**Appendix A: Requirements and Verifications Table**

Table 4: R/V Table

|  |  |  |
| --- | --- | --- |
| **Requirements** | **Verification** | **Points** |
| **Passive Infrared (PIR) Sensor**   1. Must output high of 3.3 V (+/- .3V) when motion is detected. Must output low of 0.0 V (+/-.3 V). 2. Must be able to detect an object approaching the drone within 20 ft(+/- 5 ft) | 1. **Verification process for item 1**     1. Connect PIR sensor power to 5V from arduino board. Note that you must allow the necessary 1 minute setup time before trying to detect motion    2. Connect ground and output signal of PIR sensor to digital multimeter    3. Ensure that the output voltage when no motion is detected is between -.3V and .3V    4. Wave hand over top of lens on PIR sensor and ensure that the output voltage is between 3V and 3.6V 2. **Verification process for item 2** 3. Connect PIR sensor power and GND to 5V and GND on arduino board 4. Connect PIR sensor output in series to a 330 Ohm resistor, LED, and ground said loop 5. Have one person stand 50 feet from drone and begin walking to drone 6. Probed output voltage using multimeter from sensor during this period should be at 0.0(+/-.3)V 7. Once person approaching drone is within 20 ft , probed output voltage using multimeter from sensor should be 3.3(+/-.3V) | **2.5** |
| **Doppler Radar Pre-Amplifier Circuit**   1. The two stage amplifier must have a gain of 12317 +/- 300. The first stage must have a gain of 101 +/- 10. The second stage must have a gain of 122 +/- 10. 2. Must have a passband from 2.82 Hz (+/- 1.82Hz) to 701 Hz +/- 100 Hz. This corresponds to outputting a high for objects moving in the sensor range at speeds between the respective lower bound and upper bound ranges of .032 to.148 mph and 19.19 to 25.5 mph | 1. **Verification process for item 1**     1. Build the two stage amplifier according to figure 12 in Sachin’s lab notebook on a breadboard. Figure attached at the end of the RV table as well.    2. Disconnect the connection shown in the schematic that connects the output of the first opamp to the lowpass filter that leads into the positive terminal of the second opamp    3. Test the gain of the first OpAmp by connecting the output of the stage to an oscilloscope    4. Using a waveform generator, input a signal with a frequency of 300 Hz (+/- 5Hz) (well within the passband) and an amplitude of 5 mV (+/-.5mV)    5. Verify that the output voltage has an amplitude of .5V (+/- .1V)    6. Test the gain of the second OpAmp by connecting the output of it to an oscilloscope    7. Input a signal with a frequency of 300 Hz (well within the passband) and an amplitude of 10mV    8. Verify that the output voltage has a voltage peak to peak of 2.4 V (+/- .4V) 2. **Verification process for item 2** 3. Now input a signal with a frequency of 1300 Hz and an amplitude of 5mV to the first stage of the the opamp 4. Verify that the output voltage now has an amplitude of half the voltage at 300 Hz (+/-.2V) 5. Now input a signal with a frequency of 1300 Hz and an amplitude of 10 mV to the 2nd stage of the the opamp 6. Verify that the output voltage now has a peak to peak voltage of half the voltage at 300 hz (+/- .2 V) | **10** |
| **Doppler Radar Sensor**   1. Must see an output waveform constant output signal with a DC offset of 2.5V (+/- .5V) when there is no motion 2. Must see an output waveform with a peak to peak voltage of 4V (+/-.5V) when motion is detected 3. Must output high of 3.3 V (+/- .3V) when motion of something deemed a threat is detected. A threat is anything with a peak to peak voltage greater than 2 volts peak to peak (+/- .5 volts). Must output low of 0.0 V (+/-.3 V) during other cases. | 1. **Verification process for item 1** 2. Connect the Doppler sensor to the pre-amplifier circuit as specified by the attached schematic. 3. Connect the output of the preamplifier to an oscilloscope 4. Have one person stand 40 feet from the sensor and stand completely still 5. Output signal on the oscilloscope should be a constant output signal with a DC offset between 2 and 3 volts 6. **Verification process for item 2** 7. Subject then begins walking towards the drone and should notice an increase in a sinusoidal like output signal with a peak to peak voltage no greater than 4V. This output will increase as the subject gets closer to the sensor until it saturates. 8. Once person is within 15ft (+/-5ft) sensor output should be saturated 9. **Verification process for item 3** 10. Now connect the output of the sensor to one of the ADC pins on the ODROID 11. Run the provided testscript which will analyze the voltage level of the input signal 12. If the input signal is greater than 2 volts peak to peak (+/- .5 Vs), output a high on one of the GPIO pins. Else, output a low on the same GPIO pin. 13. Another person will probe this voltage and see either 3.3V (+/-.3V) or 0.0V (+/- .3V) depending on the decision made in part c | **10** |
| **System Threat Detection**   1. Sensors must both detect high within the same 1.5 seconds | \*Note that it must first be shown that both sensors operate correctly on their own first for this verification to work properly   1. **Verification process for item 1**    1. Connect PIR sensor to 5V power and GND    2. Connect output of PIR sensor to a 330 ohm resistor and LED    3. Connect the doppler radar as described in the previous module    4. Take the output of doppler sensor after the digital decision making and feed it through another 330 ohm resistor in series with an LED    5. Position both sensors next to each other, facing in the same direction and have a person (1) start still 40 feet away    6. Have another person (2) behind sensor with a stop watch    7. Have person (1) begin walking toward the drone    8. Have person (2) start stop watch when either LED is lit    9. Ensure that both LEDs are high within 1.5 seconds | **2.5** |

