

Color Tracking Turret Project - Design Review 2

Arafat Amin,^{*} Ben Chu,[†] Mark Martinez,[‡] Michael Podust,[§] and Stephen Tu[¶]
Dept. of Mechanical Engineering - University of California, Berkeley
(Dated: April 2, 2010)

Design Review 2 written report for our ME102B project, where we describe in more detail some of the details that we left out of the first design review. Specifically, a detail CAD drawing of the turret is presented, along with electrical circuits and code snippets. The goal of this report is to convince the reader that our project team is on track to having a working prototype by the design expo, which we believe is the case.

^{*}arafat@berkeley.edu

[†]benchu@berkeley.edu

[‡]mmartinez213@berkeley.edu

[§]mpodust@berkeley.edu

[¶]stephen_tu@berkeley.edu

I. EXECUTIVE SUMMARY

A. Introduction

Modern day warfare is an increasingly impersonal affair. We seek to investigate the extents to which technology can be applied to weaponry, in order to mitigate the number of military casualties. Specifically, we want to explore the area of image processing with respect to target recognition to see exactly how much human intervention we can remove from the control loop.

B. Goals

Specifically, we propose to prototype a system which provides the following functionality:

1. Tracking of arbitrary color of a target.
2. Ability to control the turret remotely, with a simple user interface.
 - (a) Input of which color to consider as the target.
 - (b) Fire on command.

C. Design

We will seek to design the entire system from the ground up. The components include:

1. Mechanical missile turret armed with electronically powered firing mechanism.
2. Camera mounted to turret for color recognition.
3. Computational server that will be responsible for running the control loops and the color detection algorithms.
4. Interactive server that will interface with the computational server to respond to human commands, in addition to providing a user interface.

D. Proposed Implementation Details

In our first implementation, we propose the following architecture. The turret will have two degrees of freedom, X and Y . There will be two motors controlling both axis independently. These motors will receive PWM signals from an Arduino microcontroller board mounted on the turret. The Arduino board will, in turn, receive control command data from the computational unit via a serial link (or wireless). We will use a simple serialization format for this communication. The camera mounted on the turret will be connected to the computational unit via USB.

The computational unit will be running the algorithm described previously. We will make use of Intel's Open Computer Vision (OpenCV) API to do our image processing.

The interactive unit will expose a user interface to the turret via a web interface. This interface will expose functionality such as changing the color to track, and firing the missile. Such commands will be communicated to the computational unit via either an asynchronous RPC framework if we decide to run the two units on two separate machines, or a UNIX socket if running on the same machine.

E. Challenges

Most of the challenges we foresee deal with the accuracy and sensitivity of our color detection algorithm. Because we are using a simple heuristic model, our algorithm is quite susceptible to changes in the background; it will work best when there is an obvious contrast between the color to track and the background. Also, the color will need to be a solid block. A more complete model would incorporate a combination of heuristics and statistical machine learning models over large data sets to provide a more accurate interpretation of the image.

Another challenge we foresee has to do with the accuracy of the missile. Because we are only detecting color, depth information is not captured very well. Therefore, how we position the gun (how we determine (X_I^*, Y_I^*)) might not be as accurate in terms of missile trajectory). To deal with this, we might take into consideration how bright the recognized color is, and adjust our reference position accordingly. This would probably be aided by switching to do calculations in the hue-saturation-value (HSV) color space (instead of RGB), which OpenCV has built in support for.

CONTENTS

I. Executive Summary	2
A. Introduction	2
B. Goals	2
C. Design	2
D. Proposed Implementation Details	2
E. Challenges	2
II. Introduction	5
A. The Project	5
B. Background Literature	5
III. Specifications	6
A. QFD	6
B. Translation from Customer to Engineering Requirements	6
C. Competitors	6
D. Concept Generation	7
E. Concept Selection	7
1. Why Color Tracking Turret?	7
2. Concept Selection Matrix	8
3. Concept Scoring Matrix	8
IV. Concept Description	8
A. Solidworks Drawing	8
B. Architecture Overview	9
C. Controller	9
1. Controller Serial Wire Format	9
2. Serial Communication Implementation	9
D. Image Processing Unit	10
1. HSV Color Space Algorithm	11
2. Unit Details	11
3. Image Processing Unit Socket Wire Format	11
E. Interactive Unit	11
V. Plan	12
A. Gantt Chart	12
B. RASCI Chart	12
VI. Challenges	12
A. Actuation of Gun with Desired Reaction Times	12
B. Object Detection	12
VII. Parameter Analysis	14
A. Detailed Solidworks Renderings	14
B. Finite Element Analysis on the Gear Shaft	15
C. Gear Analysis	17
D. Detailed Dimensions on Parts to Machine	17
1. Gun Clamp	18
2. Box Top	18
3. Box Front	19
4. Box Side	20
5. Top Motor Platform	20
6. L-Bracket	21
VIII. Final Design	22
A. Block Diagram of System	22
B. Hardware	22
1. Motor	22
2. Circuitry	22
C. Software	22
1. Arduino software	22

	4
2. Image software	24
3. IPC software	26
4. Network Packets	27
5. Testing/Performance of Design	27
D. Cost Analysis	28
IX. Code Repository	28
X. Conclusion	28
XI. Appendix	28
A. Concepts	28
References	29

II. INTRODUCTION

In [1] and [2], we discussed our initial color tracking turret project in detail. In this report, we hope to unify all these details into a comprehensive summary and present this information in a clear, easy to follow manner. We also hope to address some implementation level concerns that readers may have of the project.

A. The Project



FIG. 1: Airsoft Turret

Figure 1 is an illustration of a similar project that we are trying to accomplish [3]. Our project is going to be a color tracking missile turret, which will have either a paintball or BB gun mounted on board, and will be controllable from a remote interface. The target audience for this *kind* of project is military establishments, who seek to minimize the number of human operators in the field. The reasoning behind this is well documented; high military casualties fuel combat aversion among citizens (especially in the United States), which then constrains the military budget and thus places further burdens on the military [4]. While this area of interest is a wide one with a lot of potential projects, we chose to focus on tracking based on image processing because it provides us a technically challenging problem to work on. Furthermore, image processing becomes nearly inevitable when humans are removed from the loop; rarely are basic sensor readings sufficient to provide the information necessary to make the same kind of decisions that humans would make, with any high level of certainty.

Even though our prototype will only contain a toy gun, it is reasonable to expect that a working prototype could be expanded to mount a military grade weapon given some further engineering effort.

B. Background Literature

Arguably the most complex part of the project is the color tracking aspect. However, because we have decided to offload our image processing to a separate computing unit and not perform it on the microcontroller, we are not going to be restricted only to algorithms that run under strict memory and speed constraints.

In [5], it is argued that the HSI color space is the most feasible for autonomous robot situations. Our own experimentation with OpenCV has witnessed similar results, and so most likely we will be working with HSI (or some closely related variant, such as HSV). However, the techniques in [5] are not particular well suited for situations where the surroundings will present a lot of disturbances in the image (such as the target to track will fade away, there will be interference, etc.). Therefore, more complex models based on adaptive Kalman filters have been proposed by [6]. These techniques are considerably more advanced, and would only really be necessary if our more naive techniques perform very poorly.

A color tracking robot is a fairly common project, done at many levels of abstraction. See [7], [8], and [9]. Furthermore, the project is often done as a school project, meaning that it is a tractable semester long project.

III. SPECIFICATIONS

A. QFD

The QFD is shown in Figure 2. First a list of a customer needs were made and ranked from 1 to 10 in terms of priority. The needs were ease of use, unit cost, easy to obtain, reliability and quality, with reliability and the most pressing customer need. Then a list of engineering requirements were made and a relation weight from 1-10 was given. These engineering requirements include: complexity, manufacturing cost, distribution, quality control, precision, reliability and speed. From the values of the relation weights, a target engineering score was given 1-10. Speed and reliability were given the highest target value. Next we compared the how our product addressed the needs of the customer, with our competitor, which is a commercial color tracking device.

Requirement Relations Matrix (1-10)									
		Target	4	6	4	7	8	9	9
		Engineering Requirements	Low Complexity	Low Manufacturing Cost	Distribution and Marketing	Quality Control	Precise Machining	Software Speed	Electronic Reliability
Weight (1-10)	Customer Requirements								
7	Ease of Use		6	1	1	5	5	9	8
7	Cheap Unit Price		10	10	1	3	1	1	1
5	Easy to Obtain		1	1	8	1	1	1	1
9	Reliability		3	2	1	7	8	9	9
8	Quality		1	2	1	6	7	8	8
		Competition Benchmarks (1-10)							
		Our Product							
		Commercial Color Tracking							
			6	5					
			4	9					
			6	8					
			7	6					
			8	7					

FIG. 2: QFD

B. Translation from Customer to Engineering Requirements

1. **Ease of Use** - The customer does not want to have to understand a lot of theory, or have to configure the device very much. Therefore, we believe *low complexity* to be a translation of this requirement.
2. **Cheap Unit Price** - Ideally, the customer would be able to purchase many of these products at a low cost per unit. Therefore, we will require *low manufacturing cost* as an engineering requirement (there is a direct correlation here).
3. **Easy to Obtain** - The product should be readily available to customers. Therefore a distributed marketing system should be in place.
4. **Reliability** - This is probably the most important requirement, since misfires would trigger many consequences and backlash. Thus we will place the highest priority on *precise manufacturing* and *electronic reliability*. This is a trade-off with the previous two requirements, but a very necessary one.
5. **Quality** - This goes hand in hand with the previous requirement. We will focus on *quality control* here as our engineering requirement.

