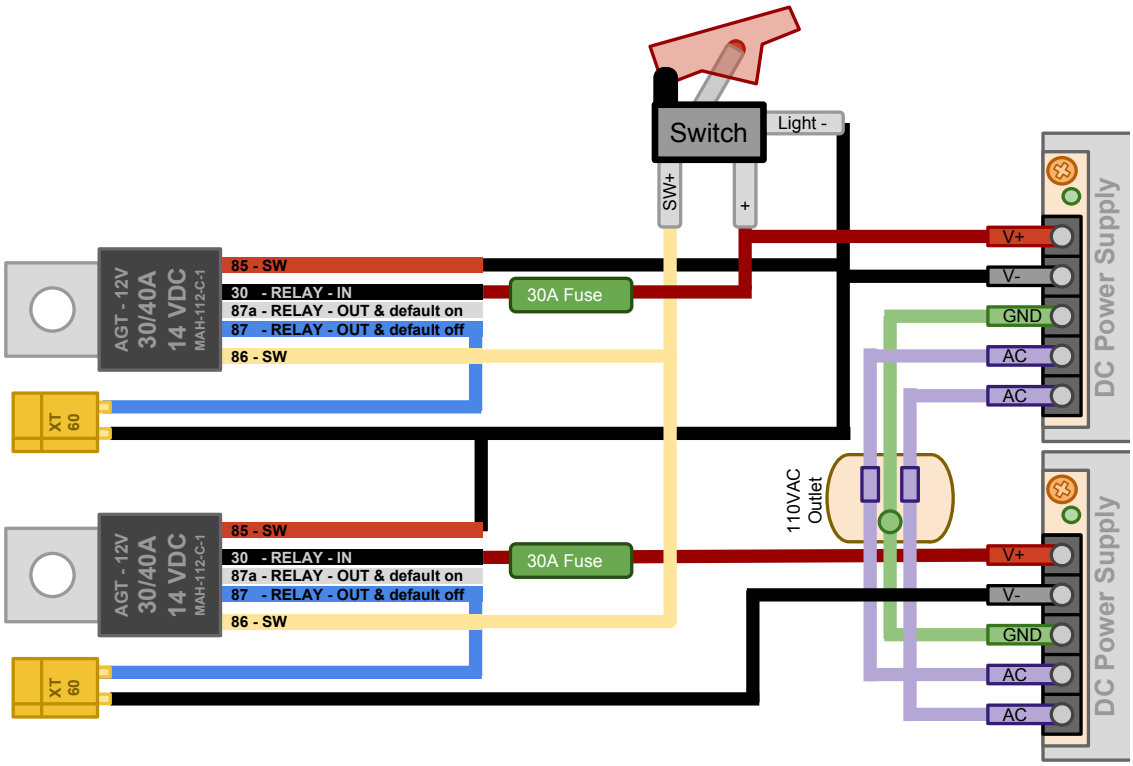


Figure 7.20: The wiring diagram of the power supply unit.

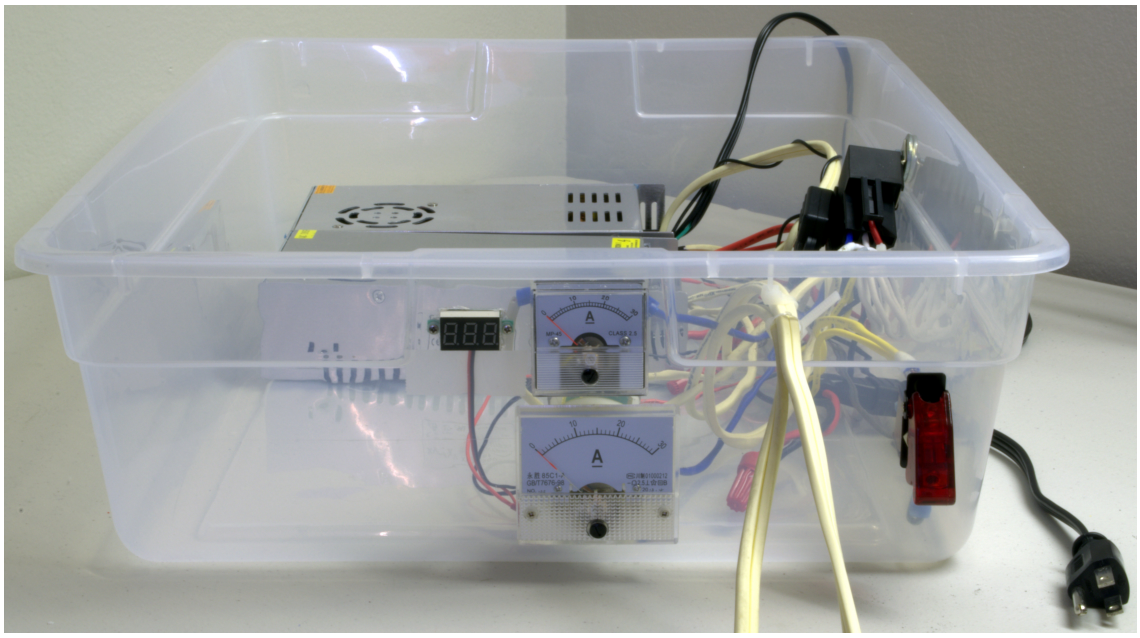


Figure 7.21: The finished power supply unit with panel meters.