**5.1.2 OpenCV Algorithms**

|  |  |  |
| --- | --- | --- |
| Requirements | Verification | Points |
| Total .25fps processing speed | 1. Python testbench wrapper using time module to measure time elapsed between image frame capture and output from Border decryption code 2. Ensure total elapsed time less than 4 seconds per input image maximum. 3. Testbench also used for color decryption verification. | **7.5** |
| Distance calculation using focal length with accuracy of 85(+/-5)% at 10ft | 1. Bring drone to height of 10 feet using given drone software 2. Run distance algorithm to get estimated height (given initial data point to find camera focal length) 3. Calculate the percent difference of the two heights to see if only 15%(+/-5)% different, providing enough room for the drone to slow down enough to land. | **2.5** |
| Color decryption of border works at 10 ft off the ground with 80(+/-5)% accuracy. | 1. Use random collection test set of images at varying heights surrounded by different backgrounds with different encoded strings and different angles of rotation with testbench for timing verification. 2. Determine success rate by determining average success trials:   # of successes / # of trials  3. Some test images from the collection must include more than one landing pad, only one of which is correct. | **5** |

**5.1.3 MAVProxy Algorithms**

|  |  |  |
| --- | --- | --- |
| Requirements | Verification | Points |
| Ensure that the control algorithm gets the drone to land at the center of the landing pad +/- 2.5 ft from the middle. | 1. Modify initial takeoff software procedure so that drone flies at a lower height of 5 feet off the ground as opposed to standard operating 30 feet to facilitate easy processing of the landing pad border 2. Run the main script without landing pad code verification to determine where the drone reaches in relation to the center of the landing pad. | **5** |

**5.1.4 Phone App**

|  |  |  |
| --- | --- | --- |
| Requirements | Verification | Points |
| User confirmation sent and received within 1 minute(+-1 minute) from GCM client to Odroid XMPP server and vice-versa. | 1. Send packet from GCM client on app with original UTC timestamp, compare with utc time when received on Odroid C1+ XMPP server. Time difference should be negligible, timestamps must differ by no more than 0-2 minutes. 2. Send packet from Odroid server with utc timestamp , compare with UTC timestamp on GCM client on mobile app when received. Time difference between internal clocks should be negligible. | **3** |
| Works on Android Lollipop devices. | 1. Use simulated phone in Android Studio to run application on. Simulated phone should be able to run as much of the app as possible without network connection as possible without error. | **2** |

**Appendix B: Source Code**

**stable\_image.cpp**

#include <opencv2/highgui/highgui.hpp>  
#include "opencv2/opencv.hpp"  
#include <unistd.h>  
  
#define DEBUG true  
  
using namespace *cv*;  
  
/\* copy over frame1 to frame2 and re-populate (no longer necessary) \*/  
void capture\_img(VideoCapture cam, Mat \*frame1, Mat \*frame2, int firstFrame) {  
 \*frame2 = (\*frame1).clone();  
 cam >> \*frame1;  
}  
  
int main(int argc, char \*argv[]) {  
 if ((!DEBUG) && (argc < 2)) {  
 std::cout << "invalid args" << std::endl;  
 }  
  
 /\* ODROID should only have one camera, ignore laptop webcam \*/  
 VideoCapture cam;  
 if (DEBUG) {  
 cam = VideoCapture(1);  
 }  
 else {  
 cam = VideoCapture(0);  
 }  
  
 /\* make sure camera correctly opens \*/  
 if(!cam.isOpened()) {  
 std::cout << "Unable to open camera." << std::endl;  
 }  
  
 /\* camera *settings* (PNG) (512 x 512 image) (50% exposure) (20% brightness) \*/  
 cam.set(CV\_CAP\_PROP\_FOURCC,CV\_FOURCC('M','P','N','G'));  
 cam.set(CV\_CAP\_PROP\_FRAME\_WIDTH, 512);  
 cam.set(CV\_CAP\_PROP\_FRAME\_HEIGHT, 512);  
 cam.set(CV\_CAP\_PROP\_EXPOSURE, 50);  
 cam.set(CV\_CAP\_PROP\_BRIGHTNESS, 20);  
  
 Mat image1;  
 Mat image2;  
 int firstFrame = 0;  
 int frameIdx = 0;  
  
 /\* start processing frames \*/  
 while (1) {  
 capture\_img(cam, &image1, &image2, firstFrame);  
  
 /\* ignore first *frame* (no longer needed) \*/  
 if (firstFrame == 0) {  
 firstFrame = 1;  
 }  
 else {  
 Mat blurred;  
  
 /\* borrowed from http://stackoverflow*.com*/questions/4993082/how-to-sharpen-an-image-in-opencv [14] \*/  
 /\* apply blurring to image capture to account for small movements \*/  
 double sigma = 5, threshold = 5, amount = 1;  
 GaussianBlur(image2, blurred, Size(), sigma, sigma);  
 Mat lowContrastMask = abs(image2 - blurred) < threshold;  
 Mat sharpened = image2\*(1+amount) + blurred\*(-amount);  
 image2.copyTo(sharpened, lowContrastMask);  
  
 if (argc < 2) {  
 imwrite("tpp"+std::to\_string(frameIdx)+"*.png*", sharpened);  
 }  
 else {  
 imwrite(argv[1], sharpened);  
 }  
 frameIdx += 1;  
 }  
 sleep(2);  
 /\* small sleep to allow for scene to change \*/  
 //usleep(500);  
 }  
 return 0;  
}

**normalize\_image.cpp**

#include <algorithm>  
#include <vector>  
  
#include <opencv2/core/core.hpp>  
#include "opencv2/opencv.hpp"  
  
#include "gen\_border.hpp"  
  
#define IMG\_HEIGHT 512  
#define IMG\_WIDTH 512  
#define ROW 10  
#define COL 10  
#define PURPLE 4  
  
using namespace *cv*;  
  
