On the Design and Operation of an Omega2-based Intelligent Mood Lighting System

ECE105
Prof. Ward
Fiona Yiu
4 December 2017

Group 76: "Extremely Destructive People"
Kaisheng Huang
Edwin Hooper
Lawrence Yuan Pang

Project Overview

The purpose of our project is to provide intelligent mood lighting for a small area, based on environmental cues. The lighting color can be generated in three ways: based on averaging image color, background color behind a laser pointer, and background color behind a person's face. In addition, the project logs its results with warnings and errors, and improves its own face detection model iteratively.

This project is about several themes: intelligent image processing for recognizing certain elements, processing data using the limited capabilities of an embedded system on a chip (SoC), interfacing analog output with an embedded SoC, and interacting with compiled programs through system scripts.

Our original plan was to have it aim a laser pointer at humans and geese, but the objective could not be fulfilled as we had issues with the motors, and we lacked access to the necessary C++ libraries on the Omega2. Our current project does not have motors and does not include the machine learning aspects of the original plan; however, it retains the webcam and much of the software for image processing, object detection, and logging purposes.

Hardware Design

The hardware consists of four basic components: the Omega2 SoC, the Logitech C170 webcam, the common cathode RGB LED, and the toilet paper roll.

The C170 webcam is an inexpensive VGA-quality webcam that captures images in the YUYV colour space. This webcam was chosen due to its plug-and-play capabilities (using fswebcam, as specified above in the shell script description), the fact that high resolution pictures were not necessary, and its price. It was connected to the Omega2 simply using its inbuilt USB interface.

The common cathode RGB LED was wired to the Omega2 using a simple breadboard. It receives R, G, B values from three separate pins, read as analog inputs. The higher the voltage that was passed to the pin, the brighter the respective colour would be. The common cathode pin (as per its name) grounds the LED, and was connected to the Omega2's grounding pin.

The toilet paper roll was a fairly last-minute design decision that was deemed necessary due to the inherent "separation" of colour characteristic of RGB LED's. In order to provide a suitable colour output that was representative of the scene, the three colours had to be diffused slightly and combined into one. Furthermore, the toilet paper roll prevents any damage to the eyes from accidentally looking straight into the LED.

The Omega2 SoC was where all the commands were run from. GPIO pins 0, 1, and 2 were wired to RGB LED pins for R, G, and B, and the ground pin was connected to ground the LED. The C170 webcam was attached through USB. Control for the Omega2 was done through ssh from a laptop connected to the Omega2's local network.

Software Design

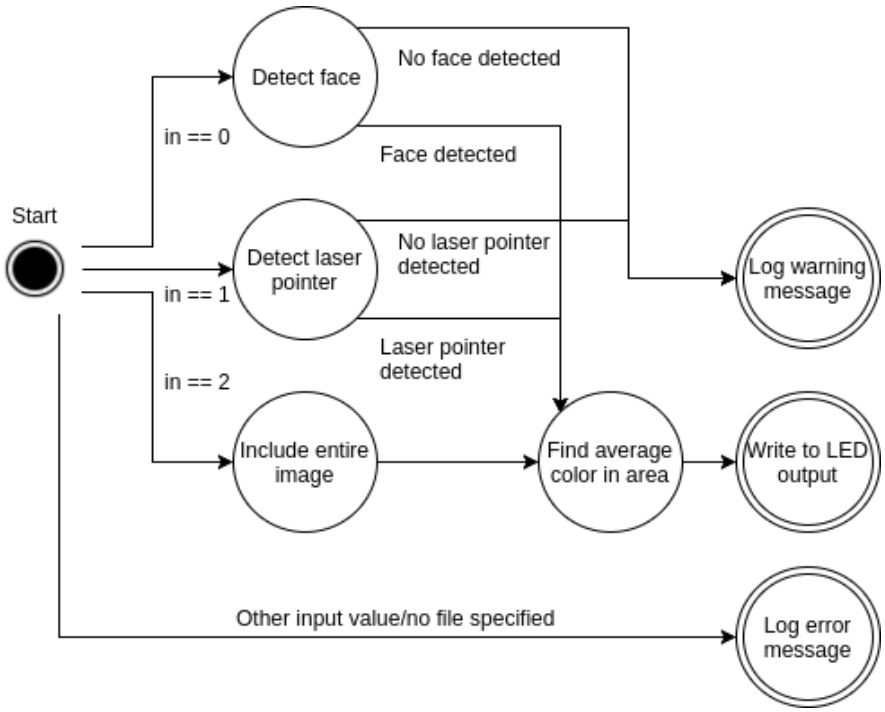
Software Overview

The entire project is run through the MountainOfFaith.sh shell script. This script reads in input from the webcam, and saves it in image.jpg. It then calls Main.cpp, which contains the main logic of the project, with two arguments: the mode of operation, and the name of the image file. As outlined above, there are three modes of operation: face detection, laser pointer detection, and averaging across the entire image. The Main.cpp file will write output to the output.txt file, which is then read by the shell script. The values contained in output.txt will be numbers from 0-100, which indicate PWM values to be written to the RGB LED. A higher PWM indicates the corresponding LED will be brighter.

The main logic for this project is contained in Main.cpp. Based on user input, it will be passed two command line arguments: the first will be an integer from 0-2, and the second will be the name of the image file, in JPEG format. The first argument indicates the mode of operation. 0 is for face detection, 1 is for laser pointer detection, and 2 is for averaging across the entire image.

The main method will first read the JPEG image with stb_image, an open source library for reading image files. It will then convert the data in the image to an object of Image class. The Image class is defined in Image.h, and its methods are implemented in Main.cpp. Depending on the mode of operation, it will then attempt to recognize a target area, which may be a face, a laser pointer, or the entire image. If it is unsuccessful in doing so, it will output a warning to the log file and default to processing the entire image. Then, it will compute the average color over and around the target area, and write this to the output.txt file, which is used to write output to the RGB LED.