CHAPTER 8

PROTOTYPE PROGRAMMING

Whether the quadrocopter is controlled by a pilot, or is completely autonomous, software must be written to achieve flight. At a minimum, a controller must be implemented that utilizes the orientation information from the IMU to stabilize flight and translate. It also maps the three easily described inputs of roll, pitch, and yaw to the four motor outputs. Additionally, both a remotely controlled or autonomous quadrocopter should relay information to a remote location in real time. The flow of data between these components will be described and the usage of that data will then be defined. To first perfect the plain control-related aspects of the system, this prototype will be operator controlled.

8.1 Architecture

The connection layout between the various various devices, with respect to programmable devices and type of connection, is shown in Figure 8.1. The critical connections between the three programmable devices are WiFi, which connects the graphical interface on the desktop client to the Udoo iMX6 Linux-capable processor, and TTL, which connects the iMX6 to its the Due-compatible microcontroller also embedded on the Udoo.

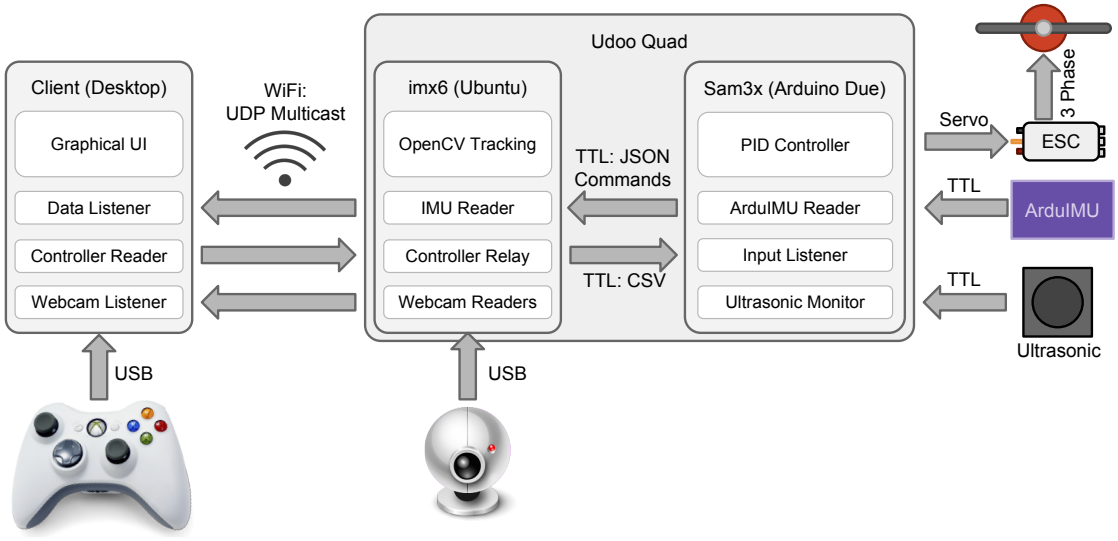


Figure 8.1: Data flow through program architecture and components.

The graphical interface on the desktop client receives all sensor information and displays the relevant information to the pilot. It also reads inputs from a USB gamepad and sends them to the microcontroller.

The microcomputer, driven by the iMX6 processor, is responsible for relaying sensor information from the Arduino Due-compatible microcontroller to the desktop client wirelessly. It also relays control signals from the client to the microcontroller. Finally, it is responsible for reading images from the USB cameras, relaying them to the client, and processing them for information.

The microcontroller, driven by the SAM3X, is where the actual controls take place. It reads sensor information from the IMU, ultrasonic sensors, and the resulting information from the camera tracking on the microcomputer. It then uses this information for a PID-style controller, which sends signals to the ESCs that control the motors.

The Python programming language is used on the client and microcomputer because it is fast and has a variety of libraries like PySerial and OpenCV which

enable simple access to USB devices and image processing respectively. To ensure that most of the program remains responsive despite blocking behavior, each major component within the client and microcomputer is a separate process. An advantage to this, for example, is that while images are being read from webcams, processing can be done on them and the data can be transmitted at the same time. Python has a fast and simple method of inter-process communication (IPC) to enable this, called managers. Managers allow a dictionary data type to be synchronized between processes and are extremely simple to program.

8.2 Device Communication

Communication between devices and sensors are achieved through three different connection types. First, several devices are USB compatible and are managed mostly by the drivers on the operating systems. Second, there is WiFi which connects the client computer with the controller to the microcomputer on the quadrocopter. Lastly, there are several TTL serial connections that connect the microcontroller to the microcomputer and various sensors.

The wireless communication by WiFi uses UDP multicast to deliver information with low latency. TCP packets require additional network and computational overhead. Multicast allows communication to and from the quadrocopter to be achieved without worrying about what the IP addresses of the two devices are on the local network or establishing and maintaining a connection. The images are also sent over WiFi but are not encoded using JSON because the overhead would be far too great. Instead, they are sent in a binary NumPy format. All other communication is encoded in the JSON format. Some examples of JSON-encoded

information is shown below. It consists of a specifier at the beginning that dictates what kind of information is being sent. Then the data can be sent as arrays, integers, decimals, or strings depending on the format of the JSON statement. The advantage to using JSON is that many high-level programming languages, including Python, provide simple ways to serialize variables and parse the packets which robustly checks formatting. The downside to using JSON in this way is that there are clearly many characters that are not entirely necessary. However, these messages only account for a few kilobytes per second of network usage, so the usability for the programmer outweighs the overhead.

```
{"cmd":"euler",   "data":[1.1,-2.5,163.2]}
{"cmd":"AccelNoG","data":[0.1,-0.3,1.0]}
{"cmd":"motors",  "data":[0.9,0.8,0.9,1.0]}
{"cmd":"maxsonar","data":431}
{"debug":"Ignored, but useful for development"}
```

Periodically, the microcontroller also sends sensor and control information to the microcomputer, in the JSON format dictated above, over TTL. New information does not always need to be relayed but should be done at a minimum frequency of every 100 milliseconds to ensure smooth display for the client interface.