/\* averages RGB channels for normalization \*/  
void rgbNormalize(Mat \*bgr\_image) {  
 for (int i = 0; i < bgr\_image->rows; i++) {  
 for (int k = 0; k < bgr\_image->cols; k++) {  
 float reVal = (float)bgr\_image->at<cv::Vec3b>(i,k)[2];  
 float blVal = (float)bgr\_image->at<cv::Vec3b>(i,k)[1];  
 float grVal = (float)bgr\_image->at<cv::Vec3b>(i,k)[0];  
 float sum = reVal + blVal + grVal;  
  
 bgr\_image->at<cv::Vec3b>(i,k)[2] = (int)((reVal/sum)\*255.0);  
 bgr\_image->at<cv::Vec3b>(i,k)[1] = (int)((blVal/sum)\*255.0);  
 bgr\_image->at<cv::Vec3b>(i,k)[0] = (int)((grVal/sum)\*255.0);  
 }  
 }  
}  
  
/\* borrowed from http://stackoverflow.com/questions/24341114/simple-illumination-correction-in-images-opencv-c [19] \*/  
/\* https://en.wikipedia.org/wiki/Adaptive\_histogram\_equalization#Contrast\_Limited\_AHE [12] \*/  
/\* convert image to lab space and evaluate CLAHE to spread colors over entire color spectrum \*/  
void claheNormalize(Mat \*bgr\_image) {  
 Mat lab\_image;  
 cvtColor(\*bgr\_image, lab\_image, CV\_BGR2Lab);  
  
 /\* split original image into 3 color planes \*/  
 std::vector<Mat> lab\_planes(3);  
 split(lab\_image, lab\_planes);  
  
 /\* create histogram CLAHE to normalize \*/  
 Ptr<CLAHE> clahe = createCLAHE();  
 clahe->setClipLimit(4);  
 Mat dst;  
 clahe->apply(lab\_planes[0], dst);  
  
 /\* merge planes and convert back to normal bgr \*/  
 dst.copyTo(lab\_planes[0]);  
 merge(lab\_planes, lab\_image);  
  
 Mat image\_clahe;  
 cvtColor(lab\_image, \*bgr\_image, CV\_Lab2BGR);  
}  
  
/\* leverage known color to scale pixels back to original color \*/  
void referenceNormalize(Mat \*bgr\_image) {  
 unsigned int rul = 0, gul = 0, bul = 0;  
 unsigned int rur = 0, gur = 0, bur = 0;  
 unsigned int rll = 0, gll = 0, bll = 0;  
 unsigned int rlr = 0, glr = 0, blr = 0;  
 float avg\_red = 0.0, avg\_blue = 0.0, avg\_green = 0.0;  
 float num\_pixels = ((2.0\*(IMG\_HEIGHT/ROW))\*(2.0\*(IMG\_WIDTH/COL)));  
 float factor\_r = 0.0, factor\_b = 0.0, factor\_g = 0.0;  
  
 /\* upper left \*/  
 for (int i = 0; i < (IMG\_HEIGHT/ROW)\*2; i++) {  
 for (int k = 0; k < (IMG\_WIDTH/COL)\*2; k++) {  
 bul += bgr\_image->at<cv::Vec3b>(i,k)[0];  
 gul += bgr\_image->at<cv::Vec3b>(i,k)[1];  
 rul += bgr\_image->at<cv::Vec3b>(i,k)[2];  
 }  
 }  
  
 /\* upper right \*/  
 for (int i = 0; i < (IMG\_HEIGHT/ROW)\*2; i++) {  
 for (int k = (IMG\_WIDTH-(2\*(IMG\_WIDTH/COL))); k < IMG\_WIDTH; k++) {  
 bur += bgr\_image->at<cv::Vec3b>(i,k)[0];  
 gur += bgr\_image->at<cv::Vec3b>(i,k)[1];  
 rur += bgr\_image->at<cv::Vec3b>(i,k)[2];  
 }  
 }  
  
 /\* lower left \*/  
 for (int i = (IMG\_HEIGHT-(2\*(IMG\_HEIGHT/ROW))); i < IMG\_HEIGHT; i++) {  
 for (int k = 0; k < (IMG\_WIDTH/COL)\*2; k++) {  
 bll += bgr\_image->at<cv::Vec3b>(i,k)[0];  
 gll += bgr\_image->at<cv::Vec3b>(i,k)[1];  
 rll += bgr\_image->at<cv::Vec3b>(i,k)[2];  
 }  
 }  
  
 /\* lower right \*/  
 for (int i = (IMG\_HEIGHT-(2\*(IMG\_HEIGHT/ROW))); i < IMG\_HEIGHT; i++) {  
 for (int k = (IMG\_WIDTH-(2\*(IMG\_WIDTH/COL))); k < IMG\_WIDTH; k++) {  
 blr += bgr\_image->at<cv::Vec3b>(i,k)[0];  
 glr += bgr\_image->at<cv::Vec3b>(i,k)[1];  
 rlr += bgr\_image->at<cv::Vec3b>(i,k)[2];  
 }  
 }  
  
 /\* average each corner \*/  
 rul /= num\_pixels;  
 gul /= num\_pixels;  
 bul /= num\_pixels;  
  
 rur /= num\_pixels;  
 gur /= num\_pixels;  
 bur /= num\_pixels;  
  
 rll /= num\_pixels;  
 gll /= num\_pixels;  
 bll /= num\_pixels;  
  
 rlr /= num\_pixels;  
 glr /= num\_pixels;  
 blr /= num\_pixels;  
  
 /\* average over four corners \*/  
 avg\_red = (rul+rur+rll+rlr)/4.0;  
 avg\_blue = (bul+bur+bll+blr)/4.0;  
 avg\_green = (gul+gur+gll+glr)/4.0;  
  
 /\* scaling factor to apply to all pizels in image \*/  
 factor\_b = (COLOR\_TABLE[PURPLE][0])/avg\_blue;  
 factor\_g = (COLOR\_TABLE[PURPLE][1])/avg\_green;  
 factor\_r = (COLOR\_TABLE[PURPLE][2])/avg\_red;  
  