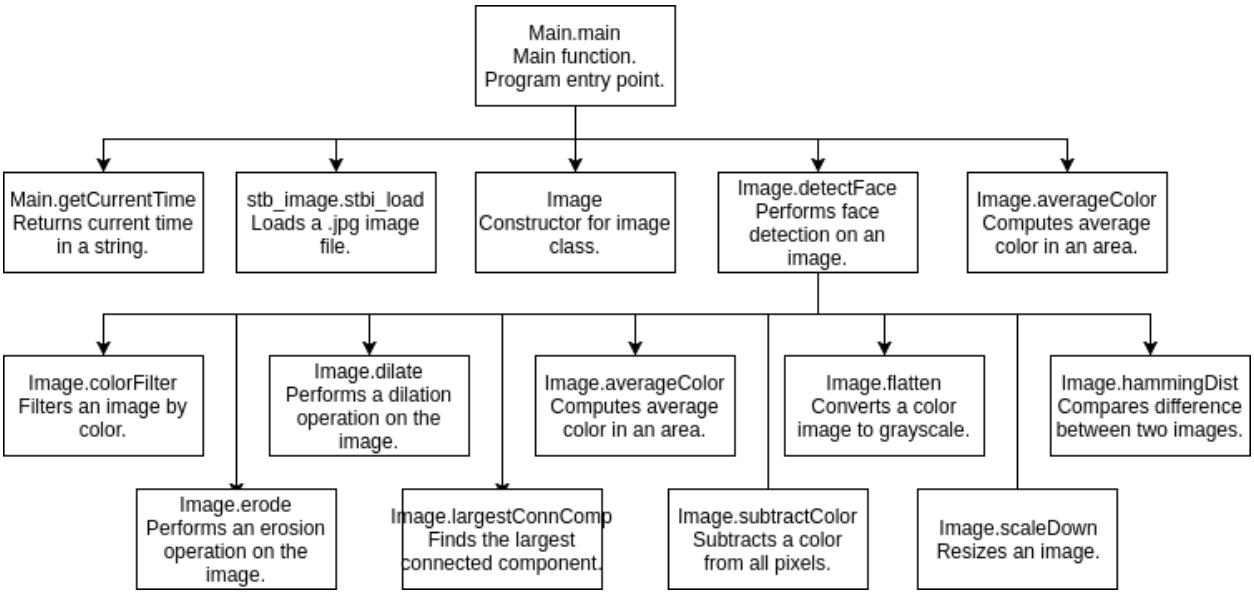
State Machine Diagram



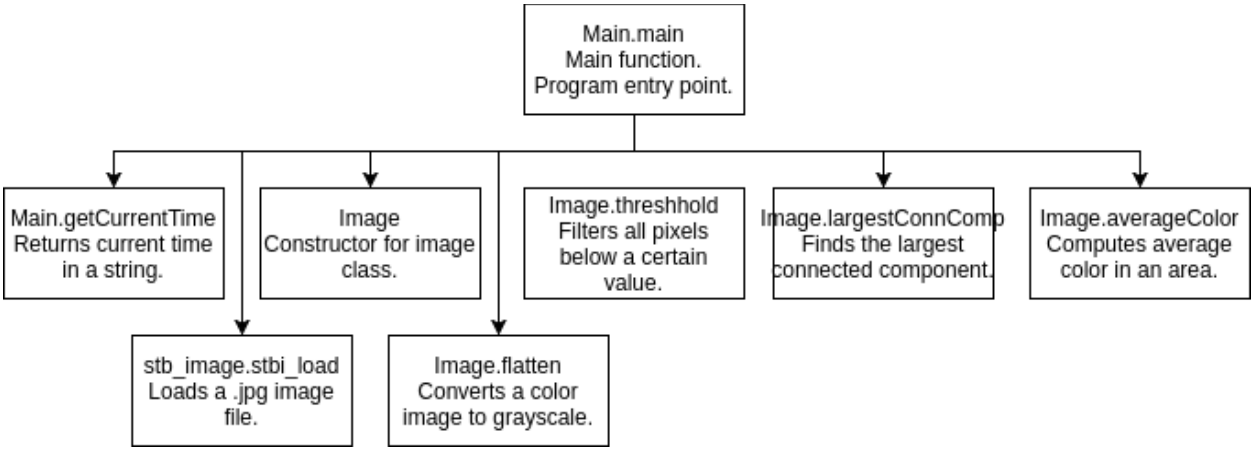
Function Call Trees

The diagrams below are the function call trees for each of the three modes of operation. System functions, such as file writing and reading, and functions used multiple times, such as getCurrentTime and constructors, are omitted or shown only once for clarity.

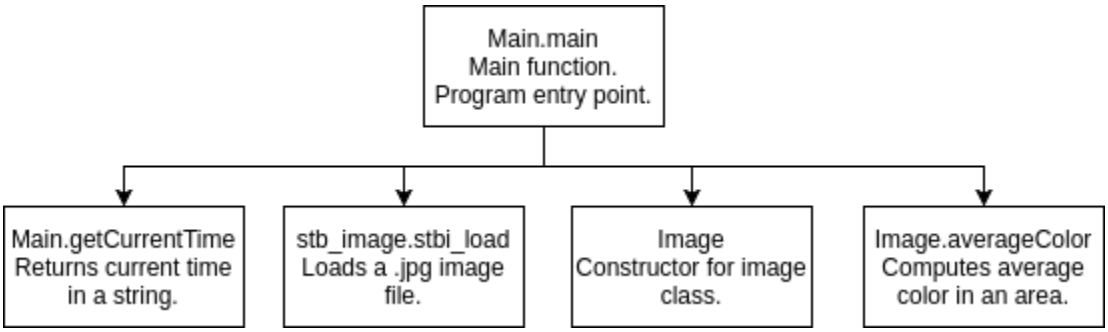
Face Detection Call Tree



Laser Pointer Detection Call Tree



Entire Image Call Tree



Classes and Methods

Image Processing

The image processing software component of this project has two components: a) face detection and b) detection of a laser pointer. The face detection component is based on the work of Darrell et al. (1998), who outlined a relatively simple face detection algorithm. The algorithm was adjusted to adapt to the constraints of this project. A rudimentary form of machine learning was also implemented so as to improve the face detection algorithm automatically.

The foundation of the image processing being performed is the Image class, defined in Image.h and implemented in Main.cpp. The members and functions of this class are described as follows:

Image Class Members

- int width, height, depth: The dimensions of the image. Depth is the number of color channels - usually 3 for red, green, and blue in that order, or 1 for grayscale images.
- std::vector< std::vector< std::vector<int> > > mat: 3-dimensional vector containing the numerical values of the image. Each element is an integer between 0 and 255 inclusive. Vectors were chosen rather than arrays due to ease of use.
- std::vector< std::vector<bool> > valid: 2-dimensional vector that acts as a filter for the image, of size width x height. Based on various operations, each pixel can be set to a boolean value based on this vector.

Image Class Methods

- Image(): Default constructor.
- Image(std::vector< std::vector< std::vector<int> > > img): Initializes mat to the parameter img. Also initializes valid to all true by default.
- Image(std::vector< std::vector< std::vector<int> > > img, std::vector< std::vector<bool> > val): Initializes mat from the parameter img, and valid from the parameter val.
- ~Image(): Default destructor.
- void colorFilter(int r, int g, int b, double tolerance): If the color of a pixel is not sufficiently close (defined by the tolerance parameter) to the value defined by (r,g,b), its filter is set to false. The metric for closeness is defined logarithmically, based on the method outlined by

Darrell et al. (1998). This is used to find colors in the image similar to skin tone, for face detection.

- void erode(int r): Erosion is an image morphological operation. For each pixel, if not all pixels within an area of r x r are true, that pixel is set to false. This is used to eliminate noise for face detection.
- void dilate(int r): Dilation is an image morphological operation. For every true pixel, set all pixels within an area of r x r to true. This is used to eliminate noise for face detection.
- std::vector<int> largestConnComp(Image& img): Finds the bounding box for the largest connected component of true pixels. Returns a vector in the form [smallest x value, smallest y value, width, height]. A connected component is one in which every true value is connected to at least one other true value. First, this function finds the size of the largest connected component and keeps track of one pixel in that component. Then, the bounds on that connected component are found. Both of these steps are accomplished by the flood-fill algorithm. The flood-fill algorithm starts at some true pixel, and adds all true neighbors of that pixel to a stack. Then it pops an element from the stack, and repeats this process until the stack is empty. This way, all true pixels connected to the starting pixel are entered in the stack at some point. Finally, all the data within the bounds of the largest connected component is written to the img parameter, passed by reference. This is used to find the areas in an image where a face or laser pointer is most likely to be.
- void subtractColor(int r, int g, int b): Subtracts an RGB value from all pixels. This is used to produce an image of the facial area.
- std::vector<int> averageColor(std::vector<int> bounds, bool val): Average color of image within some bounds. Returns vector in format [average R, average G, average B]. If parameter val is true, the average color is taken across all pixels where the filter is also true; otherwise, the average is only across pixels where the filter is false. This is useful in the case of finding the background color behind a face or laser pointer, as it is undesirable to include the actual color of the face or laser pointer. This is used to write an average color value to the RGB LED.
- void flatten(): Converts to grayscale by averaging R, G, B values. This is used to produce an image of the facial area.
- void threshold(int thresh): thresholds a grayscale image. All pixels with a value less than thresh have their filters set to false. This is used to produce an image of the facial area, and to detect bright spots (which could be a laser pointer).
- void scaleDown(int w, int h): Scales an image down to size w x h. Uses a basic averaging algorithm. This is used to produce an image of the facial area, to compare to a standard image.
- int hammingDist(Image img): Computes sum of absolute differences between each pixel in two images. This is used to determine if an area in the image contains a face.
- std::vector<int> detectFace(Image& standard, int r, int g, int b, int& loss, std::vector<int>& params, const double IMG_LEARNING_RATE, std::ofstream* logfile, std::string time): Overall function which detects faces. More explanation below.