Communication between sensors and the microcontroller are done over TTL serial. The ArduIMU is reprogrammed using an FTDI cable to output additional information at a higher speed than the orignal firmware. The baud rate is increased to the maximum transfer rate of 115,200 bits per second. Additionally, the acceleration vector was added to the output so that the microcontroller could use this information. The MaxBotix ultrasonic sensor runs at a baud rate of 9600. Both of these sensors are attached to the serial input pins only, as they are exclusively read. New information is delineated by a newline character '\n'. The IMU must be parsed into several

different arrays. To accomplish this, the `strtod` function is used because it returns the decimal values but also optionally modifies the character pointer argument to show where parsing stops, which should be a comma character between other elements of the euler, acceleration, and angular velocity arrays.

Controller inputs from the microcomputer are also accepted over TTL serial and parsed in a similar way. However, due to the several different types of inputs, each line is prefixed by a command character.

s - Start active control. Allows motor outputs.

k - Kill active control. Immediately disables motors.

c - New control targets. Followed by a comma separated array of percentages for roll, pitch, yaw, and thrust.

8.3 Microcomputer

The programming for the microcomputer is done in Python and broken into several processes. One process receives information from the microcontroller and client, then it relays it to the other device. It is also responsible for reading cameras from USB, which is accomplished with OpenCV from Python. A single frame is read, stored into a manager to be shared between processes, and the array is encoded into a NumPy format and sent over UDP multicast. To reduce network and processor usage, the cameras are read at their minimum resolution of 160 pixels by 120 pixels, which should allow both a higher frame rate and lower latency to display. A single camera can be read at 15 frames per second at this speed.

8.3.1 Camera Reading

To allow the processor load of reading so many cameras to be distributed among the many cores of the Udoo, each camera is run as a separate process. Still, reading multiple USB cameras proves to be extremely problematic - they spontaneously develop an issue where they cannot be read until the system is restarted. To combat this issue, synchronization was added to the processes so that only one camera would be read at a time and read in sequence using a series of locks.

The total data throughput over USB for the two camera reading methods are shown in Figure 8.2. The total activity on the USB connection is measured using WireShark. While the lack of locks temporarily increases the total throughput, after about 15 seconds of operation, one or more cameras in the 3, 4, and 5 tests stop being able to be read. As a result, the throughput drops considerably down to 1 MByte/s when this happens. Tests with artificially high throughputs due to the time-delayed behavior of the loss of the cameras are denoted with an asterisk (*). The test run with all 5 cameras lost access to a camera as the test began. Due to the fact that the tests run with inter-process locks were less likely to lose the ability to read a camera, they are actually faster in the long run.

There are many potential causes for this issue. Since each camera is read in a separate process, the five loads can be distributed accross the four cores of the Udoo. Furthermore, the locks prevent the cameras from being read at the same time. So, it is unlikely that the processor is a bottleneck since the load never increased above 70% of a single core. It is more likely that the bottleneck is the speed of USB 2.0 because the maximum theoretical throughput of USB 2.0 is 35 MBytes/s. If this were the bottleneck, the theoretical limit of the number of cameras on USB 2.0 would be 35.

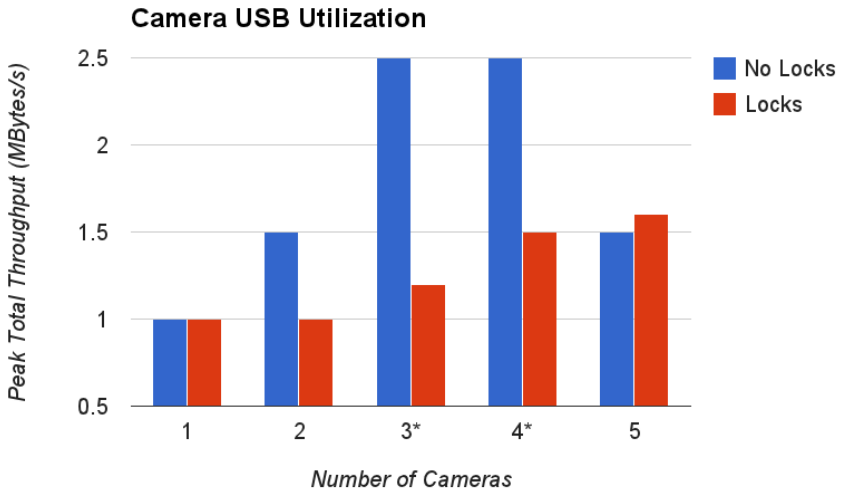


Figure 8.2: The total throughput with different read methods.

