

Color Tracking Turret Project - Phase 2

Arafat Amin,^{*} Ben Chu,[†] Mark Martinez,[‡] Michael Podust,[§] and Stephen Tu[¶]
Dept. of Mechanical Engineering - University of California, Berkeley
 (Dated: February 21, 2010)

Second phase proposal for ME102B project, where the project is discussed in more detail, along with literature review and some initial project management charts. We will go into a lot more implementation details in this report than in the initial proposal.

I. INTRODUCTION

In [1], we proposed our initial color tracking turret project. In this report, we will focus on carrying out the design in detail, first by looking at the background literature, and second by proposing our drafts for typical project management charts (QFD, Gantt, and RASCI). Then we will discuss implementation details, and outline what we believe is tractable for the remainder of this semester.

II. 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 memory and speed constraints.

In [2], 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. However, the techniques in [2] 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 [3]. 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 [4], [5], and [6]. Furthermore, the project is often done as a school project, meaning that it is a tractable semester long project.

III. QFD

The QFD is shown in Figure 1.

IV. GANTT CHART

The Gantt chart is shown in Figure 2.

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

FIG. 1: QFD

V. RASCI CHART

The RASCI chart is shown in Figure 3. 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

^{*}arafat@berkeley.edu

[†]benchu@berkeley.edu

[‡]mmartinez213@berkeley.edu

[§]mpodust@berkeley.edu

[¶]stephen_tu@berkeley.edu

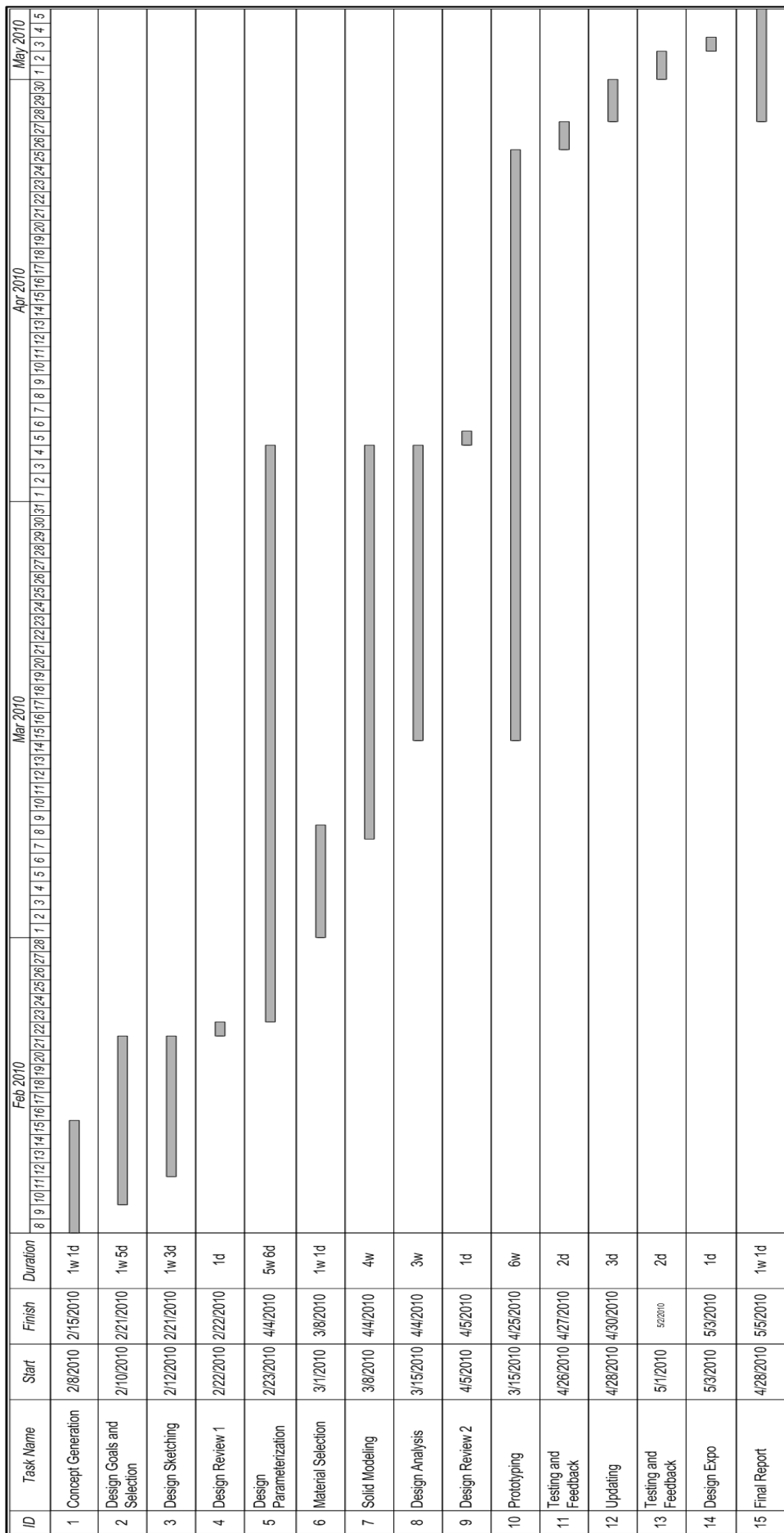


FIG. 2: Gantt Chart

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. 3: RASCI Chart

VI. FURTHER IMPLEMENTATION DETAIL

This project was designed to be quite modular; there are well defined interfaces all throughout, making it easy to develop in parallel. In this section, we will discuss the different modules of the project, and define their interfaces, starting from the lowest level.

A. 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 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 4. Note that the *fire* field is 1 byte despite being only a boolean value, since 1 byte is smallest unit of transfer.

x_t (2 bytes)	y_t (2 bytes)	x_{des} (2 bytes)	y_{des} (2 bytes)	<i>fire</i> (1 byte)
-----------------	-----------------	---------------------	---------------------	----------------------

FIG. 4: 9-byte serial packet format

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 [7], 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.