Face Detection Algorithm

First, the image is filtered based on similarity to skin color. Then, an opening morphological operation is performed (this consists of erosion followed by dilation). The purpose

of this is to eliminate noise, such as very small connected components or irregular edges. The largest connected component is then found. A face, if one exists in the image, is likely to be located here. The image is then cropped to only where this connected component is located. The average face color is subtracted from the image, and it is thresholded and converted to grayscale. This produces an image of the face. It is then scaled down to 16x16, to be compared to a standard facial image, contained in the file std.txt. If the hamming distance between the two images is sufficiently small, the face is considered valid.

Laser Pointer Detection Algorithm

This algorithm is considerably simpler than the face detection algorithm. The image is converted to grayscale and then thresholded. Typically, as our webcam produces rather dark images, the laser will be the only object sufficiently bright. However, if other bright objects are in the image, some checks are implemented to see if the detected area is really a laser. If the area is small enough and the most common color is red (we are using a red laser), it is considered a laser. Otherwise, we repeat this process with a different area, until a laser is found or all areas are processed.

Other Methods

string getCurrentTime(): Defined in Main.cpp but not part of the Image class. This function returns the current timestamp as a string. It is used for logging purposes.

System-Dependent Components

System-dependent code is that which functions specifically on the Omega2. The code contained in Main.cpp is generally not system-dependent. The primary interaction between the Omega2 and the software is through the shell file. The majority of shell file commands can be executed on any generic UNIX-based environment (provided that the ash or bash shell is present). However, the writing of PWM values to the RGB LED is system-dependent, making use of the Omega2's GPIO pins and the fast-gpio package built in.

System-Independent Components

Most of the software is system-independent. The image processing methods and algorithms in the Image class can operate similarly on any platform. The main logic of the software, contained in Main.cpp, is also independent of the Omega2. Receiving input from the webcam and storing it to memory through the shell file is mostly system-independent, only with the note that the fswebcam package must be installed (and of course, it must be run on a UNIX system with a shell that supersedes ash). Finally, the logging infrastructure, which will be further explained below, is system-independent as well.

Logging Infrastructure

In total, the software component of this project interacts with four different files: Image.jpg, output.txt, logfile.txt, and std.txt. Image.jpg is the image read in from the webcam,

output.txt contains the RGB values written to the LED, logfile.txt contains a standard log, and std.txt contains the values of a standard facial image, which is used in the facial detection algorithm.

Logging

Log statements written to logfile.txt may take on one of three levels of significance. Normal statements concerning the day-to-day operation of the project include outputting the location and dimensions of any faces or laser pointers detected, and the RGB values written to the LED. It also logs important function calls, such as to different image processing algorithms in the Image class, for debugging purposes.

The second level includes warning statements, which are prefixed by "Warning: ". These will be logged if there is no face or laser pointer detected in the image.

Finally, there are error statements, prefixed by "Error: ". These will be logged in the case of insufficient command line arguments, or failure to open a file. In every case, the log statement is prefixed by a timestamp, returned by the getCurrentTime() function in Main.cpp.

Standard Face Image

The standard face image is contained in std.txt. While it is used in facial detection, it also serves another purpose. Although the Omega2 is not powerful enough to support true machine learning, a very rudimentary form of machine learning could still be implemented where the algorithm can continuously self-improve as it makes more face detections.

If a face is detected in the image, the average color of the face is found. A weighted average of these values and the original "standard" face color is done, which becomes the new face color to be used in filtering the next image. A similar operation is done on the image of the face. The previous image is updated with a weighted average of the new values. At the end, the updated image and color are written to the std.txt file, so these values can be reused when the program executes again. Ideally, if the user of the product scans many images of their face, the values in std.txt will resemble the user's own face more and more closely. This allows the face detection algorithm to continuously improve itself, without the need for external input – a basic form of unsupervised machine learning.

Shell Script

The interfacing between the hardware and the software on the Omega2 was done through a UNIX shell script written for the ash shell (natively run on the Omega2).

The script opens with ASCII art (inspired by the Omega2's "What will you invent" banner).

Interfacing with the webcam (a Logitech C170) was done through the package fswebcam (installed through opkg, natively built for openwrt on the MIPS-24K architecture). The fswebcam program was run with the flags *-p YUYV* which sets the colour encoding system in the file to the YUYV format, (the only one suitable for the C170), *--no-banner*, which disables the bottom banner with date/time that is written by default by fswebcam, and *-r 300x300*, which sets the resolution to fit the size of a 300x300px image (since the aspect ratio was not 1:1, the image ended up 352x288). Finally, the parameter was set to *image.jpg*, specifying the output file name.

The next steps in the script are to clear the logfile from the previous run (as logfiles build up quickly due to the amount of data stored within them) and call the Main C++ program with the parameters passed in.

The script sets up the output file to be read, then reads it line by line and writes the value to the output pins accordingly. Output pins 0-2 were set up to the LED pins for R,G,B respectively, and written using the fast-gpio package available. It sets them to output pwm (software pwm provided through digital outputs with sequences of delays in between) at 500Hz, with a on-time percentage specified by the output file. No error checking was done for the output file, as the values **must** have been set by the C++ program to fit the required format.

Finally, the logfile is opened and displayed for demo purposes.

Cross-Compilation Considerations

Cross-compilation was done utilizing the LEDE buildchain for the MIPS-24K architecture. The toolchain was set up on a local Arch Linux system referencing notes created by the TA's. The toolchain was modified to allow compilation of C++ programs and work with fast-gpio. xCompile.sh and the makefile can be found in the appendix with minor modifications made from the original example files; notable changes include changing the compiler to g++ instead of gcc and editing target names.

Additional Aspects

A summary of the source files of the project is listed below.

C/C++ Files

- Main.cpp: main logic of the program, and implements methods in the Image class.
- Image.h: defines the Image class.
- stb_image.h: open-source public-domain library used to read image files.

Shell Files:

- MountainOfFaith.sh: runs the program, interacts with hardware and calls C/C++ files.
- xCompile.sh: cross-compilation script for the program.

Other Files:

- std.txt: contains a standard face image.
- makefile: makefile for cross-compilation.

The contents of std.txt and stb_image.h are not included in the appendix. std.txt does not contain code, only integer values; stb_image.h is not original code and also over 7000 lines long.

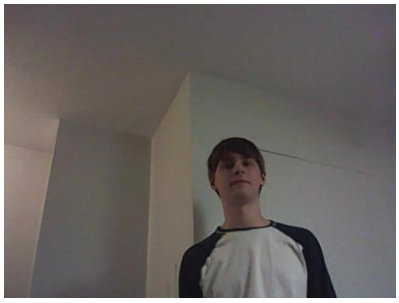
Testing

Hardware and software components were first tested separately, then together. For software, most testing focused on the facial and laser pointer detection algorithms. Images with and without faces or laser pointers were taken in different lighting conditions and with images of different sizes. Using the OpenCV library, intermediate steps in each detection algorithm were displayed, using a separate ImageTest.cpp file which is not part of the project source code. This allowed for refinement of the parameters used in the algorithm, such as skin tone and threshold. These tests were done on a laptop instead of the Omega2, so it was easy to display and compare results.