C. Competitors

Our main competitor is the Sentry Project [10]. They offer a complete package which includes the physical turret and all the electrical/software components needed to make it track. However, there are several disadvantages to their offering which we hope to address:

1. **High Cost** - The complete package runs USD 250 for their product.
2. **No Remote Administration** - The administration panel must be directly connected to the sentry. We plan to offer remote administration over the web.
3. **No Modularization** - To use their offering, the user is tied down to their entire stack, from the microcontroller and gun all the way up to the GUI. We plan to modularize our offering, so that components can be easily substituted for one another.

D. Concept Generation

We did not have to go through many concept generation iterations before settling on our final one (the turret presented in detail in this report). The only details that we were uncertain of with respect to the project were what kind of gun to mount on the turret. We finally decided on a paintball gun, which provides a challenge because it is a heavier gun (than say a BB gun). As a backup plan, if we are unable to actuate the paintball gun in a desirable manner, we will resort to mounting a laser pointer, which we believe to be much simpler to implement.

Refer to the Appendix to see our other concepts.

E. Concept Selection

We will describe the advantages and disadvantages of each concept here, and then we will show the scoring/selection matrices. Recall that the Appendix lists the description of each of these concepts (we will omit the description here for conciseness):

1. Projectile Shooter

- (a) **Advantages** - Neat concept, interesting problem to solve.
- (b) **Disadvantages** - Mechanically challenging to implement (above ground shooting apparatus is quite complex).

2. Robot Scripting Language

- (a) **Advantages** - Really hands-on practical project, provides great experience.
- (b) **Disadvantages** - Time consuming. Writing a compiler is not an easy task!

3. Maze Solver

- (a) **Advantages** - Simple algorithm known to solve any simply-connected maze.
- (b) **Disadvantages** - Mechanically complex; lots of hardware needed to build robot.

4. Autonomous RC Car

- (a) **Advantages** - Hardware is easy to obtain; has been done many times.
- (b) **Disadvantages** - Moving base makes any sort of computer vision difficult.

5. Color Tracking Turret

- (a) **Advantages** - Challenging project, involves many aspects of hardware and software.
- (b) **Disadvantages** - Color tracking is a somewhat naive model for object detection; a more robust solution would utilize more complex methods for detection.

1. Why Color Tracking Turret?

We objectively believe that the color tracking turret is the best project proposal for the following reasons:

1. **Mechanical Complexity** - We offer two degrees of freedom ((X, Y) -rotation), in addition to (possibly) having to mechanically actuate the trigger.
2. **Electronic Complexity** - Our system is a closed feedback loop system, so we will require a non-trivial electronic component to control the mechanical actuation.

3. **Software Complexity** - The project offers potential for unbounded complexity (we can always keep improving on our recognition/detection algorithm) in the image processing component, while adding another layer of complexity with the remote interface.

Because this project offers complexity in each of the three fields, we believe that this gives each group member the ability to focus on the area which he is the strongest. None of the other projects listed above are as well balanced in complexity as this project. For instance, the *Robot Scripting Language* project, while tremendously interesting, is more focused on software design than mechanical design (and is probably better suited for a language design course).

Likewise, the *Maze Solver* project can become very complex on the software level if we do not restrict ourselves to simply-connected graphs. More complex graph algorithms would have to be used, which is probably more suited for an algorithms course.

The *Projectile Shooter* project was deemed to be too mechanically complex for the group's liking. Additionally, the problem is not very well defined, and it is not clear what the applications would be.

Finally, the *Autonomous RC Car* project just did not spur much interest from the group, perhaps because it is a problem that has been done many times.

2. Concept Selection Matrix

The Concept Selection Matrix is shown in Table I.

TABLE I: Concept Selection Matrix

Criteria	Datum (Color Tracking Turret)	Projectile Shooter	Robot Language	Maze Solver	RC Car
Mechanical Complexity	0	+	–	0	–
Electrical Complexity	0	+	–	+	–
Software Complexity	0	–	+	–	–
Interest Level	0	–	+	–	–
Cost to Prototype	0	–	0	–	+
Total	0	–1	0	–2	–3

3. Concept Scoring Matrix

We use the Concept Scoring Matrix, shown in Table II to select between the *Color Tracking Turret* and the *Robot Language*.

TABLE II: Concept Scoring Matrix

Criteria	Weight	Datum (Color Tracking Turret)	Robot Language
Group Involvement	1	0	–
Time Commitment	2	0	–
Application	3	0	–
Ease of backup plan	2	0	–
Total		0	–8

IV. CONCEPT DESCRIPTION

A. Solidworks Drawing

A preliminary Solidworks 3D drawing of the missile turret is show in Figure 3. Note that a more detailed and evolved version of this drawing can be found in Figure 9 and Figure 10.

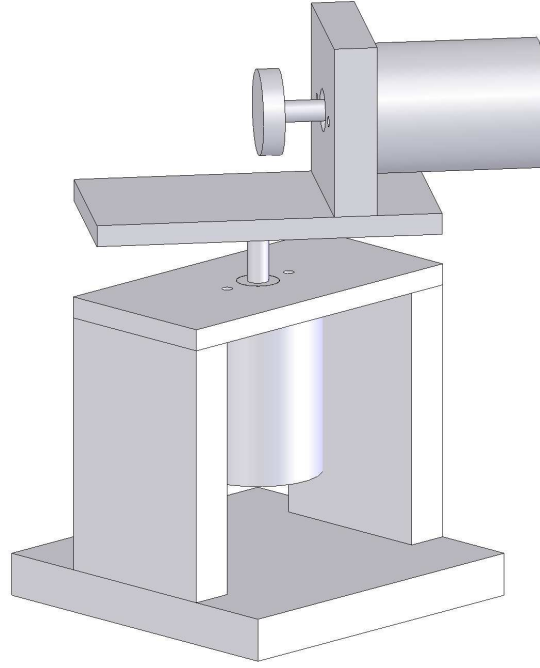


FIG. 3: Color Tracker Turret Solidworks Drawing

B. Architecture Overview

A high level overview of the proposed system architecture is shown in Figure 4. Each of the components in the figure will be addressed in more detail in subsequent sections.

C. Controller

The controller will be a standard PID control algorithm. There will be one PID running for each axis (two of them in total). It will be running off of the Arduino board, sending PWM signals directly to the on-board circuitry of the turret. The controller gains will be hardwired (fine-tuned, of course) into the board, but the reference input and current state information will all come over the serial interface via the wire format described below.

1. Controller Serial Wire Format

Since the controller will be receiving input data from the computational unit via a serial cable, we need to define a wire protocol for this communication. We currently do not believe that we need any handshake protocol to establish communication other than what the serial interface already provides. The packet format is depicted in Figure 5. Note that the *fire* field is 1 byte despite being only a boolean value, since 1 byte is smallest unit of transfer.

2. Serial Communication Implementation

Because the Arduino Serial API allows a programmer to query the number of bytes available to read in the serial receive buffer [11], we can effectively mimic non-blocking IO (even though the serial API is blocking) by simply checking on each loop iteration whether or not the number of bytes available is enough for us to build a packet (or possibly more than one packet). This way we can keep our controller code as simple as possible, to avoid bugs. The only thing we have to worry about is making sure that the rate of packet arrival is no faster than the time interval of each PID controller iteration.