Power usage is a considerable limiting factor of USB connections. If the cameras were consuming too much power, the capacitor on the hub might be the cause of the time delay in the failures. However, an externally powered hub was used, which should be able to draw more current than the USB connection on the microcomputer alone. Another possible contributing factor to the failures is the unshielded and manually twisted USB wires, but it does not fully explain why some cameras would actually work for a short period of time. Finally, the issue may be caused by software. The UVC driver may not be fully compatible with these cameras. Also, OpenCV does not have an ideal interface to access multiple cameras because the identifier for the camera must be determined manually before the device is specified. It is possible that OpenCV may not have an implementation of a UVC camera reader that is capable of reading multiple cameras reliably. The current version of OpenCV was downloaded and compiled for the device, and some alterations to reduce delays during failure were made. The Python program written could also be the cause, but there are not many functions to call, using OpenCV exclusively, that can control the camera interface. For example, a start, end, or pause transmission function might be

expected, but there are none. Manual delays between camera reads to reduce frame rates proved to have little effect on the issue as well. The cameras may be sending data whether they are prompted for it or not.

The cameras also proved to be difficult because, even when they work well, the USB and TTL UARTs do not seem to be able to be read from simultaneously. As a result, when a camera is being read, the controller information is missed.

8.4 Client

The client, which runs on a desktop or laptop, is responsible for receiving and displaying information about the quadrocopter to the user as well as sending controller data from a human interface device like a gamepad.

The graphical interface implemented is shown in Figure 8.3. It displays orientation information, shown in green, in a heads-up display style similar to those seen on helicopters. There is a compass, which shows magnetic north and the heading in degrees. Overlaid onto the camera feed, there is a horizon line that acts as a roll and pitch indicator. Additionally, the motor speeds are shown at the bottom as red pie charts. They are in a top-down view that matches the camera displays with the front propeller at the top and back propeller at the bottom. There are also progress-bar-style indicators to show the recency of the information in the top left. The camera feeds are also displayed on screen with recency indicators below them matching the color of leg they represent. The large center feed is the front. The left and right feeds that are center aligned vertically are the left and right views respectively. The bottom feed is the downward-facing camera. In the top right, the

rearview camera would be seen, but it is not working due to the limitations discussed in Section 8.3.1.

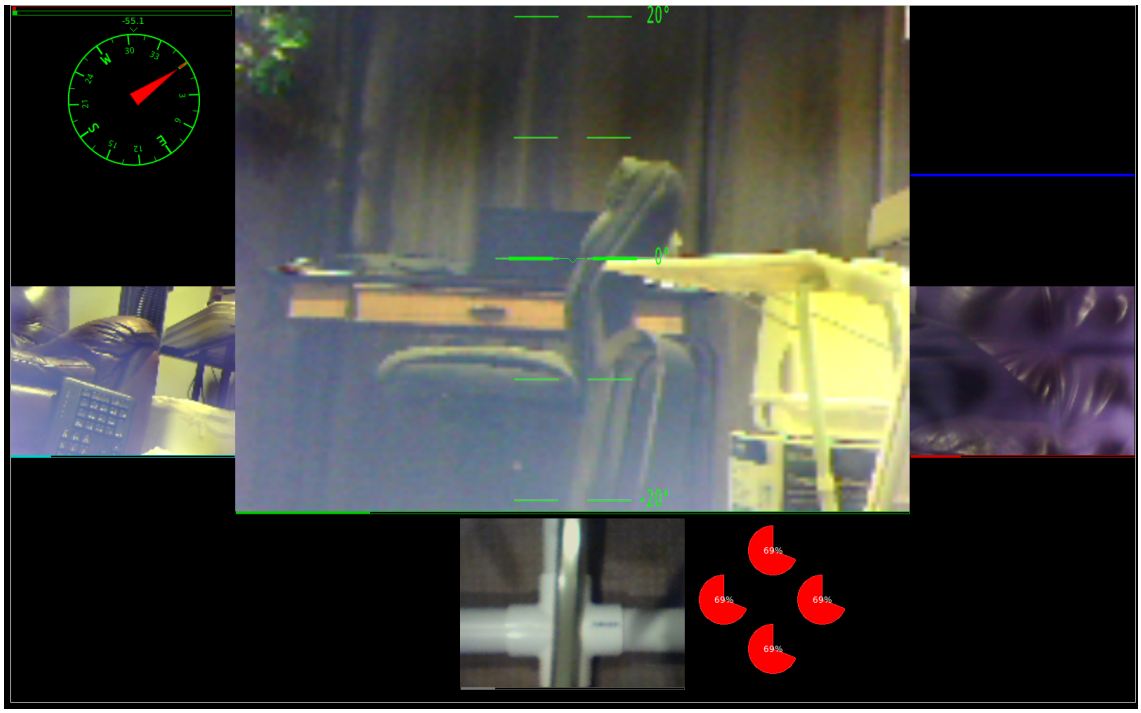


Figure 8.3: The graphical interface on the client with real-time feeds.

The horizon pitch indicator requires calibration specific to the camera - the angle of view. The height of the screen in degrees must be converted into pixel units to convert orientation information for display. Figure 8.4 shows how distances can be measured to determine the vertical angle of view. For objects close to the quadrocopter, the position in degrees may be different because the quadrocopter rotates about the center of mass, which is 16 inches away from the camera. However, the horizon is at an infinite distance away. So the angle should be measured by a triangle starting at the camera. The distance from the center of mass to the camera is negligible for this indicator. The vertical angle of view is found in Eq. (8.1) to be approximately 38 degrees. Therefore, the angular distance from the center to top should be about 19 degrees.