 /\* scale each pixel \*/  
 for (int i = 0; i < bgr\_image->rows; i++) {  
 for (int k = 0; k < bgr\_image->cols; k++) {  
 bgr\_image->at<cv::Vec3b>(k,i)[0] = std::min(255, int(bgr\_image->at<cv::Vec3b>(k,i)[0]\*factor\_b));  
 bgr\_image->at<cv::Vec3b>(k,i)[1] = std::min(255, int(bgr\_image->at<cv::Vec3b>(k,i)[1]\*factor\_g));  
 bgr\_image->at<cv::Vec3b>(k,i)[2] = std::min(255, int(bgr\_image->at<cv::Vec3b>(k,i)[2]\*factor\_r));  
 }  
 }  
}  
  
/\* based on http://docs.opencv.org/2.4/doc/tutorials/core/basic\_linear\_transform/basic\_linear\_transform.html [20] \*/  
/\* brighten/darken image using saturation \*/  
void brighten(Mat image, Mat \*bgr\_image, int beta=-50) {  
 double alpha = 1.0;  
  
 /\* saturate each pixel to brighten/darken image \*/  
 for(int y = 0; y < image*.rows*; y++ ){  
 for(int x = 0; x < image*.cols*; x++ ) {  
 /\* apply to each color channel \*/  
 for(int c = 0; c < 3; c++ ) {  
 bgr\_image->at<Vec3b>(y,x)[c] =  
 saturate\_cast<uchar>(alpha\*(image*.at*<Vec3b>(y,x)[c] ) + beta);  
 }  
 }  
 }  
}  
  
int main(int argc, char\*\* argv)  
{  
 if (argc < 3) {  
 std::cout << "invalid args" << std::endl;  
 }  
  
 else {  
 Mat image;  
 Mat bgr\_image;  
  
 int idx = 0;  
  
 /\* darken the image and then rgb *normalize* (last iter will brighten) \*/  
 while (idx < 2) {  
 image = imread( argv[1]);  
 bgr\_image = Mat::zeros( image.size(), image.type() );  
 if (idx < 1) {  
 brighten(image, &bgr\_image, 0);  
 rgbNormalize(&bgr\_image);  
 }  
 else {  
 /\* clahe normalize and then final rgb \*/  
 brighten(image, &bgr\_image, 0);  
 claheNormalize(&bgr\_image);  
 //rgbNormalize(&bgr\_image);  
 }  
 imwrite(argv[2], bgr\_image);  
 idx += 1;  
 }  
 std::cout << "success" << std::endl;  
 }  
 return 0;  
}

**gen\_border.cpp**

#include <iostream>  
#include <math.h>  
  
#include <opencv2/highgui/highgui.hpp>  
#include <opencv2/opencv.hpp>  
  
#include <openssl/sha.h>  
  
#include "gen\_border.hpp"  
  
/\* defines for image handling \*/  
#define IMG\_HEIGHT 512  
#define IMG\_WIDTH 512  
#define OFFSET\_X 20  
#define OFFSET\_Y 20  
#define ROW 10  
#define COL 10  
  
using namespace *cv*;  
  
/\* consts for border generation \*/  
const short PADDING\_BITS[] = {0, 1, 8, 9, 10, 11, 18, 19, 44, 45, 54, 55, 80, 81, 88, 89, 90, 91, 98, 99};  
const short PAD\_LEN\_MID = *4*;  
const short PAD\_LEN\_END = *16*;  
  
/\* encode blocks back to corresponding color \*/  
char toHex(int c) {  
 if (c < 10) {  
 return '0'+c;  
 }  
 else {  
 return 'A'+(c-10);  
 }  
}  
  
/\* percentage of matching characters (only equal length strings) \*/  
float similarity(string s1, string s2) {  
 int special\_idx = 0;  
 int total\_wrong = 0;  
 for (int i = 0; i < s1.length(); i++) {  
 if (i == PADDING\_BITS[special\_idx]) {  
 special\_idx += 1;  
 }  
 else {  
 if (s1[i] != s2[i]) {  
 total\_wrong += 1;  
 }  
 }  
 }  
 return 1-(float)total\_wrong/(s1.length());  
}  
  
string generateBorder(const unsigned char \*str, string output\_file, bool create\_img = false) {  
 /\* quad digits for now\*/  
 unsigned char hash[20];  
 char quad\_digs[ROW\*COL+1];  
 int special\_idx = 0, idx = 0, hash\_idx = 0, write\_idx = 0, h1 = -1, h2 = -1, l1 = -1, l2 = -1;  
 /\* create SHA1 encoded string for security \*/  
 SHA1(str, sizeof(str) - 1, hash);  
 while (idx < ROW\*COL) {  
 /\* ignore padding blocks \*/  
 if (idx == PADDING\_BITS[special\_idx]) {  
 special\_idx += 1;  
 quad\_digs[idx] = '4';  
 }  
 else {  
 /\* take a hex digit and break into base 4 (4 primary colors) \*/  
 if (h1 == -1) {  
 int val = hash[hash\_idx];  
 h1 = val % 4;  
 val /= 4;  
 h2 = val % 4;  
 val /= 4;  
 l1 = val % 4;  
 val /= 4;  
 l2 = val % 4;  
 val /= 4;  
 hash\_idx += 1;  
 }  
 if (l2 != -1) {  
 quad\_digs[idx] = l2+'0';  
 l2 = -1;  
 }  
 else if (l1 != -1) {  
 quad\_digs[idx] = l1+'0';  
 l1 = -1;  
 }  
 else if (h2 != -1) {  
 quad\_digs[idx] = h2+'0';  
 h2 = -1;  
 }  
 else if (h1 != -1) {  
 quad\_digs[idx] = h1+'0';  
 h1 = -1;  
 }  
 }  
 idx += 1;  
 }  
 /\* C-style string \*/  
 quad\_digs[100] = '\0';  
  