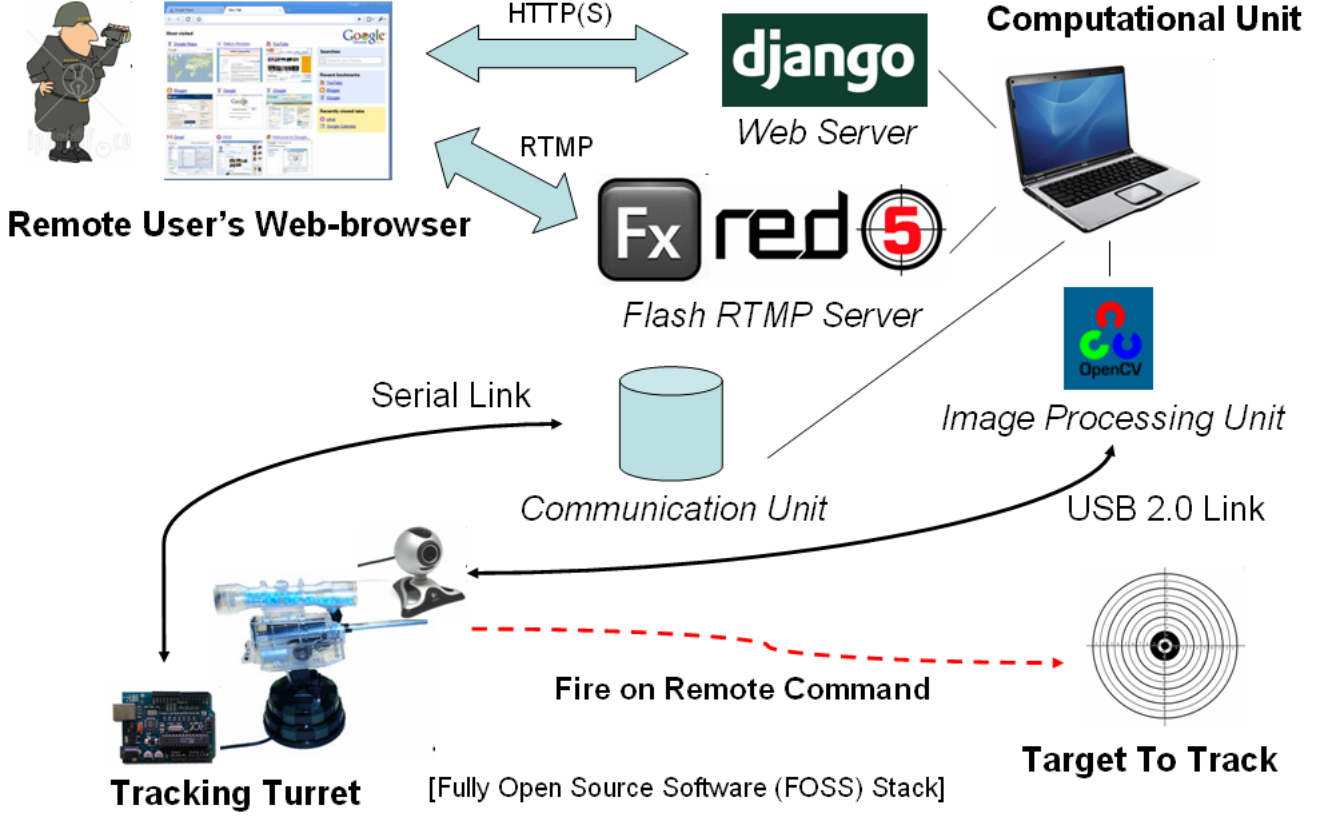


FIG. 4: High Level System Architecture Overview

x_t (2 bytes)	y_t (2 bytes)	x_{des} (2 bytes)	y_{des} (2 bytes)	$fire$ (1 byte)
-----------------	-----------------	---------------------	---------------------	-----------------

FIG. 5: 9-byte serial packet format

D. Image Processing Unit

The image processing unit will monitor a camera (web-cam) that is attached to the turret. The behavior of this unit is described as follows:

Algorithm 1 ImageProcessingUnit($(R_d, G_d, B_d), x_{des}, y_{des}$)

- 1: Let $(H_d, S_d, V_d) := \text{ConvertToHSV}((R_d, G_d, B_d))$
 - 2: **while** true **do**
 - 3: Capture (R_i, G_i, B_i) values for each pixel from camera
 - 4: Let $(H_i, S_i, V_i) := \text{ConvertToHSV}((R_d, G_d, B_d))$
 - 5: Process all events in the events to process buffer
 - 6: Let $I :=$ converted black/white image based on heuristic function (inputs are (H_d, G_d, B_d) and (H_i, G_i, B_i))
 - 7: Let $(X_I, Y_I) :=$ spatial centroid of I
 - 8: Run PID on X_I and Y_I such that $(X_I, Y_I) \rightarrow (X_I^*, Y_I^*)$
 - 9: Let $S :=$ serialize $x = (X_I, Y_I)$ and $r = (x_{des}, y_{des})$ into the wire format
 - 10: Send S to serial port (to controller)
 - 11: **end while**
-

The spatial centroid (X_I, Y_I) is defined as follows:

$$X_I = \frac{M_{10}}{M_{00}} \quad (1)$$

$$Y_I = \frac{M_{01}}{M_{00}} \quad (2)$$

$$M_{mn} = \sum_j \sum_k x_k^m y_j^n P_{j,k} \quad (3)$$

Where (x_j, y_k) is a pixel location, $P_{j,k}$ is a pixel value, and m and n are the order of the moment[12]. As for the image heuristic function, we have settled on using an HSV color space algorithm.

1. HSV Color Space Algorithm

Algorithm 2 HSVColorSpace($(H_d, G_d, B_d), (H_i, G_i, B_i)$)

1: **return** $|H_d - H_i| < H_{TOL}$ and $|S_d - S_i| < S_{TOL}$

During the tuning phase, instead of using constants for H_{TOL} and S_{TOL} , we can make these tolerances instead a piecewise step function based on the environment.

The conversion from RGB to HSV color space is described in [13].

2. Unit Details

This unit will run on a Linux x86 machine, and ideally we want this unit to be as fast as possible (for best performance). We will use the OpenCV library to do our image processing work, and we will use the C version for best performance. To query for input from the interactive unit, this unit will listen on a either a UNIX or TCP socket, depending on whether or not the interactive unit runs on the same machine. In order to not hinder with the main image loop, we will listen on the socket in a background thread. To implement this, we will probably use the standard `pthread` library.

3. Image Processing Unit Socket Wire Format

The wire format for this unit is shown in Figure 6. Here, the input from the interactive unit is simply the RGB value of the color we want to track, and a field indicating whether or not we want to fire. Once again, we do not believe we need a handshake protocol here either (on top of what is already provided).

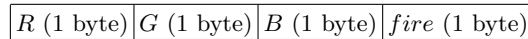


FIG. 6: 4-byte socket packet format

E. Interactive Unit

This unit is the side that faces the user(s). We will display a video of what the turret currently sees, along with a button to press when the user wants to fire, and the ability to select the color of the target. We will probably implement this as a website, and use Adobe Flash technology to stream video to the site. The Adobe Flex SDK is free to use now, and there are working open source implementations of a Flash server [14]. As for the website, we will probably make use of the Django [15] web framework, and implement as many of the features we can using AJAX. By implementing a website, it should be in theory possible to see through our turret's camera, and control it from anywhere in the world where there is internet (obviously we will put security measures in place). However, this definitely realizes our goal of removing humans out of the loop as much as possible.

This unit will relay user input commands by opening a UNIX or TCP socket, and sending the command via the packet format described in Figure 6.

V. PLAN

A. Gantt Chart

The Gantt chart is shown in Figure 7.

B. RASCI Chart

The RASCI chart is shown in Figure 8. The acronyms are as follows (listed in decreasing order of responsibility):

1. **RA** - Responsible
2. **A** - Accountable
3. **S** - Support
4. **C** - Consulted
5. **I** - Informed

VI. CHALLENGES

There are several challenges that we anticipate throughout the semester.

A. Actuation of Gun with Desired Reaction Times

A paintball gun, we believe, is potentially too heavily to actuate in a responsive manner, using the hardware provided. We are investigating upgrading to a more powerful motor and an H-Bridge that is able to handle more current. If this proves to be too much of an issue, then we will restrict ourselves to a lighter gun, until we can get the response times we desire.

B. Object Detection

We are currently working on implementing object detection, in addition to simply color recognition, in our image processing unit. We have so far been able to leverage the built in OpenCV Haar classifier algorithms to perform real-time facial recognition. The idea here is that we can train these classifiers to recognize any arbitrary object, so we will no longer be restricted to a single color.

The challenge here is that the real-time performance of the Haar classifier algorithm is still too slow to use in this kind of application. We will be investigating several methods we can use to optimize the performance so we can have the desired real-time performance characteristics.