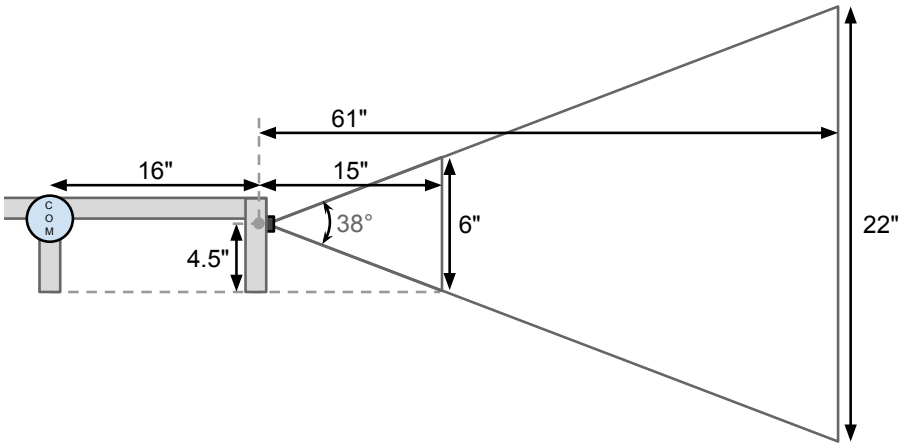


Figure 8.4: Measurements for converting screen height to angle.

$$\operatorname{atan}\left(\frac{6/2}{15}\right)\cdot 2 \approx \operatorname{atan}\left(\frac{22/2}{61}\right)\cdot 2 \approx 38\,\mathrm{deg} \tag{8.1}$$

In processes separate from the GUI, UDP listeners wait for camera and sensor information. Also, there is a process that reads the gamepad and sends the inputs to the microcomputer.

8.5 Microcontroller

The microcontroller, an Arduino Due-compatible SAM3X ARM processor, is programmed using the Arduino IDE with C++. The separate components are shown in Figure 8.5. The controller is updated by the main loop at most once every 10 milliseconds. The limit is in place because the servo signal can only send one pulse every 20 milliseconds. The limit ensures that the ESCs receive new data at the maximum rate possible but does not wastefully calculate controls. Between each

of the loop iterations - some of which may not update the controller - the serial connections are checked to see if there is new data in the buffer.

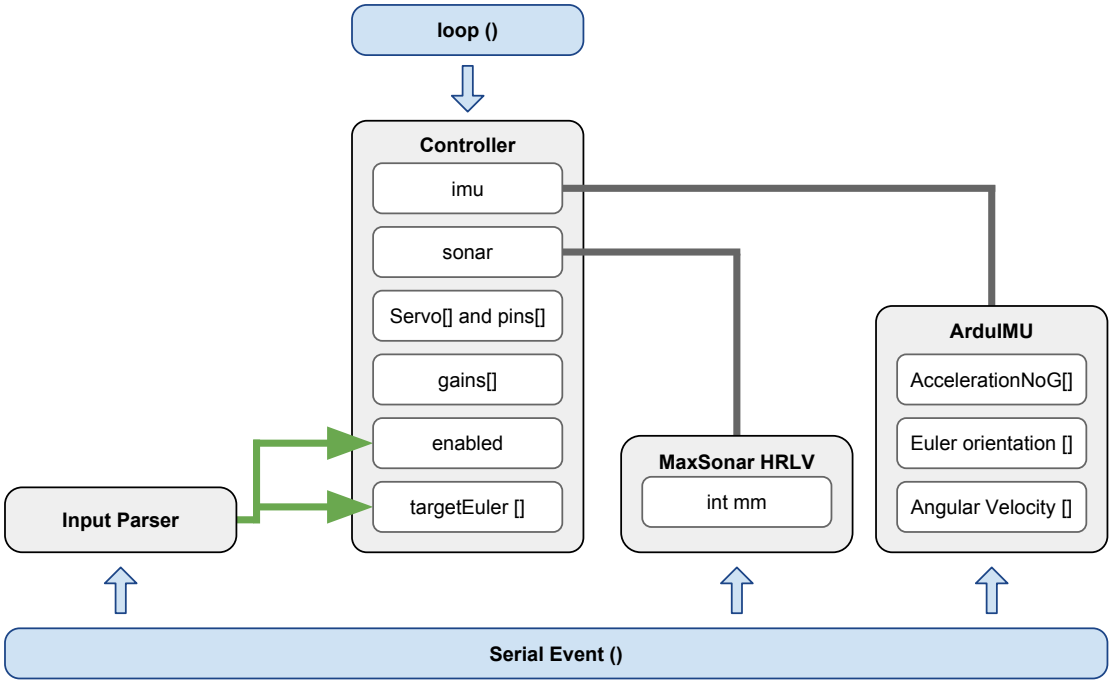


Figure 8.5: Block diagram showing software components of the controller.

8.5.1 PID Controller

A PID-type controller is used to control the orientation. At a minimum, the proportional component of the controller is necessary for responsive corrections. Unlike in the simulation, derivative gain is less necessary because the motors, propellers, and air resistance introduce a speed resistance to the dynamics of the system. Integral correction may become necessary, however, if the center of mass is significantly far from the center of the four motors. To some extent, the correction for this steady-state error can be offloaded to the pilot. Similarly, if a position controller were

added, which would set the inputs for the angular controller, the angular steady-state error would be absorbed into it.