 /\* create the black image with white border \*/  
 if (create\_img) {  
 Mat *img*(IMG\_HEIGHT+OFFSET\_Y\*12, IMG\_WIDTH+OFFSET\_X\*12, CV\_8UC3, Scalar(0,0,0));  
 rectangle(img, Point(0+OFFSET\_Y\*2, 0+OFFSET\_X\*2), Point(IMG\_HEIGHT+OFFSET\_Y\*10, IMG\_WIDTH+OFFSET\_X\*10), Scalar(255,255,255), OFFSET\_Y\*2);  
 rectangle(img, Point(0+OFFSET\_Y\*4, 0+OFFSET\_X\*4), Point(IMG\_HEIGHT+OFFSET\_Y\*8, IMG\_WIDTH+OFFSET\_X\*8), Scalar(0,0,0), OFFSET\_Y\*2);  
 /\* color in the blocks \*/  
 int spec\_idx = 0, hex\_idx = 0, block\_idx = 0, digit;  
 for (int k = 0; k < ROW; k++) {  
 for (int i = 0; i < COL; i++) {  
 digit = quad\_digs[hex\_idx]-'0';  
 Scalar color = COLOR\_TABLE[digit];  
 hex\_idx += 1;  
 block\_idx += 1;  
 rectangle(img, Point(i\*(IMG\_HEIGHT/ROW)+OFFSET\_Y\*6, k\*(IMG\_WIDTH/COL)+OFFSET\_X\*6), Point((i+1)\*(IMG\_HEIGHT/ROW)+OFFSET\_Y\*6, (k+1)\*(IMG\_WIDTH/COL)+OFFSET\_X\*6), color, -1);  
 }  
 }  
  
 /\* save template \*/  
 imwrite(output\_file, img);  
 }  
 return quad\_digs;  
}  
  
void decodeBorder(string input\_file, string toFind) {  
 /\* load the image \*/  
 Mat img = imread(input\_file, CV\_LOAD\_IMAGE\_COLOR);  
 resize(img, img, Size(IMG\_WIDTH, IMG\_HEIGHT));  
  
 /\* result strings \*/  
 char default\_z[101];  
 for (int i = 0; i < ROW\*COL; i++) {  
 default\_z[i] = 'Z';  
 }  
 default\_z[100] = '\0';  
 string results[] = {default\_z, default\_z, default\_z, default\_z};  
  
 /\* threshold the values \*/  
 Mat hsv\_color;  
 hsv\_color.create(1,1,CV\_8UC(3));  
 Mat *hsv\_img*(IMG\_HEIGHT, IMG\_WIDTH, CV\_LOAD\_IMAGE\_COLOR, Scalar(0,0,0));  
 Mat *mask*(IMG\_HEIGHT, IMG\_WIDTH, CV\_8UC3, Scalar(0,0,0));  
 Mat *or\_res*(IMG\_HEIGHT+OFFSET\_Y, IMG\_WIDTH+OFFSET\_X, CV\_8UC3, Scalar(0,0,0));  
  
 for (int bgr\_idx = 0; bgr\_idx < COLOR\_TABLE\_LEN; bgr\_idx++) {  
 /\* convert img and color to hsv, get inRange and bitwise AND with mask\*/  
 cvtColor(img, hsv\_img, CV\_BGR2HSV);  
 Scalar bgr = COLOR\_TABLE[bgr\_idx];  
 Mat *color*(1, 1, CV\_8UC(3), bgr);  
 cvtColor(color, hsv\_color, CV\_BGR2HSV);  
 inRange(hsv\_img, Scalar(hsv\_color*.at*<Vec3b>(0,0)[0]-10,0,0), Scalar(hsv\_color*.at*<Vec3b>(0,0)[0]+10,255,255), mask);  
 Mat res;  
 bitwise\_and(img,img,res, mask=mask);  
   
 /\* decode blocks from information \*/  
 vector<int> l\_blocks[4];  
 for (int i = 0; i < res*.rows*; i++) {  
 for (int k = 0; k < res*.cols*; k++) {  
 /\* check if value > 0 \*/  
 int b = (int)res*.at*<Vec3b>(i, k)[0];  
 int g = (int)res*.at*<Vec3b>(i, k)[1];  
 int r = (int)res*.at*<Vec3b>(i, k)[2];  
 if ((b > 0) || (g > 0) || (r > 0)) {  
 int n\_row = floor(i\*(float)ROW/IMG\_HEIGHT);  
 int n\_col = floor(k\*(float)COL/IMG\_WIDTH);  
 int blocks[4] = {int(n\_row\*ROW+n\_col), (ROW\*COL-1)-int(n\_col\*COL+(ROW-1-n\_row)), (ROW\*COL-1)- int(n\_row\*ROW+n\_col), int(n\_col\*COL+(ROW-1-n\_row))};  
  
 /\* check if block already used for this rotation angle \*/  
 for (int b\_idx = 0; b\_idx < 4; b\_idx++) {  
 bool flag = false;  
 for (int lb\_idx = 0; lb\_idx < l\_blocks[b\_idx].size(); lb\_idx++) {  
 if (l\_blocks[b\_idx][lb\_idx] == blocks[b\_idx]) {  
 flag = true;  
 break;  
 }  
 }  
 if (flag == true) {  
 continue;  
 }  
  