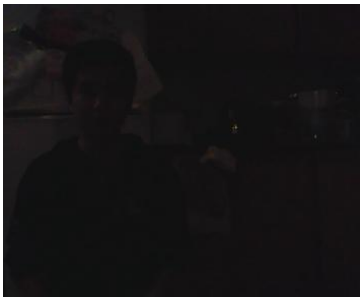
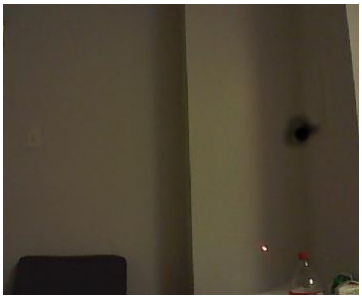
Hardware testing was mostly performed on the RGB LED. An LEDtest.sh shell script was created to write arbitrary RGB values to the LED. Using this script, the raw RGB values were calibrated to more closely resemble human perception. In particular, the red and blue signals were strengthened and the green weakened. It was also noted that the RGB LED would not display clear colours if given values that were too similar (often resulting in an odd blue-white colour), and thus the value for the strongest primary colour was doubled to compensate.

After each part was found to work separately, we migrated the source code to the Omega2 and tested the project as a whole. Again, experimentation was conducted with various lighting conditions and angles. The image size was also modified so it could be processed in a reasonable time by the Omega2 and not run out of memory (originally set to 640x480, it was resized to 352x288). We first used bright, primary colors in the background to make it obvious if the LED had changed color. Once these tests were passed, we then used subtler colors. We found that while the RGB LED was not always capable of faithfully reproducing the original color (probably due to limitations of the LED itself), it was consistently able to output the most dominant color in the background. We also tested various corner cases to assess the logging capabilities of the project.

Example testing images are given below:





Limitations

There are several tasks which this project doesn't do, which one might expect that it should. It is not capable of successful object detection under extreme lighting conditions, such as very dim or very bright environments. It can also confuse bright reflections with the laser pointer, and fails to detect faces at angles which are far from being head-on. As well, it cannot diffuse the light from the RGB LED to a consistent, uniform color, although this seems to be a fundamental limitation of the LED itself.

Our current project differs significantly from the original proposal. It no longer involves motors, due to power draw issues and faulty motor drivers. It also doesn't include continuous scanning, as the Omega2's processing ability is not sufficient. The overall purpose of the project has been changed from laser pointer aiming to mood lighting, which may appear to be completely different. However, most of the software remains the same, including object detection, image processing, and logging. In fact, we have added more modes of operation compared to the original proposal.

Lessons Learned

Fundamentally, the worst mistake made in this project was the lack of planning and adequate testing for each component that was intended to be used.

In order to have prevented difficulties with the motors, a servo expansion should have been purchased ahead of time, and datasheets for the servo and stepper were properly read to prevent a power draw mismatch. Hardware that was taken from prior projects should have been thoroughly verified for their functionality. In general, hardware components should have been acquired early and tested.

Integration of software and hardware should have been integrated at an earlier stage, so as to understand the limitations of the Omega2 and take account for it in the software design. Rather than testing all at once, each component should have been built and tested in a more modular way. In addition, more logical test cases could have been created during construction, to ensure the proper function of each component.

Overall, we have learned about the limitations of an embedded system in software design, the importance of modular testing, and the need to acquire and integrate components early.

Appendix

Peer Contribution

Kai Huang: Hardware design, assembly, testing, shell scripting, toolchain management
Ford Hooper: Hardware design, documentation, and acquisition
Lawrence Pang: image processing, main C/C++ code, software documentation

Source Code

Image.h

```
#include <math.h>
#include <assert.h>
#include <limits.h>
#include <vector>
#include <iostream>
#include <ostream>
#include <fstream>
#include <string>


class Image {
 public:
   int width, height, depth;
   std::vector< std::vector< std::vector<int> > > mat;
   std::vector< std::vector<bool> > valid;
   Image();
   Image(std::vector< std::vector< std::vector<int> > > img);
   Image(std::vector< std::vector< std::vector<int> > > img, std::vector< std::vector<bool> > val);
   ~Image();
   void colorFilter(int r, int g, int b, double tolerance);
   void erode(int r);
   void dilate(int r);
   std::vector<int> largestConnComp(Image& img, int maxSize);
   void subtractColor(int r, int g, int b);
   std::vector<int> averageColor(std::vector<int> bounds, bool val);
   void flatten();
   void threshhold(int thresh);
   void scaleDown(int w, int h);
   int hammingDist(Image img);
   std::vector<int> detectFace(Image& standard, int r, int g, int b, int& loss, std::vector<int>&
params, const double IMG_LEARNING_RATE, std::ofstream* logfile, std::string time);
};
```