Focusing exclusively on proportional control, the controller for each axis of rotation is shown in Eq. (8.4). Before being used in the controller, the change in angle should be normalized so it uses the smallest positive and negative values to represent angles, shown in Eq. (8.3). This normalization is primarily important for the yaw controller, since the target heading and current heading can be more than 90° apart. Figure 8.6 shows an example of how the angle wrap is beneficial. Without it, the controller would traverse a longer rotational distance.

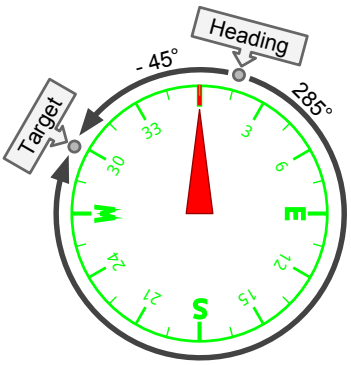


Figure 8.6: Angle wrap.

$$\Delta\theta = \theta_{target} - \theta_{current} \tag{8.2}$$

$$\Delta\theta_{norm} = \begin{cases} \Delta\theta - 360 & \text{if } |\Delta\theta - 360| < |\Delta\theta| \\ \Delta\theta + 360 & \text{if } |\Delta\theta + 360| < |\Delta\theta| \end{cases} \tag{8.3}$$

$$c = k_p(\Delta\theta_{norm}) \tag{8.4}$$

The output of the angular controllers, c, are then converted into speed motor outputs, v, as shown in Eq. (8.6). Due to the dynamics of the quadrocopter, the symmetry of the angular control variables are required in order to create the net torques desired, which are shown in Eq.(8.5).

$$\begin{aligned} M_{roll} &= F_{right} - F_{left} \\ M_{pitch} &= F_{front} - F_{back} \\ M_{yaw} &= F_{front} + F_{back} - F_{left} - F_{right} \end{aligned} \tag{8.5}$$

$$\begin{aligned} v_{front} &= c_{thrust} + c_{pitch} + c_{yaw} \\ v_{right} &= c_{thrust} + c_{roll} - c_{yaw} \\ v_{back} &= c_{thrust} - c_{pitch} + c_{yaw} \\ v_{left} &= c_{thrust} - c_{roll} - c_{yaw} \end{aligned} \tag{8.6}$$

Finally, the motor outputs, which were in percentages from 0 to 1, should be mapped to the range of the servo `writeMicroseconds` function, which is from 1000 to 2000 microseconds.

$$s = 1000 \cdot s + 1000 \tag{8.7}$$

8.5.2 Output Saturation

The outputs, which in this case are the speeds the motors go, are limited. Without careful constraints, the motors may be asked to go faster than they can. If this were to occur, balance would not be able to be maintained. In addition, the simple controller, defined in Eq. (8.6), is prone to making motors spin while landed even if no thrust is supposed to be applied. If $\Delta\theta$ on one axis is non-zero, then only half of the motors needed start to spin, and the other half will receive negative values,

which are interpreted as zero. These constraints cause the moment imparted to be half of what is expected and cause the thrust applied to be different from what is desired.

The first step to solving the issue is to find the maximum control that can be applied. If the thrust desired is no or full, no angular control can be applied. The most angular control that can be applied, then, is the distance from no to full thrust, which is expressed in Eq. 8.8.

$$c_{maxAngle} = \begin{cases} c_{thrust} & \text{if } c_{thrust} < 0.5 \\ 1 - c_{thrust} & \text{else} \end{cases} \tag{8.8}$$

By examining Eq. (8.6), it can be seen that the largest angular control can occur two ways - either through a combination of roll and yaw or by pitch and yaw. Roll and pitch controls never influence the same motor. By reducing the roll-yaw (8.9) and pitch-yaw (8.10) control pairs, by the amount their combination is over the maximum control proportionally, it can be ensured that the motor speeds will never exceed stopped and full speed.

$$\text{if } |c_{roll}| + |c_{yaw}| > c_{maxAngle} \text{ , then: } \begin{cases} c_{roll} = c_{roll} \cdot \dfrac{c_{maxAngle}}{|c_{roll}| + |c_{yaw}|} \\ c_{yaw} = c_{yaw} \cdot \dfrac{c_{maxAngle}}{|c_{roll}| + |c_{yaw}|} \end{cases} \tag{8.9}$$

$$\text{if } |c_{pitch}| + |c_{yaw}| > c_{maxAngle} \text{ , then: } \begin{cases} c_{pitch} = c_{pitch} \cdot \dfrac{c_{maxAngle}}{|c_{pitch}| + |c_{yaw}|} \\ c_{yaw} = c_{yaw} \cdot \dfrac{c_{maxAngle}}{|c_{pitch}| + |c_{yaw}|} \end{cases} \tag{8.10}$$

CHAPTER 9

CONCLUSION

To develop technology for indoor autonomous flight of quadrocopters, several design methods, sensors, and control systems were created and tested. Before construction of a prototype could begin, design and test methods were developed. For example, a system was developed to design a linkage system for a potential manipulator for the quadrocopter to use, which optimizes designs using a modified genetic algorithm. Also, a simulation was developed to test an improved PID control system interfaced with a color grouping system. A new Bluetooth-based positioning sensor was developed to enable autonomous control indoors, since position information was required by the PID control systems in the simulation, then standard sensors were tested such as the IMU and distance sensors. The distance sensors collect information on where solid obstacles are in space, and that data needs to be stored efficiently to be used in real time, so a data structure and algorithm were developed. A safer chassis was designed for indoor flight using CAD software which utilizes the landing legs to protect the propellers. Finally, the prototype was constructed, which is shown in Figure 9.1. The construction required steps to ensure that the weight was as low as possible to allow manuverable flight, and a high-current power supply was compiled to allow rapid testing. The final step in construction was to write software to control the prototype, which consists of multi-process design and PID controllers that handle output saturation.