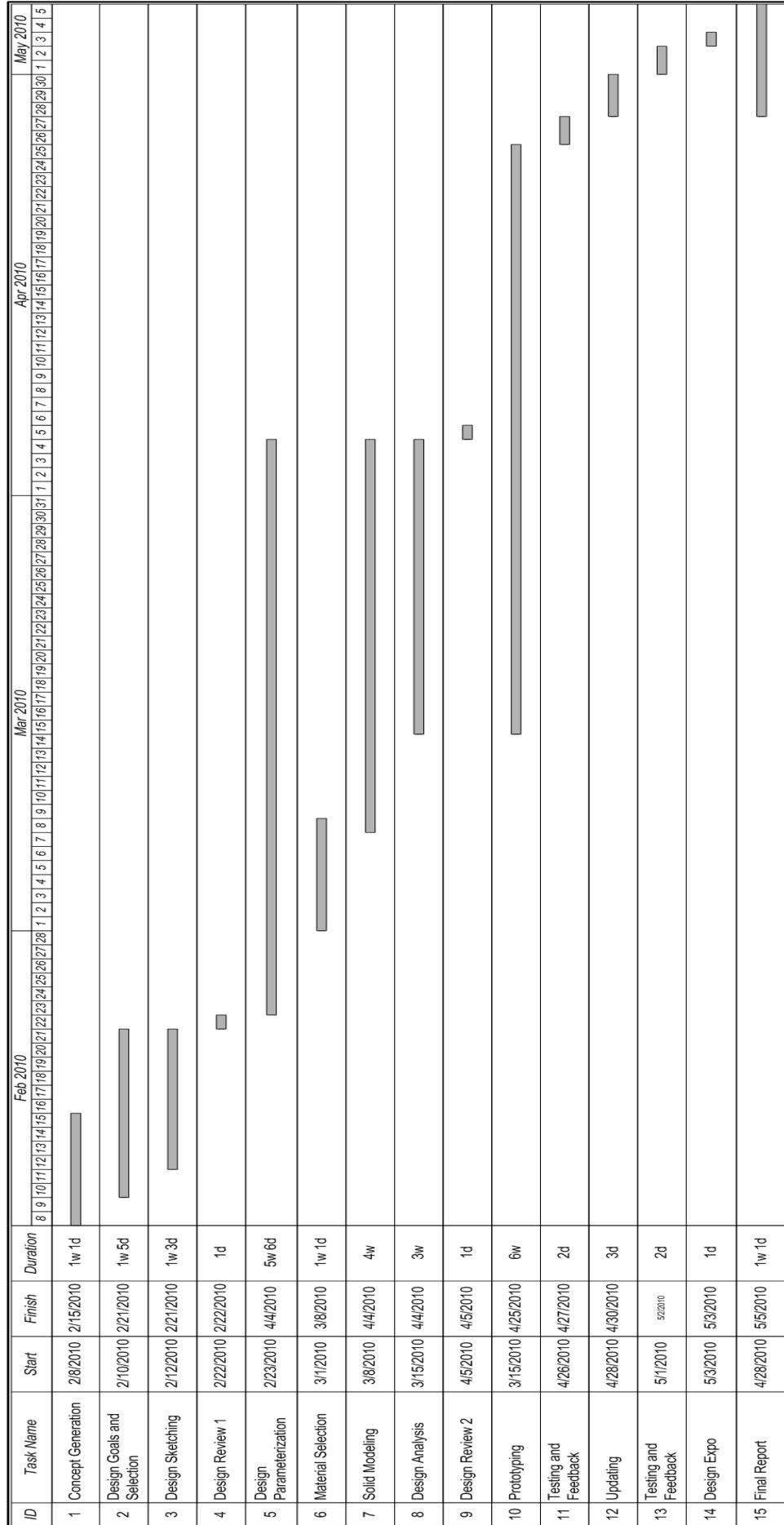


FIG. 7: Gantt Chart

Software	Arafat Amim	Ben Chu	Mark Martinez	Mikhail Podust	Stephen Tu
Development & Optimization	I	S	I	S	RA
System Design (CAD)	R	S	RA	S	I
Prototyping & Building	RA	S	S	R	C
Alpha Testing	S	RA	S	C	A
Beta Testing	S	S	S	RA	S

FIG. 8: RASCI Chart

VII. PARAMETER ANALYSIS

A. Detailed Solidworks Renderings

More detail CAD drawings of the turret are now available in Figure 9 and in Figure 10.



FIG. 9: Turret View One

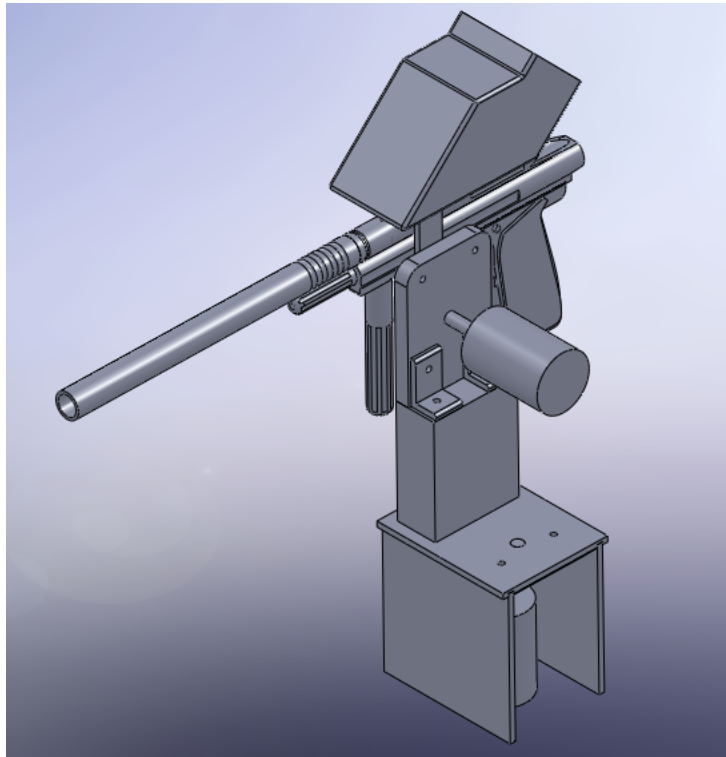


FIG. 10: Turret View Two

B. Finite Element Analysis on the Gear Shaft

We ran two finite element simulation on both the gear shafts to calculate the factor of safety of our turret. In our simulations, we assumed that the gun, webcam, and camera contributed a 5-pound load in total. We also assumed that the gear shaft was made from 1020 Steel. Thus, it has a known yield stress σ_{yp} which can be used in the FEA simulations. Figure 11 and Figure 12 show the results of our simulation.