Main.cpp

```
#include <stdint.h>
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
#include <vector>
```

```
#include <ostream>
#include <fstream>
#include <stdio.h>
#include <time.h>
#include <string>
#include <math.h>
#include "image.h"
using namespace std;


//define parameters
int CAMERA_WIDTH = 640;
int CAMERA_HEIGHT = 480;
int EXTEND_DIMS = 5;
const double RGB_LEARNING_RATE = 0.1;
const double IMG_LEARNING_RATE = 0.1;
const int R_SKIN_INIT = 255;
const int G_SKIN_INIT = 217;
const int B_SKIN_INIT = 139;
const int R_LASER = 255;
const int G_LASER = 255;
const int B_LASER = 255;
const char* LOG_FILE_NAME = "logfile.txt";
const char* OUTPUT_FILE_NAME = "output.txt";
const char* STD_FILE_NAME = "std.txt";


//empty default constructor
Image::Image() {
}


//constructor with image vector
Image::Image(std::vector< std::vector< std::vector<int> > > img) {
 mat = img;
 width = img.size();
 height = img[0].size();
 depth = img[0][0].size();
 valid = std::vector< std::vector<bool> >(width,std::vector<bool>(height,true));
}


//constructor with image and filter vectors
Image::Image(std::vector<    std::vector<    std::vector<int>    >    >    img,    std::vector<
std::vector<bool> > val) {
 mat = img;
```

```
 width = img.size();
 height = img[0].size();
 depth = img[0][0].size();
 valid = val;
}


//empty default destructor
Image::~Image() {
}


//filter based on color
//if color is not sufficiently close, filter for pixel is set to false
void Image::colorFilter(int r, int g, int b, double tolerance) {
 assert(depth==3);
 double tuple[3];
 tuple[0] = log(g+1);
 tuple[1] = log(r+1)-log(g+1);
 tuple[2] = log(b+1)-(log(r+1)+log(g+1))/2;
 for (int i = 0; i<width; i++) {
   for (int j = 0; j<height; j++) {
     double sample[3];
     //logarithmic method based on work of Darrell et al. (1998)
     sample[0] = log(mat[i][j][1]+1);
     sample[1] = log(mat[i][j][0]+1)-sample[0];
     sample[2] = log(mat[i][j][2]+1)-(sample[0]+sample[1])/2;
     double             dist             =             sqrt((sample[0]-tuple[0])*(sample[0]-
tuple[0])/(tuple[0]*tuple[0])+(sample[1]-tuple[1])*(sample[1]-
tuple[1])/(tuple[1]*tuple[1])+(sample[2]-tuple[2])*(sample[2]-tuple[2])/(tuple[2]*tuple[2]));
     if (dist>tolerance) {
       valid[i][j] = false;
     }
   }
 }
}


//erosion operation
//if not all pixels within an area of rxr are true, that pixel is set to false
void Image::erode(int r) {
 assert(r%2==1);
 int dim = r/2;
 std::vector< std::vector<bool> > temp = valid;
 for (int i = 0; i<width; i++) {
   for (int j = 0; j<height; j++) {
```

```
      if (valid[i][j]) {
        for (int k = -dim; k<=dim; k++) {
          for (int l = -dim; l<=dim; l++) {
            if (!(i+k>=0&&i+k<width&&j+l>=0&&j+l<height&&valid[i+k][j+l]))
              temp[i][j] = false;
          }
        }
      }
    }
  }
  valid = temp;
}

//dilation operation
//for every true pixel, set all pixels within an area of rxr to true
void Image::dilate(int r) {
  assert(r%2==1);
  int dim = r/2;
  std::vector< std::vector<bool> > temp = valid;
  for (int i = 0; i<width; i++) {
    for (int j = 0; j<height; j++) {
      if (valid[i][j]) {
        for (int k = -dim; k<=dim; k++) {
          for (int l = -dim; l<=dim; l++) {
            if (i+k>=0&&i+k<width&&j+l>=0&&j+l<height)
              temp[i+k][j+l] = true;
          }
        }
      }
    }
  }
  valid = temp;
}

//finds the bounding box for the largest connected component of true pixels
//returns a vector in the form [smallest x value, smallest y value, width, height]
std::vector<int> Image::largestConnComp(Image& img, int maxSize = INT_MAX) {
  std::vector< std::vector<bool> > checked(width,std::vector<bool>(height,false));
  std::vector<int> wvals(0);
  std::vector<int> hvals(0);
  int wpoint = -1;
  int hpoint = -1;
```

```
int max = 0;
//finds size of largest connected component and keeps track of one pixel in that component
//uses flood fill algorithm
for (int i = 0; i<width; i++) {
  for (int j = 0; j<height; j++) {
    if (!checked[i][j]&&valid[i][j]) {
      wvals.clear();
      hvals.clear();
      int count = 1;
      wvals.push_back(i);
      hvals.push_back(j);
      while (wvals.size()!=0&&hvals.size()!=0) {
        int k = wvals.back();
        wvals.pop_back();
        int l = hvals.back();
        hvals.pop_back();
        assert(wvals.size()==hvals.size());
        if (k+1>=0&&k+1<width&&l+1>=0&&l+1<height&&!checked[k+1][l+1]&&valid[k+1][l+1]) {
          checked[k+1][l+1] = true;
          wvals.push_back(k+1);
          hvals.push_back(l+1);
          count++;
        }
        if (k+1>=0&&k+1<width&&l-1>=0&&l-1<height&&!checked[k+1][l-1]&&valid[k+1][l-1]) {
          checked[k+1][l-1] = true;
          wvals.push_back(k+1);
          hvals.push_back(l-1);
          count++;
        }
        if (k-1>=0&&k-1<width&&l+1>=0&&l+1<height&&!checked[k-1][l+1]&&valid[k-1][l+1]) {
          checked[k-1][l+1] = true;
          wvals.push_back(k-1);
          hvals.push_back(l+1);
          count++;
        }
        if (k-1>=0&&k-1<width&&l-1>=0&&l-1<height&&!checked[k-1][l-1]&&valid[k-1][l-1]) {
          checked[k-1][l-1] = true;
          wvals.push_back(k-1);
          hvals.push_back(l-1);
          count++;
        }
      }
      if (count>max&&count<=maxSize) {
```

```
          max = count;
          wpoint = i;
          hpoint = j;
        }
      }
    }
  }
  if (wpoint==-1&&hpoint==-1)
    return std::vector<int>(0);
  checked = std::vector <std::vector<bool> >(width,std::vector<bool>(height,false));
  wvals.clear();
  hvals.clear();
  int wmin = width;
  int wmax = 0;
  int hmin = height;
  int hmax = 0;
  wvals.push_back(wpoint);
  hvals.push_back(hpoint);
  //finds bounds on largest connected component starting with tracked pixel from above
  //again uses flood fill algorithm
  while (wvals.size()!=0&&hvals.size()!=0) {
    int k = wvals.back();
    wvals.pop_back();
    int l = hvals.back();
    hvals.pop_back();
    assert(wvals.size()==hvals.size());
    if (k<wmin)
      wmin = k;
    if (k>wmax)
      wmax = k;
    if (l<hmin)
      hmin = l;
    if (l>hmax)
      hmax = l;
    if (k+1>=0&&k+1<width&&l+1>=0&&l+1<height&&!checked[k+1][l+1]&&valid[k+1][l+1]) {
      checked[k+1][l+1] = true;
      wvals.push_back(k+1);
      hvals.push_back(l+1);
    }
    if (k+1>=0&&k+1<width&&l-1>=0&&l-1<height&&!checked[k+1][l-1]&&valid[k+1][l-1]) {
      checked[k+1][l-1] = true;
      wvals.push_back(k+1);
      hvals.push_back(l-1);
```

```
    }
    if (k-1>=0&&k-1<width&&l+1>=0&&l+1<height&&!checked[k-1][l+1]&&valid[k-1][l+1]) {
      checked[k-1][l+1] = true;
      wvals.push_back(k-1);
      hvals.push_back(l+1);
    }
    if (k-1>=0&&k-1<width&&l-1>=0&&l-1<height&&!checked[k-1][l-1]&&valid[k-1][l-1]) {
      checked[k-1][l-1] = true;
      wvals.push_back(k-1);
      hvals.push_back(l-1);
    }
  }
  //writes all pixels and filter values within bounds to an image, passed by reference
  img.width = wmax-wmin+1;
  img.height = hmax-hmin+1;
  img.depth = depth;
  img.mat = std::vector< std::vector< std::vector<int> > >(img.width, std::vector<
std::vector<int> >(img.height,std::vector<int>(depth)));
  img.valid = std::vector< std::vector<bool> >(img.width,std::vector<bool>(img.height,false));
  for (int i = 0; i<img.width; i++) {
    for (int j = 0; j<img.height; j++) {
      for (int k = 0; k<depth; k++) {
        img.mat[i][j][k] = mat[wmin+i][hmin+j][k];
      }
      img.valid[i][j] = valid[wmin+i][hmin+j];
    }
  }
  std::vector<int> out(0);
  out.push_back(wmin);
  out.push_back(hmin);
  out.push_back(img.width);
  out.push_back(img.height);
  return out;
}


//subtracts an rgb value from all pixels
void Image::subtractColor(int r, int g, int b) {
  for (int i = 0; i<width; i++) {
    for (int j = 0; j<height; j++) {
      mat[i][j][0] = (mat[i][j][0]-r)*((mat[i][j][0]-r>0)*2-1);
      mat[i][j][1] = (mat[i][j][1]-g)*((mat[i][j][1]-g>0)*2-1);
      mat[i][j][2] = (mat[i][j][2]-b)*((mat[i][j][2]-b>0)*2-1);
    }
```

```
 }
}


//average color of image within some bounds
//returns vector in format [average r, average g, average b]
std::vector<int> Image::averageColor(std::vector<int> bounds, bool val = true) {
 std::vector<int> avg(depth,0);
 int count = 0;
 for (int i = bounds[0]; i<bounds[0]+bounds[2]; i++) {
   for (int j = bounds[1]; j<bounds[1]+bounds[3]; j++) {
     if (val) {
       if (valid[i][j]) {
         count++;
         for (int k = 0; k<depth; k++) {
           avg[k]+=mat[i][j][k];
         }
       }
     }
     else {
       if (!valid[i][j]) {
         count++;
         for (int k = 0; k<depth; k++) {
           avg[k]+=mat[i][j][k];
         }
       }
     }
   }
 }
 if (count>0) {
   for (int i = 0; i<depth; i++) {
     avg[i] = avg[i]/count;
   }
 }
 return avg;
}


//converts to grayscale by averaging r, g, b
void Image::flatten() {
 std::vector<     std::vector<     std::vector<int>     >     >     temp(width,std::vector<
std::vector<int> >(height,std::vector<int>(1,0)));
 for (int i = 0; i<width; i++) {
   for (int j = 0; j<height; j++) {
     for (int k = 0; k<depth; k++) {
```

```
        temp[i][j][0]+=mat[i][j][k];
      }
      temp[i][j][0] = temp[i][j][0]/depth;
    }
  }
  mat = temp;
  depth = 1;
}


//threshholds grayscale image
void Image::threshhold(int thresh) {
  assert(depth==1);
  for (int i = 0; i<width; i++) {
    for (int j = 0; j<height; j++) {
      if (mat[i][j][0]<thresh)
        valid[i][j] = false;
    }
  }
}


//scales image down to given size
//basic averaging algorithm
void Image::scaleDown(int w, int h) {
  assert(depth==1);
  std::vector<      std::vector<      std::vector<int>      >      >      temp(w,std::vector<
std::vector<int> >(h,std::vector<int>(depth,0)));
  std::vector< std::vector<bool> > tempValid(w,std::vector<bool>(h,true));
  int wratio = width/w;
  int hratio = height/h;
  for (int i = 0; i<w; i++) {
    for (int j = 0; j<h; j++) {
      int count = 0;
      for (int k = 0; k<wratio; k++) {