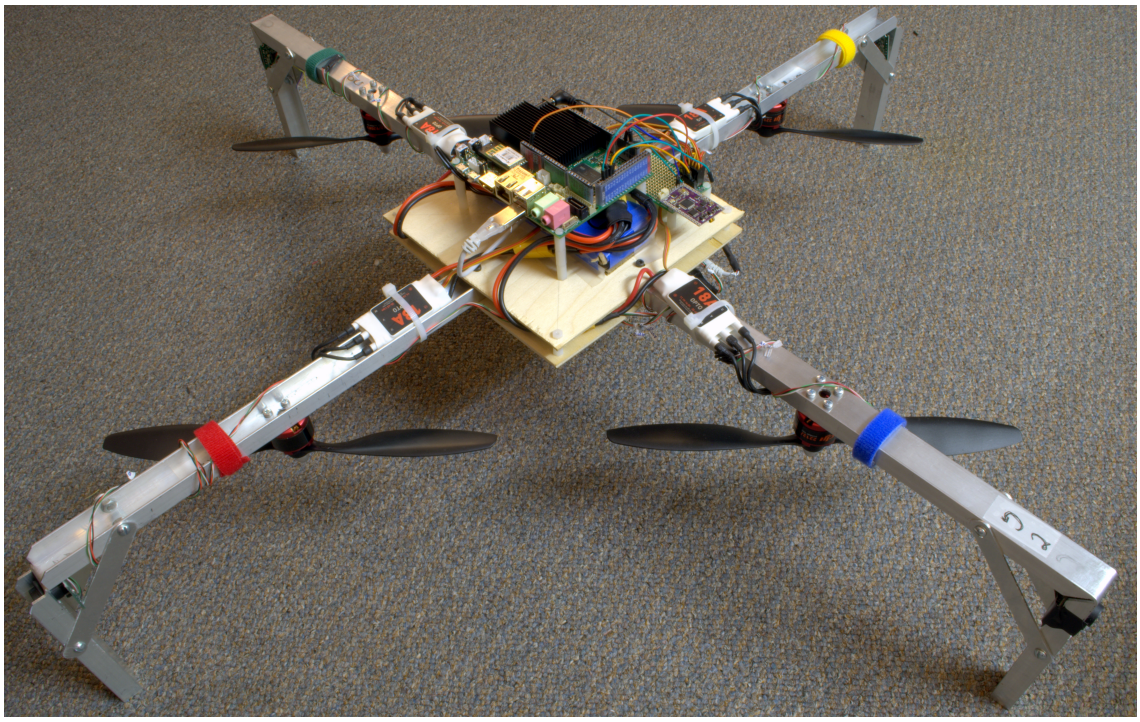


Figure 9.1: The functioning prototype.

This thesis has successfully demonstrated how a quadrocopter, optimized for indoor flight, could possess a lightweight manipulator to pick up objects. The simulation demonstrated how a learning system can automatically dictate how to sort blocks based on color using a neural network. Additionally, a fuzzy-logic modified integral controller was proven to more quickly compensate for vertical steady-state error caused by unknown mass with minimal instability. Since an autonomous quadrocopter requires position information, and absolute positioning is not easily available indoors, one was developed using Bluetooth that was tested to be accurate to about 5 inches. An autonomous path planning system requires that large amounts of information on obstacles be able to be quickly read and modified. The data structure developed to store the obstacle data can find points in 3D space with $O(log_8(n))$ speed. Finally, a custom-built and programmed quadrocopter was proven to fly with a stable controller.

REFERENCES

[1] Michael "SynapticNoise" Graves . "Head, washout and swashplate assembled on main shaft." Flickr. 11 Aug. 2009. CC BY-NC-SA 2.0. Web. http://www.flickr.com/photos/synapticnoise/3814226552/

[2] Forsyth, Robert F., Griffen A. Marr, John I. Nagata, and John R. Burden. "Airworthiness and Flight Characteristics Test: CH-47C Helicopter (Chinook)." Rep. no. AD892400. Edwards Airforce Base, CA: US Army, 1972. Print. http://www.dtic.mil/dtic/tr/fulltext/u2/892400.pdf

[3] "Prime Air." Amazon.com, Inc. 2014. http://www.amazon.com/b?node=8037720011

[4] Federal Aviation Administration. "Fact Sheet - Unmanned Aircraft Systems (UAS)." U.S. Department of Transportation. 6 Jan. 2014. Web. http://www.faa.gov/news/fact_sheets/news_story.cfm?newsId=14153

[5] Federal Aviation Administration. "Unmanned Aircraft (UAS) Questions and Answers." U.S. Department of Transportation. 26. Jul. 2013. Web. http://www.faa.gov/about/initiatives/uas/uas_faq/

[6] "2013 Jerry Sanders Creative Design Competition." *NIU Robotics*. Dr. Ryu, n.d. Web. 13 Mar. 2014. http://www.niurobotics.com/gallery.php.