 /\* determine boundaries \*/  
 float b\_x\_left\_boundary = n\_col\*((float)IMG\_WIDTH/COL);  
 float b\_x\_right\_boundary = ((n\_col+1)\*((float)IMG\_WIDTH/COL));  
 float b\_x\_left\_mid = b\_x\_left\_boundary + ((float)IMG\_HEIGHT/ROW\*.4);  
 float b\_x\_right\_mid = b\_x\_left\_boundary + ((float)IMG\_HEIGHT/ROW\*.6);  
  
 float b\_y\_top\_boundary = n\_row\*((float)IMG\_HEIGHT/ROW);  
 float b\_y\_bot\_boundary = ((n\_row+1)\*((float)IMG\_HEIGHT/ROW));  
 float b\_y\_top\_mid = b\_y\_top\_boundary + ((float)IMG\_WIDTH/COL\*.4);  
 float b\_y\_bot\_mid = b\_y\_top\_boundary + ((float)IMG\_WIDTH/COL\*.6);  
  
 /\* if in middle of boundary, check if right color or first guess \*/  
 if ((k >= b\_x\_left\_mid) && (k <= b\_x\_right\_mid) && (i >= b\_y\_top\_mid) && (i <= b\_y\_bot\_mid)) {  
 l\_blocks[b\_idx].push\_back(blocks[b\_idx]);  
 bool block\_set\_flag = false;  
 if (results[b\_idx][blocks[b\_idx]] == 'Z') {  
 block\_set\_flag = true;  
 }  
 else if (toFind[b\_idx] == toHex(bgr\_idx)) {  
 block\_set\_flag = true;  
 }  
 if (block\_set\_flag) {  
 rectangle(or\_res, Point(int(b\_x\_left\_boundary), int(b\_y\_top\_boundary)), Point(int(b\_x\_right\_boundary), int(b\_y\_bot\_boundary)), COLOR\_TABLE[bgr\_idx], -1);  
 results[b\_idx][blocks[b\_idx]] = toHex(bgr\_idx);  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
  
 /\* determine best rotation string \*/  
 float dist\_score = 0.0;  
 int idx = -1;  
 for (int i = 0; i < 4; i++) {  
 float new\_score = similarity(results[i], toFind);  
 if (new\_score > dist\_score) {  
 dist\_score = new\_score;  
 idx = i;  
 }  
 }  
 std::cout << "result " << dist\_score << std::endl;  
}  
  
int main(int argc, char\* argv[]) {  
 if (argc < 6) {  
 std::cout << "invalid args" << std::endl;  
 }  
 else {  
 string should\_encode = argv[1];  
 string should\_decode = argv[2];  
 string em\_addr\_str = argv[3];  
 const unsigned char \*email\_address = new unsigned char[em\_addr\_str.length()+1];  
 strcpy((char \*)email\_address,em\_addr\_str.c\_str());  
 string encode\_filename = argv[4];  
 string decode\_filename = argv[5];  
 if (should\_encode == "Y") {  
 generateBorder(email\_address, encode\_filename, true);  
 }  
 if (should\_decode == "Y") {  
 string encoded\_hash = generateBorder(email\_address, encode\_filename, false);  
 decodeBorder(decode\_filename, encoded\_hash);  
 }  
 }  
 return 0;  
}

**gen\_border.hpp**

#ifndef BORDER\_H  
#define BORDER\_H  
  
/\* perfect lighting \*/  
const cv::Scalar COLOR\_TABLE[] = {cv::Scalar(0,34,255), cv::Scalar(64,247,2), cv::Scalar(255,30,0), cv::Scalar(3, 251, 255), cv::Scalar(255,0,174)};  
  
/\* saturated lighting \*/  
//const cv::Scalar COLOR\_TABLE[] = {cv::Scalar(104, 0, 150), cv::Scalar(105,102,47), cv::Scalar(147,72,35), cv::Scalar(62, 95, 96), cv::Scalar(255,0,174)};  
  
const short COLOR\_TABLE\_LEN = 5;  
  
#endif

**gen\_template.cpp**

#include <iostream>  
#include <math.h>  
#include <limits>  
#include <sstream>  
  
#include <opencv2/highgui/highgui.hpp>  
#include "opencv2/opencv.hpp"  
  
#include "gen\_border.hpp"  
  
#define IMG\_HEIGHT 512  
#define IMG\_WIDTH 512  
#define COLOR\_TABLE\_LEN 5  
  
using namespace *cv*;  
  
const float pi = (float)(*22*/*7*);

/\* borrowed from <http://docs.opencv.org/3.1.0/da/d6e/tutorial_py_geometric_transformations.html#gsc.tab=0> [5] \*/  
Mat sub\_img(Mat img, Point2f center, float theta,int width, int height){  
 Mat out;  
 Point *pt*(width/2, height/2);  
  
 /\* rotates image by *theta*(radians) \*/  
 Mat M = getRotationMatrix2D(pt,theta\*180/pi,1);  
 Size *s\_1* (height,width);  
 warpAffine(img,out,M,s\_1);  
  
 return out;  
}  
  
void white\_out(Mat img,Point top\_l,Point bottom\_r, vector<Point> contour){  
 /\* white out unwanted pixels in bounded rectangle but outside of contour \*/  
 for(int x = top\_l*.x*; x< bottom\_r*.x*; x++){  
 for(int y = top\_l*.y*; y<bottom\_r*.y*; y++){  
 Point *test* (x,y);  
 double dist = pointPolygonTest(contour,test,true);  
 if(dist < 1) {  
 img*.at*<Vec3b>(y,x) = Vec3b(255,255,255);  
 }  
 }  
 }  
}  
  
void find\_matching\_contour(Mat im, vector< vector<Point> >\* contours, int\* best\_cnt\_idx )  
{  
 Mat imgray;  
 Mat thresh;  
 vector<Vec4i> hierarchy;  
 double parent\_area, child\_area;  
 float ratio, min\_ratio = -1;  
  
 /\* convert to grayscale, find contours \*/  
 cvtColor(im,imgray,COLOR\_BGR2GRAY);  
 threshold(imgray,thresh,127,255,0);  
 findContours(thresh,\*contours,hierarchy,CV\_RETR\_TREE,CV\_CHAIN\_APPROX\_SIMPLE);  
  