        for (int l = 0; l<hratio; l++) {
          if
(i*wratio+k>=0&&i*wratio+k<width&&j*hratio+l>=0&&j*hratio+l<height&&valid[i*wratio+k][j*hrati
o+l]) {
            temp[i][j][0]+=mat[i*wratio+k][j*hratio+l][0];
            count++;
          }
        }
      }
      if (count!=0)
        temp[i][j][0] = temp[i][j][0]/count;
```

```
      if (count<wratio*hratio/2)
        tempValid[i][j] = false;
    }
  }
  mat = temp;
  valid = tempValid;
  width = w;
  height = h;
}


//computes sum of absolute differences between each pixel in two images
int Image::hammingDist(Image img) {
  assert(img.width==width);
  assert(img.height==height);
  assert(depth==1);
  int count = 0;
  for (int i = 0; i<width; i++) {
    for (int j = 0; j<height; j++) {
      if ((img.mat[i][j][0]>0)^(valid[i][j]))
        count++;
    }
  }
  return count;
}


//detects face in image and returns its bounds
//returns a vector in the form [smallest x value, smallest y value, width, height]
//based on work of Darrell et al. (1998)
std::vector<int> Image::detectFace(Image& standard, int rskin, int gskin, int bskin, int& loss,
std::vector<int>& params, const double IMG_LEARNING_RATE, ofstream* logfile, string time) {
  //various parameters
  double tolerance = 10;
  int rerode = 3;
  int rdilate = 5;
  int thresh = 50;
  int w = 16;
  int h = 16;
  int lim = 200;
  //filters image based on similarity to skin color
  colorFilter(rskin,bskin,gskin,tolerance);
  *logfile << time << "Applied color filter to image." << endl;
  //image opening (erosion followed by dilation)
  //generally eliminates noise
```

```
 erode(rerode);
 dilate(rdilate);
 Image newimg = Image();
 //finds the largest connected component
 //hopefully, this is where the face is
 std::vector<int> dims = largestConnComp(newimg);
 if (dims.size()>0) {
   *logfile << time << "Found possible face candidate." << endl;
   //creates a mask of the face by subtracting average face color
   params = averageColor(dims);
   newimg.subtractColor(rskin,gskin,bskin);
   //convert to grayscale
   newimg.flatten();
   //threshhold to eliminate noise
   newimg.threshhold(thresh);
   //scale down to compare to standard face mask
   newimg.scaleDown(w,h);
   *logfile << time << "Created face mask from image." << endl;
   //compare to standard face mask
   loss = newimg.hammingDist(standard);
   *logfile << time << "Compared image mask to standard face mask." << endl;
   //if face is close enough to standard mask, return bounds
   if (loss<=lim) {
     //update standard mask
     //weighted average between standard mask and mask in current image
     double alpha = loss*IMG_LEARNING_RATE;
     for (int i = 0; i<w; i++) {
       for (int j = 0; j<h; j++) {
         standard.mat[i][j][0]             =             (int)(alpha*newimg.mat[i][j][0]+(1-
alpha)*standard.mat[i][j][0]);
       }
     }
     return dims;
   }
   else
     *logfile << time << "Warning: Face candidate not sufficiently similar to standard face
mask." << endl;
 }
 //return empty vector
 *logfile << time << "Warning: No face candidate found." << endl;
 return std::vector<int>(0);
}


//gets current local time as string
```

```
string getCurrentTime() {
 time_t rawtime;
 char buffer[25];
 time(&rawtime);
 struct tm* timeinfo = localtime(&rawtime);
 strftime(buffer,25,"%c",timeinfo);
 return string(buffer)+": ";
}


int main(const int argc, const char* const argv[]) {
 //various parameters
 int r = R_SKIN_INIT;
 int g = G_SKIN_INIT;
 int b = B_SKIN_INIT;
 ifstream file;
 ofstream logfile;
 ofstream output;
 file.open(STD_FILE_NAME);
 logfile.open(LOG_FILE_NAME,std::ios::app);
 if (argc!=3) {
   logfile << getCurrentTime() << "Error: Insufficient command line arguments." << endl;
   return -1;
 }
 int choice = atoi(argv[1]);
 const char* IMAGE_FILE_NAME = argv[2];
 //read standard face mask from file
 int w, h;
 file >> w;
 file >> h;
 std::vector<      std::vector<      std::vector<int>      >      >standardMat(w,std::vector<
std::vector<int> >(h,std::vector<int>(1,0)));
 for (int i = 0; i<w; i++) {
   for (int j = 0; j<h; j++) {
     file >> standardMat[i][j][0];
   }
 }
 file.close();
 logfile << getCurrentTime() << "Read image file." << endl;
 Image standard(standardMat);
 int count = 0;
 //read jpg file using the stb_image library
 int bpp;
 uint8_t* rgb_image = stbi_load(IMAGE_FILE_NAME, &CAMERA_WIDTH, &CAMERA_HEIGHT, &bpp, 3);
```

```
 std::vector<     std::vector<     std::vector<int>     >     >mat(CAMERA_WIDTH,std::vector<
std::vector<int> >(CAMERA_HEIGHT,std::vector<int>(3,0)));
 for (int i = 0; i<CAMERA_HEIGHT; i++) {
   for (int j = 0; j<CAMERA_WIDTH; j++) {
     for (int k = 0; k<3; k++) {
       mat[j][i][k] = rgb_image[(i*CAMERA_WIDTH+j)*3+k];
     }
   }
 }
 stbi_image_free(rgb_image);
 logfile << getCurrentTime() << "Created image object." << endl;
 Image i(mat);
 std::vector<int> dims(4,0);
 //face detection if command line argument is 0
 if (choice==0) {
   logfile << getCurrentTime() << "Starting face detection." << endl;
   EXTEND_DIMS = 5;
   int loss = 0;
   std::vector<int> params(3,0);
   //detect face
   dims                                                                                    =
i.detectFace(standard,r,g,b,loss,params,IMG_LEARNING_RATE,&logfile,getCurrentTime());
   //if face detection is successful
   if (dims.size()>0&&dims[2]>=15&&dims[3]>=15) {
     EXTEND_DIMS = 20;
     //update average face colour
     //weighted average between standard color and average color in image
     double alpha = RGB_LEARNING_RATE*log(255-loss)/log(255);
     r = params[0]*alpha+r*(1-alpha);
     g = params[1]*alpha+g*(1-alpha);
     b = params[2]*alpha+b*(1-alpha);
     logfile << getCurrentTime() << "Adjusted skin color filter to (" << r << ", " << g << ",
" << b << ")." << endl;
     logfile << getCurrentTime() << "Face detected in frame " << count << " with upper-left
corner (" << dims[0] << ", " << dims[1] << ") and dimensions " << dims[2] << "x" << dims[3] <<
"." << endl;
   }
   else {
     EXTEND_DIMS = 0;
     dims[0] = 0;
     dims[1] = 0;
     dims[2] = CAMERA_WIDTH;
     dims[3] = CAMERA_HEIGHT;
     logfile << getCurrentTime() << "Warning: No face detected in frame " << count << ".
Averaging over entire image." << endl;
   }
 }
```

```
//laser pointer detection if command line argument is 1
else if (choice==1) {
  logfile << getCurrentTime() << "Starting laser pointer detection." << endl;
  EXTEND_DIMS = 5;
  Image bw = i;
  //convert to grayscale
  bw.flatten();
  //threshhold to find bright point (i.e. the laser pointer, hopefully)
  bw.threshhold(235);
  while (dims.size()>0) {
    //find the largest (probably only) connected component
    Image newimg = Image();
    dims = bw.largestConnComp(newimg,50);
    logfile << getCurrentTime() << "Found possible laser pointer candidate." << endl;
    //detect dominant red color
    int redCount = 0;
    for (int j = dims[0]; j<dims[0]+dims[2]; j++) {
      for (int k = dims[1]; k<dims[1]+dims[3]; k++) {
        if (i.mat[j][k][0]>=i.mat[j][k][1]&&i.mat[j][k][0]>=i.mat[j][k][2])
          redCount++;
      }
    }
    //if red is dominant and area is small, pointer detected
    if ((redCount+0.0)/(dims[2]*dims[3])>0.8&&dims[2]<10&&dims[3]<10)
      break;
    //otherwise, set current connected component to false
    else {
      logfile << getCurrentTime() << "Candidate was not a laser pointer. Retrying." << endl;
      for (int j = dims[0]; j<dims[0]+dims[2]; j++) {
        for (int k = dims[1]; k<dims[1]+dims[3]; k++) {
          bw.valid[j][k] = false;
        }
      }
    }
  }
  //if laser pointer detection successful
  if (dims.size()>0)
    logfile << getCurrentTime() << "Laser pointer detected in frame " << count << " at (" <<
(dims[0]+dims[2]/2) << ", " << (dims[1]+dims[3]/2) << ")." << endl;
  else {
    EXTEND_DIMS = 0;
    dims[0] = 0;
    dims[1] = 0;
```

```
      dims[2] = CAMERA_WIDTH;
      dims[3] = CAMERA_HEIGHT;
      logfile << getCurrentTime() << "Warning: No laser pointer detected in frame " << count <<
". Averaging over entire image." << endl;
    }
  }
  //averaging over entire image
  else {
    logfile << getCurrentTime() << "Averaging over entire image." << endl;
    EXTEND_DIMS = 0;
    dims[0] = 0;
    dims[1] = 0;
    dims[2] = CAMERA_WIDTH;
    dims[3] = CAMERA_HEIGHT;
  }
  //dimensions of an area surrounding target detection
  //whether face or laser pointer
  logfile << getCurrentTime() << "Computing average color around target." << endl;
  std::vector<int> extended(4,0);
  extended[0] = max(dims[0]-EXTEND_DIMS,0);
  extended[1] = max(dims[1]-EXTEND_DIMS,0);
  extended[2] = min(dims[2]+2*EXTEND_DIMS,CAMERA_WIDTH-extended[0]);
  extended[3] = min(dims[3]+2*EXTEND_DIMS,CAMERA_HEIGHT-extended[1]);
  //compute average color in area
  std::vector<int> avg = i.averageColor(extended,choice!=0);
  //scale down from 0-255 to 0-100
  for (int i = 0; i<avg.size(); i++)
    avg[i] = avg[i]*100/255;
  //calibrations
  avg[1] = (int)(0.75*avg[1]);
  if (choice==1)
    avg[0] = (int)(0.5*avg[0]);
  //color adjustments for emphasis
  if (avg[0]>=avg[1]&&avg[0]>=avg[2])
    avg[0] = min(2*avg[0],100);
  if (avg[1]>=avg[0]&&avg[1]>=avg[2])
    avg[1] = min(2*avg[1],100);
  if (avg[2]>=avg[0]&&avg[2]>=avg[1])
    avg[2] = min(2*avg[2],100);
  logfile << getCurrentTime() << "Wrote RGB color (" << avg[0] << ", " << avg[1] << ", " <<
avg[2] << ") to LED." << endl;
  //write to output file for led control
  output.open(OUTPUT_FILE_NAME);
  output << avg[0] << endl << avg[1] << endl << avg[2] << endl;
```

```
  output.close();
  //write updated standard face mask to file
  ofstream stdwrite;
  stdwrite.open(STD_FILE_NAME);
  stdwrite << standard.width << " " << standard.height << endl;
  for (int i = 0; i<standard.width; i++) {
    for (int j = 0; j<standard.height; j++) {
      stdwrite << standard.mat[i][j][0] << " ";
    }
    stdwrite << endl;
  }
  stdwrite.close();
  logfile << getCurrentTime() << "Updated standard image template." << endl;
  logfile.close();
 }
```