[7] Resig, John, and Dave Methvin. "jQuery." The jQuery Foundation, 26 Aug. 2006. Web. http://jquery.com/.

[8] Resig, John, and Dave Methvin. "jQuery UI." The jQuery Foundation, Sept. 2007. Web. http://jqueryui.com/.

[9] Witkowski, Pawel. "jQuery Rotate." June 2009. Web. <https://code.google.com/p/jqueryrotate/>.

[10] Baranovskiy, Dmitry. "Raphael." 2008. Web. http://raphaeljs.com/

[11] Ross, Timothy J. *Fuzzy Logic with Engineering Applications: With Engineering Applications*. 3rd ed. Hoboken: Wiley, 2010. Print.

[12] Hartenburg, Richard S., and Jacques Denavit. "Geometric Methods of Synthesis with Three Accuracy Points." *Kinematic Synthesis of Linkages*. New York: McGraw-Hill, 1964. N. pag. Cornell. Web. 14 Oct. 2013. http://ebooks.library.cornell.edu/k/kmoddl/pdf/013_009.pdf.

[13] Hartenburg, Richard S., and Jacques Denavit. "Geometric Methods of Synthesis with Four Accuracy Points." Kinematic Synthesis of Linkages. New York: McGraw-Hill, 1964. N. pag. Cornell. Web. 19 Nov. 2013. http://ebooks.library.cornell.edu/k/kmoddl/pdf/013_010.pdf.

[14] Cabrera, J. A., A. Simon, and M. Prado. "Optimal Synthesis of Mechanisms with Genetic Algorithms." Mechanism and Machine Theory 37.10 (2002): 1165-177. ScienceDirect. Web. 14 Oct. 2013.

[15] Starosta, Roman. "Application of Genetic Algorithm and Fourier Coefficients (GA-FC) in Mechanism Synthesis." Journal of Theoretical and Applied Mechanics 46.2 (2008): 395-412.

[16] Peuuri, F., R. R Pen-Escalante, C. Villanueva, and D. Pech-Oy. "Synthesis of Mechanisms for Single and Hybrid Tasks Using Differential Evolution." Mechanism and Machine Theory 46.10 (2011): 1335-349. Print.

[17] Konak, Abdullah, David W. Coit, and Alice E. Smith. "Multi-objective Optimization Using Genetic Algorithms: A Tutorial." Reliability Engineering & System Safety 91.9 (2006): 992-1007. ScienceDirect. Web. http://ise.rutgers.edu/resource/research_paper/paper_05-008.pdf.

[18] Boldt, Evan. "3D SVG Graph." *Robotic Controls*. 15 Mar. 2013. Web. http://robotic-controls.com/learn/python-guis/3d-svg-graph.

[19] Coley, Gerald, and Robert P.J. Day. "BeagleBone Black System Reference Manual." A6A. Texas Instruments, Dalas, Tx. 2013. https://github.com/CircuitCo/BeagleBone-Black/blob/rev_a6a/BBB_SRM.pdf?raw=true

[20] Kaw, Autar. "Euler's Method for Ordinary Differential Equations." *Holistic Numerical Methods*. University of South Florida, 13 Oct. 2010. Web. http://mathforcollege.com/nm/mws/gen/08ode/mws_gen_ode_txt_euler.pdf.

[21] Kaw, Autar. "Runge-Kutta 4th Order Method for Ordinary Differential Equations." *Holistic Numerical Methods*. University of South Florida, 13 Oct. 2010. Web. http://mathforcollege.com/nm/mws/gen/08ode/mws_gen_ode_txt_runge4th.pdf.

[22] Coller, Brianno. "HumBird Challenge." *Spumone*. Northern Illinois University, Nov. 2012. Web. http://www.spumone.org/courses/control/humbird/.

[23] National Coordination Office for Space-Based Positioning, Navigation, and Timing. "GPS Accuracy." GPS.gov. NOAA, 18 Sept. 2013. Web. 06 Dec. 2013. http://www.gps.gov/systems/gps/performance/accuracy/URE.pdf

[24] Hodgdon, Charles. "Adaptive Frequency Hopping for Reduced Interference between Bluetooth and Wireless LAN." *Design & Reuse. Ericsson Technology Licensing*, May 2003. Web. 08 Dec. 2013. http://www.design-reuse.com/articles/5715.

[25] Bluetooth Low Energy. Tech. LitePoint, n.d. Web. http://litepoint.com/whitepaper/Bluetooth%20Low%20Energy_WhitePaper.pdf

[26] Mistry, Sandeep. "Noble." GitHub. N.p., 22 Nov. 2013. Web. 06 Dec. 2013. https://github.com/sandeepmistry/noble

[27] "Ultrasonic Ranging Module HC-SR04." ElecFreaks, Web. 06 Mar. 2013. http://www.micropik.com/PDF/HCSR04.pdf

[28] Boldt, Evan. "Computer Power Supply." *Robotic Controls*. 19 Feb. 2013. Web. http://robotic-controls.com/learn/projects/computer-power-supply.