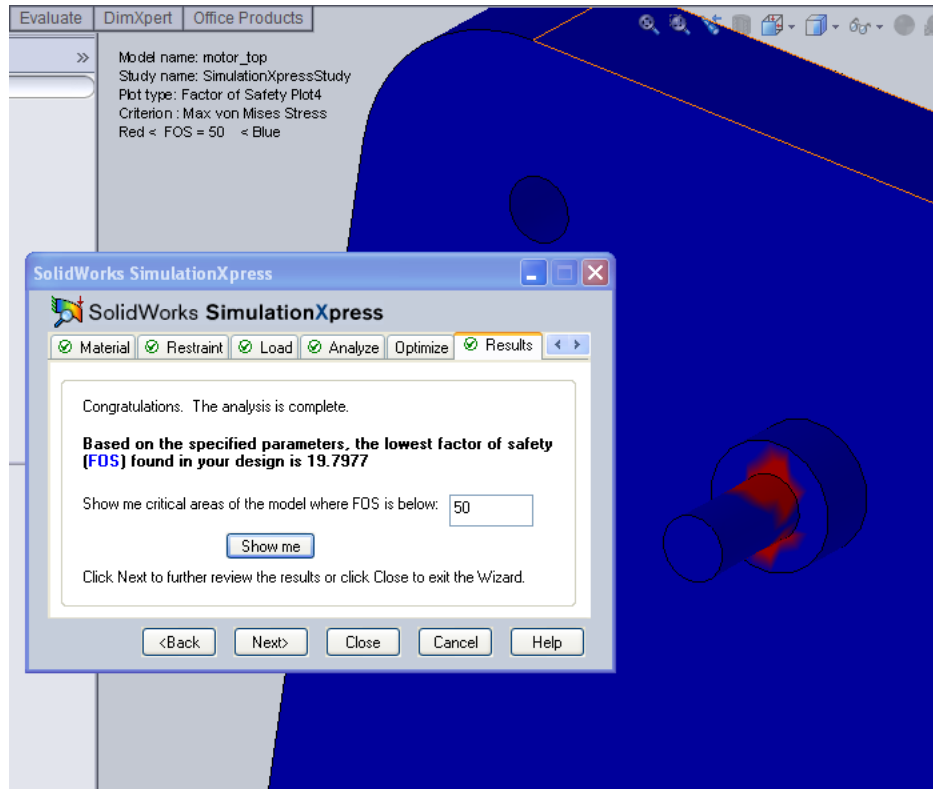


FIG. 11: Finite Element Simulation on the Pitch Motor

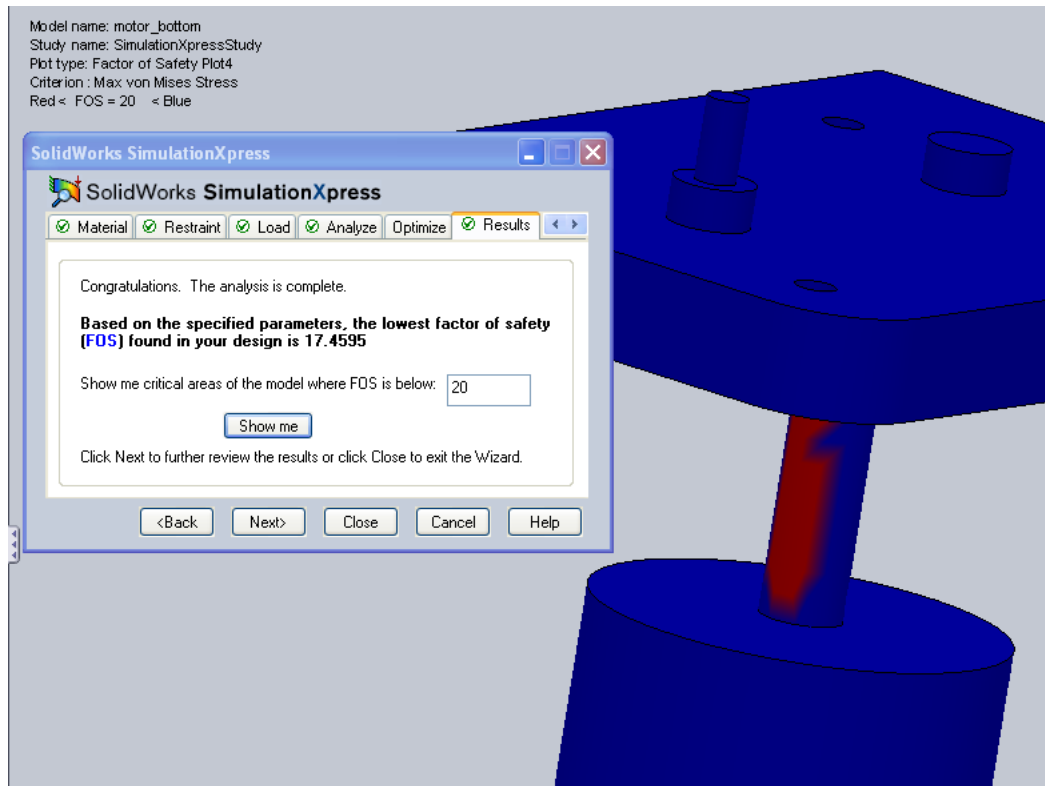


FIG. 12: Finite Element Simulation on the Yaw Motor

The FEA simulations are based off of the von Mises yield criterion. Recall that this is given as:

$$(\sigma_{11} - \sigma_{22})^2 + (\sigma_{22} - \sigma_{33})^2 + (\sigma_{11} - \sigma_{33})^2 + 6(\sigma_{23}^2 + \sigma_{31}^2 + \sigma_{12}^2) = 2\sigma_y^2 \quad (4)$$

where the σ_{ij} are the i, j -th elements of the stress tensor $\bar{\sigma}$.

Thus from this von Mises analysis, the factor of safety for the gun is around 17.5 given our assumptions. We are here implicitly assuming that the gear shaft is the weakest point, but that is a reasonable assumption since it must bear the most load during operation. Thus our design is a safe design, in terms of yielding.

C. Gear Analysis

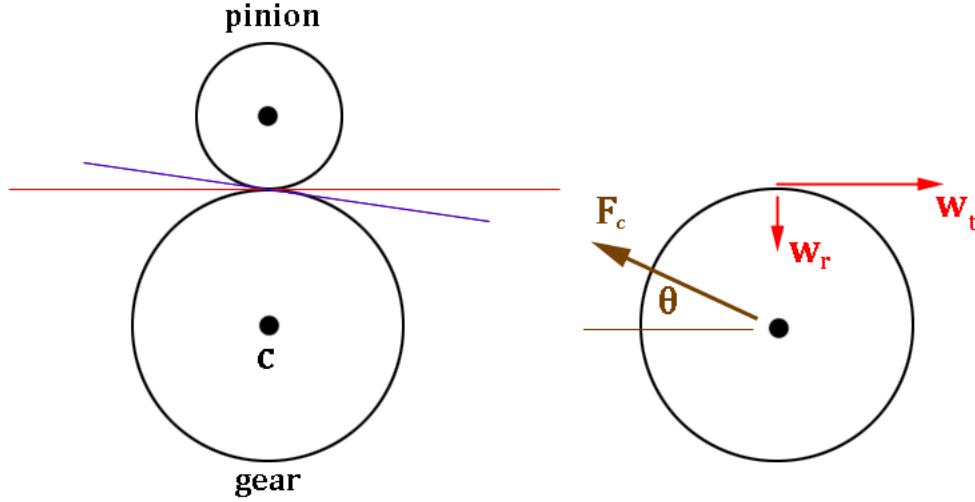


FIG. 13: Simple Gearbox Model

To calculate the force on the yaw base motor, we performed some simple gear analysis. This motor, which we got from lab, has a 44:1 ratio. Here are the assumptions we made:

1. The motor shaft is operating at 20 RPM at 1 watt ($= 1.341 \times 10^{-3} hp$).
2. There are only two gears in the gear box.
3. The pressure line of both gears is $\theta = 20$ degrees.
4. $d_g = 4in$ and $d_p = 1in$, as shown in Figure 13.

From lecture we know that:

$$HP = \frac{\bar{\omega}_t v_i}{33000} \quad (5)$$

$$v_i = \frac{\pi d_i n_i}{12} \quad (6)$$

Plugging the numbers in (for the motor shaft c), we get that:

$$\bar{\omega}_t = 0.1921 lb \quad (7)$$

$$\bar{\omega}_r = 0.0699 lb \quad (8)$$

$$\bar{\omega} = 0.2044 lb \quad (9)$$

So the motor shaft has a force of $0.2044 lb$ acting on it during normal operations.

D. Detailed Dimensions on Parts to Machine

In this section, we present dimensions for all the parts that we will be machining in the machine shop. The dimensions of these parts are based off of the dimension of the motor picked from lab, except the gun clamp which is dimensioned based on the size of the paintball gun. Ideally, the gun clamp will hold onto the gun at its center of mass, so that the least amount of torque will be needed by the motor to rotate the gun.

1. Gun Clamp

The gun clamp is show in Figure 14.

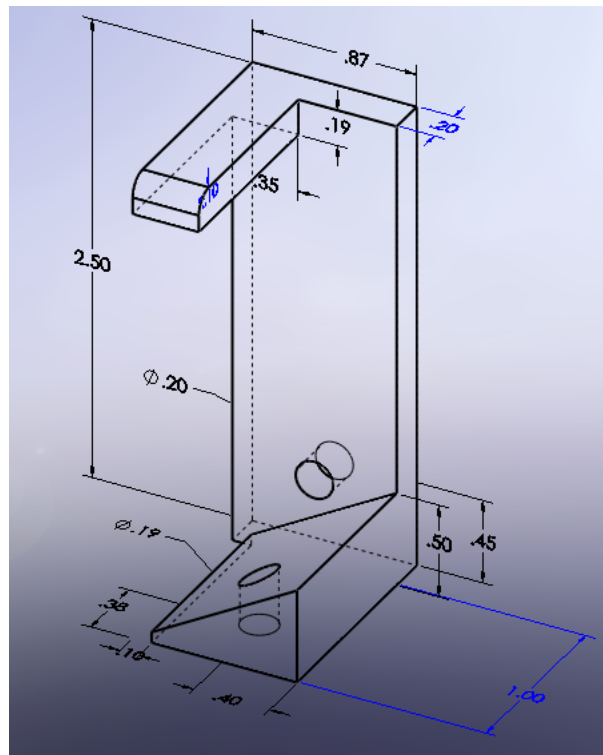


FIG. 14: Gun Clamp

2. Box Top

The box top is show in Figure 15.