B. Image Processing Unit

The image processing unit will monitor a camera (web cam) that is attached to the turret. It's algorithm was the one described in [1]. What algorithm exactly we use for this unit is not completely decided; we will be investigating several different ones to see which gives the best desired performance. However, the behavior of this unit is described as follows:

1. Capture a frame from the camera.
2. Query for changes in desired color to track, or a *fire* command from the interactive unit (described later).
3. Run some algorithm on the captured frame to determine $x = (x_t, y_t)$ and $r = (x_{des}, y_{des})$, based on the information obtained from the previous step.
4. Serialize x and r into the wire format described in Figure 4, and send it to the controller. Set the *fire* field to be non-zero if the interactive unit indicates so.
5. Repeat.

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.

1. Image Processing Unit Socket Wire Format

The wire format for this unit is shown in Figure 5. 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).

<i>R</i> (1 byte)	<i>G</i> (1 byte)	<i>B</i> (1 byte)	<i>fire</i> (1 byte)
-------------------	-------------------	-------------------	----------------------

FIG. 5: 4-byte socket packet format

C. 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 [8]. As for the website, we will probably make use of the Django [9] 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 5.

VII. 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.

VIII. CONCLUSION

Based on the analysis we have done, we believe this project to be quite challenging, yet tractable for this semester. We have been able to play around with OpenCV already, and have gotten serial communication to work with an Arduino board. We believe the rest of the work to be done is not going to be easy, but very possible.

-
- [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] S. Zhao, B. Liu, Y. Ren, and J. Han. Color tracking vision system for the autonomous robot. In *ICEMI*, 2009.
 - [3] S. Weng, C. Kuo, and S. Tu. Video object tracking using adaptive kalman filter. In *Journal of Visual Communication and Image Representation*, 2006.
 - [4] A. Grieg. Robot2 - an arm based colour tracking robot. <http://negativeacknowledge.com/2009/05/robot2-an-arm-based-colour-tracking-robot>, 2009.
 - [5] 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.
 - [6] S. Norris. Huey - a color chasing robot. <http://www.norrislabs.com/Projects/Huey/index.html>, 2007.
 - [7] Arduino reference. <http://arduino.cc/en/Serial/Available>.
 - [8] Red5 - open source flash. <http://osflash.org/red5>.
 - [9] Django. <http://www.djangoproject.com>.