MountainOfFaith.sh

```
#!usr/bin/ash

#Yay fun ASCII art here
echo "################################################################################"
echo "#                      WELCOME TO THE MORIYA SHRINE                            #"
echo "#     *****\_____________________________________________________/*****     #"
echo "#      /#############################################################\      #"
echo "#    /#################################################################\    #"
echo "#           ###################################################           #"
echo "#                     ######       ######       ######                    #"
echo "#                     ######        ####        ######                    #"
echo "#           ***\      <<|##|>>        ##        <<|##|>>     /***          #"
echo "#             ################################################             #"
echo "#             ################################################             #"
echo "#                       |##|        .,            |##|                      #"
echo "#                       |##|  ,'',.'  '.,         |##|                      #"
echo "#                  _.-_  |##|,'_.-_        ',  _.-_  |##|  _.-_             #"
echo "#                 /###\'|##| /###\         '/###\ |##| /###\                #"
echo "#       ,'''',.   ,'|#|  =##=  |#|           |#|',=##=  |#|   _           ,#"
echo "#-, ..''      '.' ################          ################,' '',.'',. .,' #"
echo "#__'_____________################__________################___________'____#"
echo "################################################################################"

#Take picture using fswebcam
```

```
#Sets colourspace to YUYV (otherwise it crashes with the Logitech webcam)
#Disables the default banner (which just overlays date/time/etc onto image)
#Stores the image as "image.jpg"
fswebcam -p YUYV --no-banner -r 300x300 image.jpg

#Clear the logfile from the previous run
rm logfile.txt

#Call the main program (Image processor) with parameters 1 and 2
./Main $1 $2

#Set up script to read from output.txt
filename="output.txt"

#Create a variable to use as a counter
COUNTERVAR=0

#Iterates through the lines of the output file
while read -r line
do
        name="$line"
        #Writes the value to the corresponding PWM pin
        #PWM set to be at 500Hz, countervar sets which pin, name sets value
        fast-gpio pwm $COUNTERVAR 500  $name
        echo "Writing $name to $COUNTERVAR"
        COUNTERVAR=$(expr $COUNTERVAR + 1)
done < "output.txt"

#Open logfile afterwards for demo
cat logfile.txt
```