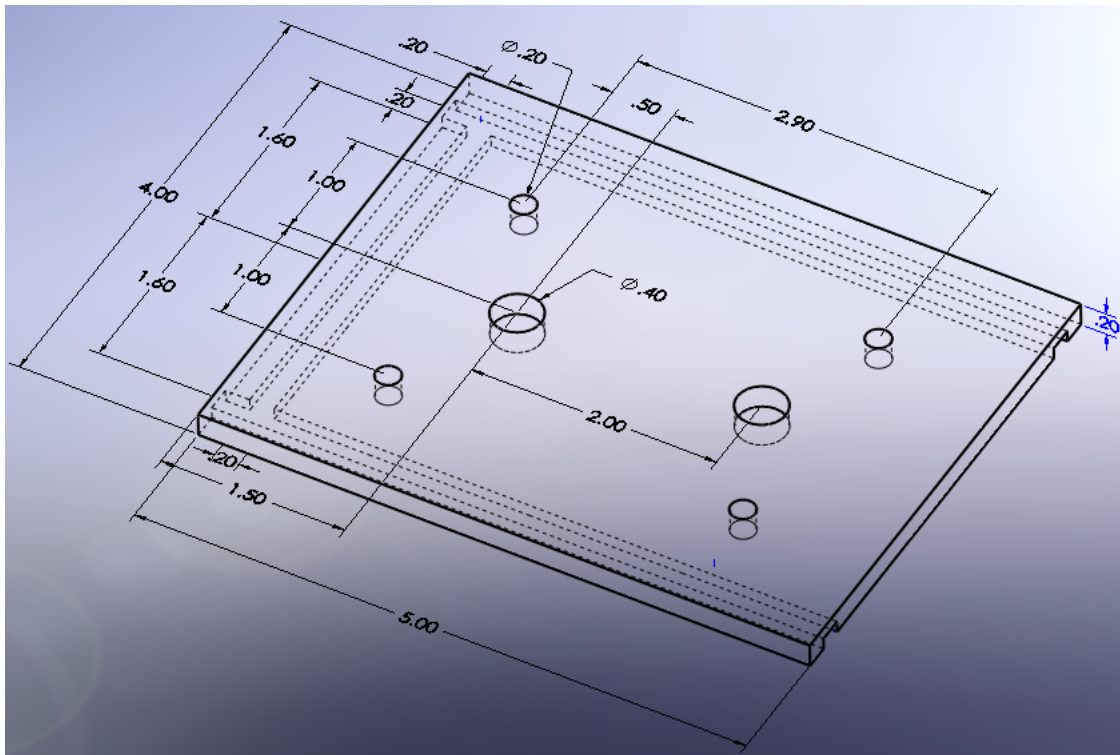


FIG. 15: Box Top

3. Box Front

The box front is show in Figure 16.

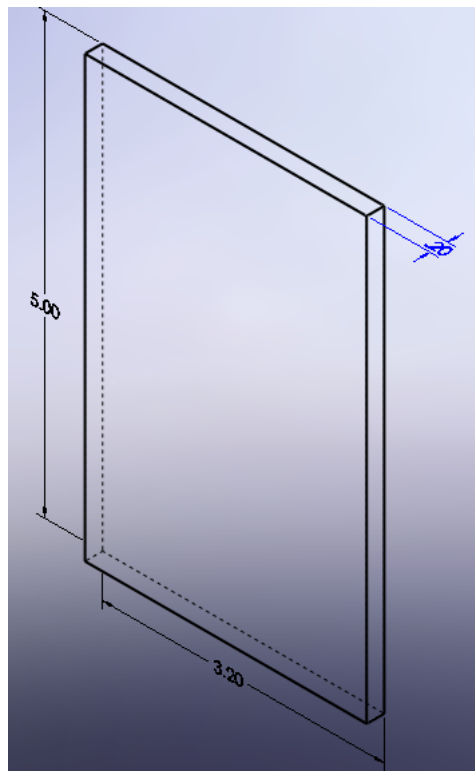


FIG. 16: Box Front

4. *Box Side*

The box side is show in Figure 17.

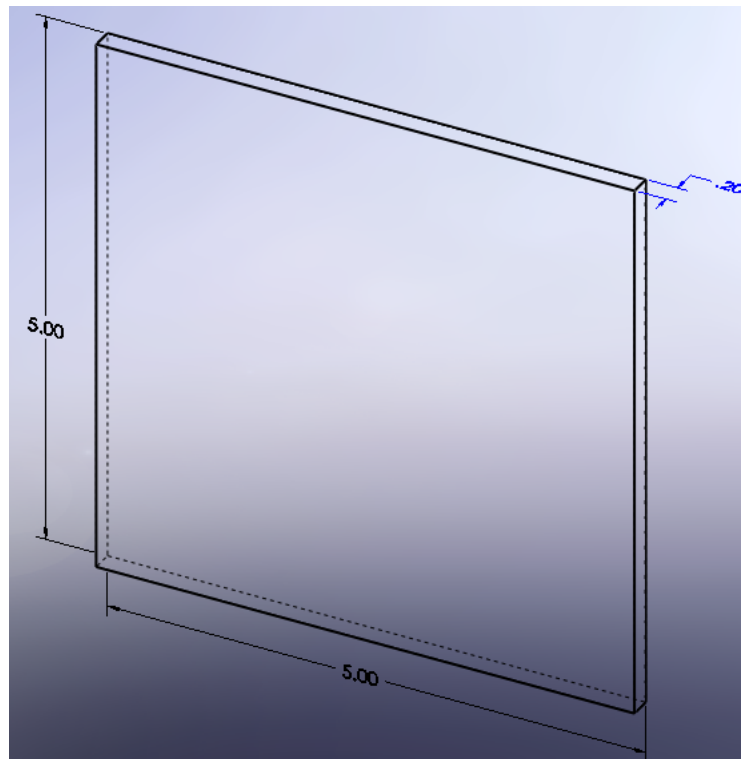


FIG. 17: Box Side

5. *Top Motor Platform*

The top motor platform is show in Figure 18.

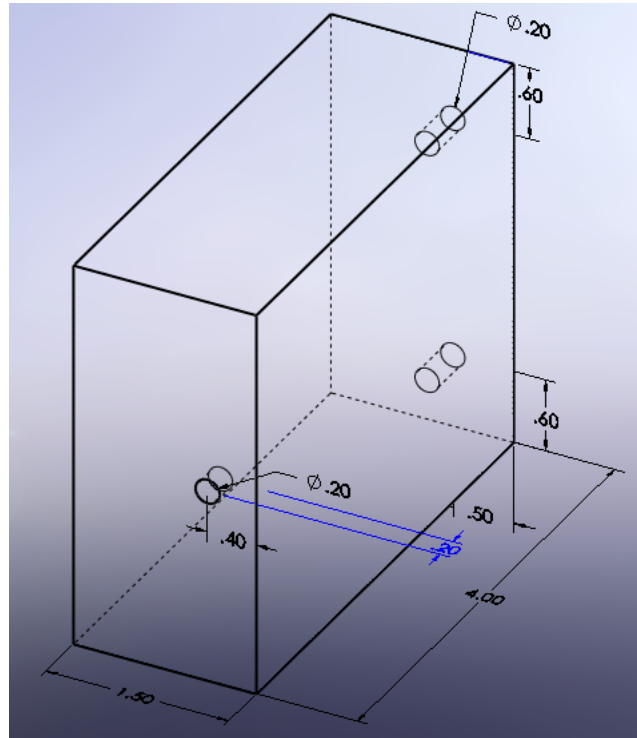


FIG. 18: Top Motor Platform

6. L-Bracket

The L-bracket is show in Figure 19.

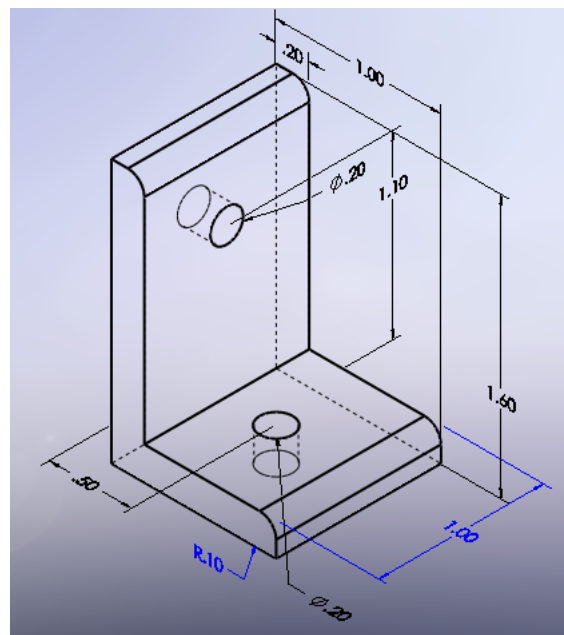


FIG. 19: L-Bracket

VIII. FINAL DESIGN

A. Block Diagram of System

Figure 20 shows a high level block diagram depiction of our system. This is representative of the design shown in Figure 4, except in block diagram form.

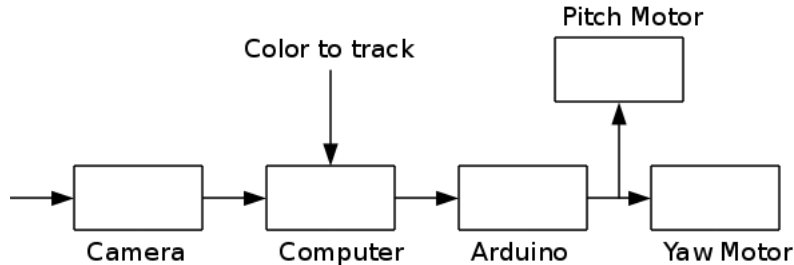


FIG. 20: Block Diagram Description of System

B. Hardware

1. Motor

The base motor has a 44:1 gear box ratio. The other motor will have a 10:1 gear box ratio. The reasoning for this is that, the base motor has to be able to support more weight (the weight of the entire platform containing the gun, etc.), so it needs to be able to provide high torque. The downside to this is that the size of the motor becomes larger, but since this is our base motor, that is not an issue. Contrast this with our other motor, which will be responsible for controlling the pitch of the gun. This motor cannot be very heavy, since it will need to be supported by the base (yaw) motor. Thus, as mentioned above, we will use a smaller motor, but we will clamp the gun at its center of gravity so that it will not require as much torque to rotate the pitch of the gun.