 \*(best\_cnt\_idx) = -1;  
 for(int x = 0; x<hierarchy.size(); x++){  
 if((int)hierarchy[x][3] == -1) {  
 continue;  
 }  
  
 parent\_area = contourArea((\*contours)[hierarchy[x][3]]);  
 child\_area = contourArea((\*contours)[x]);  
  
 /\* child insignificant \*/  
 if(child\_area<200) {  
 continue;  
 }  
  
 /\* desired ratio range between white outline and color border \*/  
 ratio = (float)parent\_area/child\_area;  
 if(ratio > .85 && ratio < 2.0) {  
 //if (ratio > min\_ratio) {  
 \*(best\_cnt\_idx) = x;  
 min\_ratio = ratio;  
 break;  
 //}  
 }  
 }  
  
}  
  
void find\_contour\_rect(vector< vector<Point> > contours, int best\_cnt\_idx, Point\* fst, Point\* snd){  
 int fst\_x = std::numeric\_limits<int>::max();  
 int fst\_y = std::numeric\_limits<int>::max();  
 int snd\_x = 0;  
 int snd\_y = 0;  
  
 /\* setup coordinates to *crop* (top left, bottom right) \*/  
 for (int x = 0; x < contours[best\_cnt\_idx].size(); x++){  
 if(contours[best\_cnt\_idx][x]*.x* < fst\_x)  
 fst\_x = contours[best\_cnt\_idx][x]*.x*;  
 if(contours[best\_cnt\_idx][x]*.x* > snd\_x)  
 snd\_x = contours[best\_cnt\_idx][x]*.x*;  
 if(contours[best\_cnt\_idx][x]*.y* < fst\_y)  
 fst\_y = contours[best\_cnt\_idx][x]*.y*;  
 if(contours[best\_cnt\_idx][x]*.y* > snd\_y)  
 snd\_y = contours[best\_cnt\_idx][x]*.y*;  
 }  
  
 fst->x = fst\_x;  
 fst->y = fst\_y;  
 snd->x = snd\_x;  
 snd->y = snd\_y;  
}  
  
void find\_contour\_rotated(Mat im, vector< vector<Point> >\* contours, int\* child, int\* bst\_idx){  
 Mat imgray;  
 Mat thresh;  
 vector<Vec4i> hierarchy;  
 int bst\_area = -1;  
 \*bst\_idx = -1;  
  
 /\* convert to grayscale and find contours \*/  
 cvtColor(im,imgray,COLOR\_BGR2GRAY);  
 threshold(imgray,thresh,127,255,0);  
 findContours(thresh,\*contours,hierarchy,CV\_RETR\_TREE,CV\_CHAIN\_APPROX\_SIMPLE);  
  
 /\* find largest contour area \*/  
 for(int x = 0 ; x<contours->size(); x++){  
 double area = contourArea((\*contours)[x]);  
 if(area > bst\_area){  
 bst\_area = area;  
 \*bst\_idx = x;  
 }  
 }  
  
 \*child = hierarchy[\*bst\_idx][2];  
}  
  
Mat recrop(Mat img, int lower\_bound, int upper\_bound) {  
 Point *fst*(IMG\_WIDTH\*2, IMG\_HEIGHT\*2);  
 Point *snd*(-1, -1);  
  
 Mat hsv\_color;  
 hsv\_color.create(1,1,CV\_8UC(3));  
 Mat *hsv\_img*(IMG\_HEIGHT, IMG\_WIDTH, CV\_LOAD\_IMAGE\_COLOR, Scalar(0,0,0));  
 Mat *mask*(IMG\_HEIGHT, IMG\_WIDTH, CV\_8UC3, Scalar(0,0,0));  
  
 /\* find purple borders \*/  
 for (int i = COLOR\_TABLE\_LEN-1; i < COLOR\_TABLE\_LEN; i++) {  
 cvtColor(img, hsv\_img, CV\_BGR2HSV);  
 Scalar bgr = COLOR\_TABLE[i];  
 Mat *color*(1, 1, CV\_8UC(3), bgr);  
 cvtColor(color, hsv\_color, CV\_BGR2HSV);  
 inRange(hsv\_img, Scalar(hsv\_color*.at*<Vec3b>(0,0)[0]-10,lower\_bound,lower\_bound), Scalar(hsv\_color*.at*<Vec3b>(0,0)[0]+10,upper\_bound,upper\_bound), mask);  
 Mat res;  
 bitwise\_and(img,img,res, mask=mask);  
  
 for (int i = 0; i < res*.rows*; i++) {  
 for (int k = 0; k < res*.cols*; k++) {  
 int b = (int)res*.at*<Vec3b>(i, k)[0];  
 int g = (int)res*.at*<Vec3b>(i, k)[1];  
 int r = (int)res*.at*<Vec3b>(i, k)[2];  
 if ((b > 0) || (g > 0) || (r > 0)) {  
 if (k < fst*.x*) {  
 fst*.x* = k;  
 }  
 if (k > snd*.x*) {  
 snd*.x* = k;  
 }  
 if (i < fst*.y*) {  
 fst*.y* = i;  
 }  
 if (i > snd*.y*) {  
 snd*.y* = i;  
 }  
 }  
 }  
 }  
 }  
  