xCompile.sh

```
#!/bin/bash


            # define the usage
            usage () {
                    echo "Arguments:"
                    echo "  -buildroot <path to buildroot>"
                    echo "  -lib \"<list of additional libraries to link in compile>\""
                    echo ""
```

```
}


# define the path to the buildroot
BUILDROOT_PATH=""
USER_LIBS=""
bDebug=0




####################
# parse arguments #
while [ "$1" != "" ]
do
        case "$1" in
                # options
                buildroot|-buildroot|--buildroot)
                        shift
                        BUILDROOT_PATH="$1"
                        shift
                ;;
                lib|-lib|--lib)
                        shift
                        USER_LIBS="$1"
                        shift
                ;;
                -d|--d|debug|-debug|--debug)
                        bDebug=1
                        shift
                ;;
                -h|--h|help|-help|--help)
                        usage
                        exit
                ;;
            *)
                        echo "ERROR: Invalid Argument: $1"
                        usage
                        exit
                ;;
        esac
done
```

```
# check to ensure correct arguments
if [ "$BUILDROOT_PATH" = "" ]
then
        echo "ERROR: missing path to buildroot"
        echo ""
        usage
        exit
fi



# define the toolchain and target names
TOOLCHAIN_NAME="toolchain-mipsel_24kc_gcc-5.5.0_musl"
TARGET_NAME="target-mipsel_24kc_musl"


# define the relative paths
STAGING_DIR_RELATIVE="staging_dir"
TOOLCHAIN_RELATIVE="$STAGING_DIR_RELATIVE/$TOOLCHAIN_NAME"
TARGET_RELATIVE="$STAGING_DIR_RELATIVE/$TARGET_NAME"


# define the toolchain paths
TOOLCHAIN="$BUILDROOT_PATH/$TOOLCHAIN_RELATIVE"
TOOLCHAIN_BIN="$BUILDROOT_PATH/$TOOLCHAIN_RELATIVE/bin"


TOOLCHAIN_INCLUDE="$BUILDROOT_PATH/$TOOLCHAIN_RELATIVE/include"
TOOLCHAIN_LIB="$BUILDROOT_PATH/$TOOLCHAIN_RELATIVE/lib"
TOOLCHAIN_USR_INCLUDE="$BUILDROOT_PATH/$TOOLCHAIN_RELATIVE/usr/include"
TOOLCHAIN_USR_LIB="$BUILDROOT_PATH/$TOOLCHAIN_RELATIVE/usr/lib"


# define the target paths
TARGET="$BUILDROOT_PATH/$TARGET_RELATIVE"


TARGET_INCLUDE="$BUILDROOT_PATH/$TARGET_RELATIVE/include"
TARGET_LIB="$BUILDROOT_PATH/$TARGET_RELATIVE/lib"
TARGET_USR_INCLUDE="$BUILDROOT_PATH/$TARGET_RELATIVE/usr/include"
TARGET_USR_LIB="$BUILDROOT_PATH/$TARGET_RELATIVE/usr/lib"
```

```
export STAGING_DIR="BUILDROOT_PATH/$STAGING_DIR_RELATIVE"


# define the compilers and such
TOOLCHAIN_CC="$TOOLCHAIN_BIN/mipsel-openwrt-linux-gcc"
TOOLCHAIN_CXX="$TOOLCHAIN_BIN/mipsel-openwrt-linux-g++"
TOOLCHAIN_LD="$TOOLCHAIN_BIN/mipsel-openwrt-linux-ld"


TOOLCHAIN_AR="$TOOLCHAIN_BIN/mipsel-openwrt-linux-ar"
TOOLCHAIN_RANLIB="$TOOLCHAIN_BIN/mipsel-openwrt-linux-ranlib"


# define the FLAGS
INCLUDE_LINES="-I $TOOLCHAIN_USR_INCLUDE -I $TOOLCHAIN_INCLUDE -I $TARGET_USR_INCLUDE -I
$TARGET_INCLUDE"
TOOLCHAIN_CFLAGS="-Os -pipe -mno-branch-likely -mips32r2 -mtune=24kc -fno-caller-saves -
fno-plt -fhonour-copts -Wno-error=unused-but-set-variable -Wno-error=unused-result -
msoft-float -mips16 -minterlink-mips16 -Wformat -Werror=format-security -fstack-
protector -D_FORTIFY_SOURCE=1 -Wl,-z,now -Wl,-z,relro"
#TOOLCHAIN_CFLAGS="-Os -pipe -mno-branch-likely -mips32r2 -mtune=34kc -fno-caller-saves
-fhonour-copts -Wno-error=unused-but-set-variable -Wno-error=unused-result -msoft-float
-mips16 -minterlink-mips16 -fpic"
TOOLCHAIN_CFLAGS="$TOOLCHAIN_CFLAGS $INCLUDE_LINES"


TOOLCHAIN_CXXFLAGS="$TOOLCHAIN_CFLAGS"
#TOOLCHAIN_CXXFLAGS="-Os -pipe -mno-branch-likely -mips32r2 -mtune=34kc -fno-caller-
saves -fhonour-copts -Wno-error=unused-but-set-variable -Wno-error=unused-result -msoft-
float -mips16 -minterlink-mips16 -fpic"
TOOLCHAIN_CXXFLAGS="$TOOLCHAIN_CXXFLAGS $INCLUDE_LINES"


TOOLCHAIN_LDFLAGS="-L$TOOLCHAIN_USR_LIB -L$TOOLCHAIN_LIB -L$TARGET_USR_LIB -
L$TARGET_LIB"


# debug
if [ $bDebug -eq 1 ]; then
        echo "CC=$TOOLCHAIN_CC"
        echo "CXX=$TOOLCHAIN_CXX"
```

```
        echo "LD=$TOOLCHAIN_LD"
        echo "CFLAGS=$TOOLCHAIN_CFLAGS"
        echo "LDFLAGS=$TOOLCHAIN_LDFLAGS"
        echo "USER_LIBS=$USER_LIBS"
        echo ""
fi




# first run make clean
make clean


# run the make command
make \
        CC="$TOOLCHAIN_CC" \
        CXX="$TOOLCHAIN_CXX" \
        LD="$TOOLCHAIN_LD" \
        CFLAGS="$TOOLCHAIN_CFLAGS" \
        LDFLAGS="$TOOLCHAIN_LDFLAGS" \
        LIB="$USER_LIBS"
```

makefile

```
# main
compiler
CC := g++


TARGET1 := Main


all: $(TARGET1)


$(TARGET1):
        @echo "Compiling C program"
        $(CXX) $(CFLAGS) $(TARGET1).cpp -o $(TARGET1) $(LDFLAGS) -l$(LIB)


clean:
```

```
	@rm -rf $(TARGET1) $(TARGET2)
```