We were able to acquire both of these motors from the lab.

2. Circuitry

The design and specification of our hardware circuitry is shown in Figure 21. Yaw and pitch motors correspond to the two degrees of freedom our turret has. All outputs from the Arduino are run through a digital buffer before connecting to the H-Bridge for board safety reasons.

In our design, we have assumed that the high voltage rail runs at 9V. This is a safe assumption that can be realized by placing a few off the shelf batteries in series to get the desired voltage.

In this diagram, we have also shown the mechanism which we will use to fire the gun. This is implemented with a solenoid and a relay.

The H-Bridges in this circuit will be the National Semiconductor LMD18200 H-Bridge, and the buffers will be the Texas Instrument SN7417 non-inverting hex buffer. The relay switch will be an NTE R40-11D2-5 relay.

C. Software

1. Arduino software

Implementing a PID controller in Arduino is quite straightforward. The code goes something like (this is only for one direction, but it is easy to imagine how another axis would work):

```

double t_prev;
double err_prev;
double x_des;
// code to set up board [...]
while (1) {

```

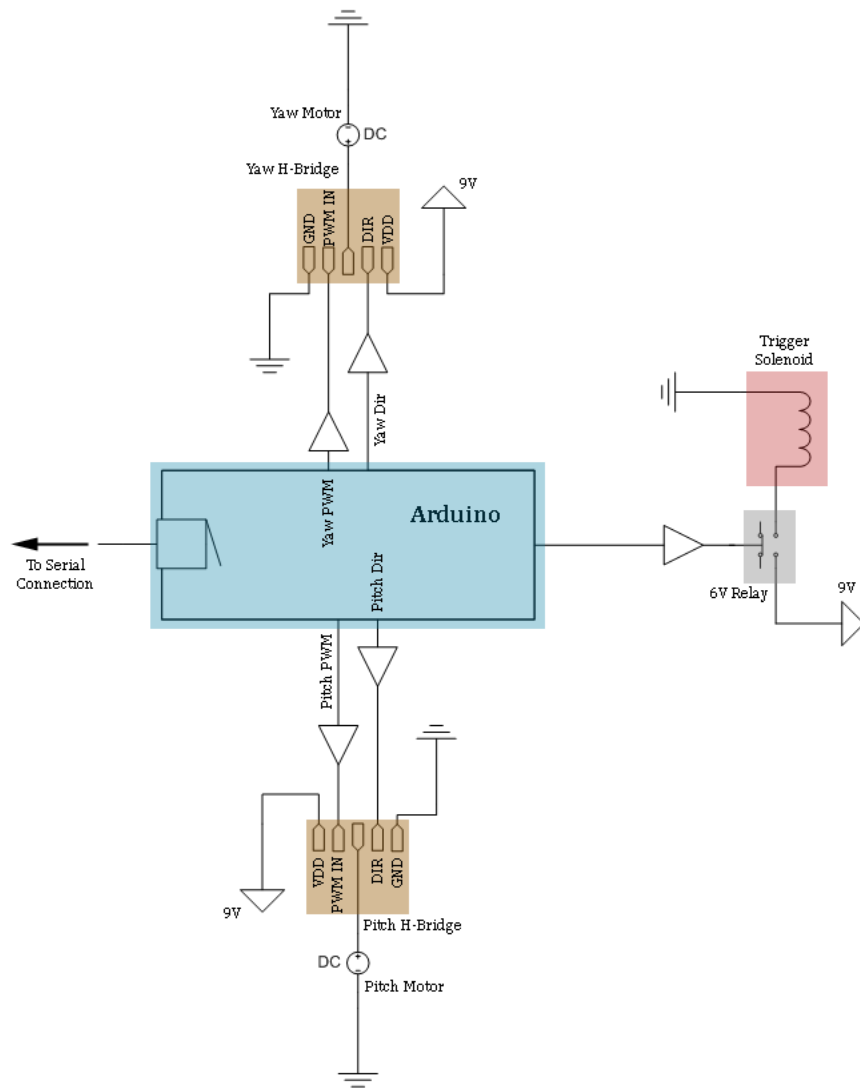


FIG. 21: Design of electronic circuitry, showing in detail how the Arduino board will interact with the motors.

```
double x_cur = getCurrentXPosition();
double err_cur = x_cur-x_des;

unsigned long t_cur = millis();
unsigned long dt = t_cur-t_prev;
double P = Kp*err_cur;
double D = Kd*((err_cur-err_prev)/dt);
double I = Ki*(I+err_cur*dt);
double out=P+I+D;

// write out to pwm [...]
}
```

The software to get the current position looks something like:

```
// at the top
#include <stdint.h> /** For uint32_t */
#include <stdio.h> /** For memcpy */

#define BUFSIZE 128

typedef struct {
```

```

    uint32_t x;
    uint32_t y;
} Position;
Position cur_pos;
char      buf[BUFSIZE];

void updatePosition() {
    int msg_size = sizeof(Position);
    if (Serial.available() >= msg_size) {
        int bytes_remaining = msg_size;
        int bytes_read = 0;
        while (bytes_remaining--) {
            buf[bytes_read++] = (char) Serial.read();
        }
        memcpy ((char*) &cur_pos, buf, msg_size);
    }
}

int getCurrentXPosition() {
    updatePosition();
    return cur_pos.x;
}

```

One potential issue that we might run into is that the Serial buffer size is only 128 bytes. However, after examining `/usr/share/arduino/hardware/cores/arduino/HardwareSerial.cpp`, which is Arduino's implementation containing a ring buffer for the serial port, we found a

```
#define RX_BUFFER_SIZE 128
```

which indicates that this buffer size is software imposed. We can therefore modify this size to be greater than 128 if we need to buffer more bytes.

2. Image software

The webcam software is long enough that it will not be fully described in this section. For a complete description, simply go to <https://code.google.com/p/me102b-sp10/source/browse/trunk/imageunit/hsv.c>. We will however highlight the relevant portions.

First of all, here is a stripped down version of acquiring a live capture feed from the webcam, using OpenCV:

```

#include "cv.h"          /** Core OpenCV Library */
#include "highgui.h"     /** OpenCV GUI Toolkit  */

#include <math.h>        /** For abs() */

/** Used later for image processing */
#define HUE 0
#define SAT 1
#define VAL 2

#define HUE_TOL 10
#define SAT_TOL 50

typedef union {
    uchar data[3];
    struct {
        uchar hue;
        uchar sat;
        uchar val;
    } colors;
} HSV;

/** Forward declaration */

```

```

void processImage(IplImage *hsvFrame);

int main(int argc, char** argv) {
    CvCapture *capture = cvCreateCameraCapture(0); // initialize camera
    while (1) {
        IplImage *rgbFrame, *hsvFrame;
        rgbFrame = cvQueryFrame(capture); // grab frame from camera
        hsvFrame = cvCreateImage(cvGetSize(rgbFrame), IPL_DEPTH_8U, 3); // initialize HSV image buffer
        cvCvtColor(rgbFrame, hsvFrame, CV_BGR2HSV); // RGB -> HSV conversion
        processImage(hsvFrame); // image processing code goes here
        cvReleaseImage(&hsvFrame); // free HSV buffer
    }
    cvReleaseCapture(&capture); // free camera resource
}

```

Now, the code which performs the HSV algorithm is as follows:

```

int significant(HSV *a, HSV *b) {
    return abs(a->colors.hue-b->colors.hue) <= HUE_TOL &&
           abs(a->colors.sat-b->colors.sat) <= SAT_TOL;
}

void setHSV(HSV *hsv, uchar hue, uchar sat, uchar val) {
    hsv->colors.hue = hue;
    hsv->colors.sat = sat;
    hsv->colors.val = val;
}

void processImage(IplImage *hsvFrame) {
    IplImage *binFrame = cvCreateImage(cvGetSize(hsvFrame), IPL_DEPTH_8U, 1);

    uchar *data = (uchar*) hsvFrame->imageData;
    uchar *datab = (uchar*) binFrame->imageData;

    int height = hsvFrame->height;
    int width = hsvFrame->width;
    int step = hsvFrame->widthStep;
    int channels = hsvFrame->nChannels;

    HSV pixel, colorToTrack;
    setDesired(&colorToTrack); // sets the desired color to track

    for(int i=0; i<height; i++) {
        for(int j=0; j<width; j++) {
            setHSV(&pixel,
                  data[i*step+j*channels+HUE],
                  data[i*step+j*channels+SAT],
                  data[i*step+j*channels+VAL]);
            datab[i*step+j] = significant(&colorToTrack, &pixel) ? 255 : 0;
        }
    }

    CvMoments moments;
    cvMoments(binFrame, &moments, 1);
    double xmom = (moments.m10/moments.m00);
    double ymom = (moments.m01/moments.m00);

    // write (xmom, ymom) to arduino board
    cvReleaseImage(&binFrame);
}

```

3. IPC software

This last section is concerned with the software needed for inter-process communication (IPC). This includes talking from the image control loop to the Arduino, in addition to from our user facing webapp to the image control loop.

First, let's handle the issue of Arduino serial communication. Fortunately, the code is not terribly complicated because of the clean Unix `/dev` interface. Here's how we establish a connection:

```
#include <stdio.h>    /** Standard input/output definitions */
#include <unistd.h>    /** UNIX standard function definitions */
#include <fcntl.h>     /** File control definitions */
#include <errno.h>     /** Error number definitions */
#include <termios.h>   /** POSIX terminal control definitions */

int open_port(const char* serial_dev) {
    int fd;
    struct termios options;
    fd = open(serial_dev, O_RDWR | O_NOCTTY);
    if (fd == -1) {
        fprintf(stderr, "open_port: Unable to open dev: %s", serial_dev);
    } else {
        if (tcgetattr(fd, &options) < 0) {
            perror("open_port: Couldn't get term attributes");
            return -1;
        }
        cfsetispeed(&options, B9600); // set incoming baud rate of 9600
        cfsetospeed(&options, B9600); // set outgoing baud rate of 9600
    }
    return fd;
}
```

Given that, reading and writing bytes is simply a matter of calling `read` and `write`, defined in `unistd.h`, with the file descriptor returned from `open_port`.

Now, let's handle the case of a user input from our web application. The strategy here is to spawn a child thread (in the image control loop) which listens on a UNIX socket. The code to listen on a unix socket looks like such:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>

#define ADDRESS "image-unit-socket"

int main(int argc, char **argv)
{
    int fromlen = sizeof(struct sockaddr_un);
    int i, s, ns, len;
    struct sockaddr_un saun, fsaun;

    if ((s = socket(AF_UNIX, SOCK_STREAM, 0)) < 0) {
        perror("could not create unix socket");
        exit(1);
    }

    saun.sun_family = AF_UNIX;
    strcpy(saun.sun_path, ADDRESS);

    unlink(ADDRESS);
    len = sizeof(saun.sun_family) + strlen(saun.sun_path) + 1;

    if (bind(s, (const struct sockaddr *) &saun, len) < 0) {
        perror("could not bind to socket");
    }
}
```

```

        exit(1);
    }

    if (listen(s, SOMAXCONN) < 0) {
        perror("could not listen on socket");
        exit(1);
    }

    while (1) {
        if ((ns = accept(s, (struct sockaddr *) &fsock, &fromlen)) < 0) {
            perror("accept returned an error:");
            exit(1);
        }
        // process data. ns here is the file descriptor from the other end
    }
}

```

Now to talk to this server, here is a Python script (since our webapp will be written in Python) which writes a message to the image unit (namely a tuple of 3 `unsigned chars`, which is the color to be tracked).

```

#!/usr/bin/python
"""
[Usage]: ./client.py [R] [G] [B]
"""
import sys
from socket import *
from struct import pack
s = socket(AF_UNIX, SOCK_STREAM)
s.connect("image-unit-socket")
s.send(pack("BBB",int(sys.argv[1]), int(sys.argv[2]), int(sys.argv[3])))
s.close

```

4. Network Packets

We have not fully integrated the packet format described in Figure 5 into our software, but the basic skeleton for the serial wire format is:

```

#include <stdint.h>

typedef struct {
    uint16_t x;
    uint16_t y;
} Point;

typedef struct {
    Point cur;
    Point des;
    uint8_t fire;
} SerialPacket;

```

The web application can simply send a packet which looks like:

```

typedef struct {
    HSV desired;
    uint8_t fire;
} WebappPacket;

```

Where HSV is defined in the previous examples.

5. Testing/Performance of Design

We will evaluate the performance of our design based off of how accurately the turret can track a specified color. We will begin by moving the target slowly, and increasing the speed of the target until it can no longer track the target. That

will be the metric we use to measure performance.

The idea is that, if we are able to get something working before the design expo, we will spend the extra time trying to increase this metric, by tweaking system parameters and trying to optimize certain algorithms.

D. Cost Analysis

The cost analysis is characterized by the table in Table III.

Part Number	Description	Price/Product	Num. Req.	Total (USD)
-	Arduino Duemilanove	30.00	1	30.00
C250	Logitech Webcam	39.99	1	39.99
-	Wargames EG Paintball Gun	59.99	1	59.99
SD12N	Velleman HI-Q Breadboard	11.95	1	11.95
SN7417	Digital Buffer	2.00	1	2.00
LMD18200	H-Bridge	13.95	2	27.90
R40-11D2-5	Relay Switch	4.09	1	4.09

TABLE III: Cost Analysis Table

IX. CODE REPOSITORY

Throughout the semester, we will be hosting all of our project related code, and documentation, on our Google Code page found at: <http://code.google.com/p/me102b-sp10>

This will be a great resource to check up on our progress as time nears the end of the semester.

X. CONCLUSION

Since the last design review, our group has showed significant progress towards building our prototype:

1. We have a specified model of the turret and have begun to machine it.
2. We have designed the electrical circuitry to control the motors.
3. We have written code to start piecing the entire system together.

In this next month, we believe that we will be able to see these goals through to completion. A lot of the work left now is just about integrating all the separate components we have together, and making sure everything works well. Most of the technical issues at this point are specific implementation details; we understand from a high level the approach we will take to solving all the technical problems we have, and its just a matter of executing on them now.

The biggest challenge facing us right now is probably having the camera respond predictably to the lighting. Because we did not opt for a high end camera, it is more sensitive to changes in the environment with regards to lighting. It is our hope that we will be able to tune the camera options enough to mitigate these factors. In the worst case, we will simply have to upgrade our camera to a more expensive model (still should be under 100 USD though).

XI. APPENDIX

A. Concepts

Concepts are listed in chronological order of generation, with the final one being the project we chose. The rest are unrelated to the problem at hand.

1. **Projectile Shooter** - Have a laser pointer point at an object on the ground, and have an above ground device shoot at it (like a labelling command).
2. **Robot Scripting Language** - Design and implement a programming language used to script robot actions. Target audience is less experienced people who want to get hands on experience with robots, but do not want to have to sift through learning the intricacies of low level languages.

3. **Maze Solver** - Design a robot which is able to, after being placed in a arbitrary point in a maze, navigate out of the maze.
4. **Autonomous RC Car** - Design an RC car which autonomously follows a lead vehicle.
5. **Color Tracking Turret** - The project currently described in this report.

-
- [1] A. Amin, B. Chu, M. Martinez, M. Podust, and S. Tu. Color tracking turret initial project proposal. <http://me102b-sp10.googlecode.com/files/proposal.pdf>, February 2010.
 - [2] A. Amin, B. Chu, M. Martinez, M. Podust, and S. Tu. Color tracking turret initial project proposal - phase 2. <http://me102b-sp10.googlecode.com/files/phase2.pdf>, February 2010.
 - [3] Airsoft turret 10 - classic and simple. <http://hacknmod.com/hack/airsoft-turret-10-classic-and-simple/>.
 - [4] Evan Huelfer. *The "casualty issue" in American military practice*. Praeger Publishers, 2003.
 - [5] S. Zhao, B. Liu, Y. Ren, and J. Han. Color tracking vision system for the autonomous robot. In *ICEMI*, 2009.
 - [6] S. Weng, C. Kuo, and S. Tu. Video object tracking using adaptive kalman filter. In *Journal of Visual Communication and Image Representation*, 2006.
 - [7] A. Grieg. Robot2 - an arm based colour tracking robot. <http://negativeacknowledge.com/2009/05/robot2-an-arm-based-colour-tracking-robot>, 2009.
 - [8] D. Bekele, H. Liang, A. Leonard, and E. Mung. Color tracking robot. <http://www.cs.columbia.edu/~sedwards/classes/2006/4840/reports/VGLR.pdf>, May 2006.
 - [9] S. Norris. Huey - a color chasing robot. <http://www.norrislabs.com/Projects/Huey/index.html>, 2007.
 - [10] The sentry project. <http://www.paintballsentry.com/index.htm>.
 - [11] Arduino reference. <http://arduino.cc/en/Serial/Available>.
 - [12] Image moments. http://software.intel.com/sites/products/documentation/hpc/ipp/ippi/ippi_ch11/ch11_image_moments.html.
 - [13] Hsl and hsv. http://en.wikipedia.org/wiki/HSL_and_HSV.
 - [14] Red5 - open source flash. <http://osflash.org/red5>.
 - [15] Django. <http://www.djangoproject.com>.