 /\* crop around purple borders and resize again --> img should be rotated already so should be okay \*/  
 Mat crop\_img = img(Rect(fst*.x*, fst*.y*,snd*.x*-fst*.x*,snd*.y*-fst*.y*));  
 resize(crop\_img, crop\_img, Size(IMG\_HEIGHT, IMG\_WIDTH));  
 return crop\_img;  
}  
  
int process(string input\_filename, string output\_filename, int lower\_bound, int upper\_bound){  
 Mat img = imread(input\_filename, CV\_LOAD\_IMAGE\_COLOR);  
 try {  
 resize(img, img, Size(IMG\_HEIGHT\*1.2, IMG\_WIDTH\*1.2));  
 }  
 catch (cv::Exception) {  
 std::cout << "file being modified.*.failed*" << std::endl;  
 return -1;  
 }  
 vector< vector<Point> > contours;  
 int best\_cnt\_idx, mult = 0, divis = 30, best\_x, best\_y, best\_w, best\_h;  
 float min\_ratio = std::numeric\_limits<float>::max();  
 Point fst;  
 Point snd;  
 vector<int> bst\_dims;  
 Mat bst\_img;  
  
 /\* find the contour we want to crop \*/  
 find\_matching\_contour(img, &contours, &best\_cnt\_idx);  
 if(best\_cnt\_idx == -1) {  
 std::cout << "missing contour.." << std::endl;  
 return -1;  
 }  
  
 /\* find bounding rect coords and whiteout extra pixels \*/  
 find\_contour\_rect(contours,best\_cnt\_idx, &fst, &snd);  
 Point *new\_fst*(std::max(0, fst*.x*-5),std::max(0, fst*.y*-5));  
 Point *new\_snd*(std::min(snd*.x*+5, IMG\_WIDTH),std::min(snd*.y*+5, IMG\_HEIGHT));  
  
 white\_out(img,new\_fst,new\_snd,contours[best\_cnt\_idx]);  
  
 /\* crop out the desirect rect and resize up \*/  
 Mat crop\_img = img(Rect(new\_fst*.x*, new\_fst*.y*,new\_snd*.x*-new\_fst*.x*,new\_snd*.y*-new\_fst*.y*));  
 resize(crop\_img, crop\_img, Size(IMG\_HEIGHT, IMG\_WIDTH));  
  
 /\* try different rotations in 5 degree increments \*/  
 Point *center*(IMG\_HEIGHT/2,IMG\_WIDTH/2);  
 while(mult <= 15) {  
 Mat rotated = sub\_img(crop\_img,center,pi/divis\*mult,512,512);  
  
 vector< vector<Point> > contours;  
 int child;  
 int bst\_idx;  
 find\_contour\_rotated(rotated, &contours, &child, &bst\_idx);  
 if (child == -1) {  
 mult += 1;  
 continue;  
 }  
 Rect curr\_rect = boundingRect(contours[child]);  
  
 double contour\_area = contourArea(contours[child]);  
 int x = curr\_rect*.x*;  
 int y = curr\_rect*.y*;  
 int w = curr\_rect*.width*;  
 int h = curr\_rect*.height*;  
 double bounded\_area = w\*h;  
 double ratio = bounded\_area/contour\_area;  
  
 /\* ratio between bounding rect and contour --> MIN for 90 degree angles \*/  
 if(ratio < min\_ratio){  
 min\_ratio = ratio;  
 best\_x = x;  
 best\_y = y;  
 best\_w = w;  
 best\_h = h;  
 bst\_img = rotated;  
 }  
 mult++;  
 }  
  
 /\* if able to rotate within reasonable bounds, resize and finish \*/  
 if(min\_ratio < 1.2){  
 Mat im = bst\_img(Rect(best\_x, best\_y, best\_w, best\_h));  
 resize(im, im, Size(IMG\_HEIGHT, IMG\_WIDTH));  
 Mat crop\_final = recrop(im, lower\_bound, upper\_bound);  
 imwrite(output\_filename, crop\_final);  
 }  
  
 else {  
 std::cout << "rotate failed" << std::endl;  
 return -1;  
 }  
 return 0;  
}  
  
int main(int argc, char\* argv[]){  
 if (argc < 5) {  
 std::cout << "invalid args" << std::endl;  
 }  
 else {  
 string input\_filename = argv[1];  
 string output\_filename = argv[2];  
  
 std::istringstream *s3*(argv[3]);  
 int lower\_bound, upper\_bound;  
 if (!(s3 >> lower\_bound)) {  
 std::cout << "invalid args" << std::endl;  
 }  
 std::istringstream *s4*(argv[4]);  
 if (!(s4 >> upper\_bound)) {  
 std::cout << "invalid args" << std::endl;  
 }  
  
 if (process(input\_filename, output\_filename, lower\_bound, upper\_bound) == 0) {  
 std::cout << "completed" << std::endl;  
 }  
 }  
 return 0;  
}

**locate\_border.cpp**

#include <iostream>  
#include <math.h>  
  
#include <opencv2/highgui/highgui.hpp>  
#include <opencv2/opencv.hpp>  
  
#include <openssl/sha.h>  
  
#include "gen\_border.hpp"  
  
/\* defined for image handling \*/  
#define IMG\_HEIGHT 512  
#define IMG\_WIDTH 512  
  
using namespace *cv*;  
  
int outlineBorder(string input\_file) {  
 Mat img;  
 try {  
 img = imread(input\_file, CV\_LOAD\_IMAGE\_COLOR);  
 resize(img, img, Size(IMG\_WIDTH, IMG\_HEIGHT));  
 }  
 catch (cv::Exception) {  
 std::cout << "failed" << std::endl;  
 return -1;  
 }  
  
 Mat hsv\_color;  
 hsv\_color.create(1,1,CV\_8UC(3));  
 Mat *hsv\_img*(IMG\_HEIGHT, IMG\_WIDTH, CV\_LOAD\_IMAGE\_COLOR, Scalar(0,0,0));  
 Mat *mask*(IMG\_HEIGHT, IMG\_WIDTH, CV\_8UC3, Scalar(0,0,0));  
 Mat *or\_res*(IMG\_HEIGHT, IMG\_WIDTH, CV\_8UC3, Scalar(0,0,0));  
  
 unsigned long long num\_pixels = 0;  
 unsigned long long x\_val = 0;  
 unsigned long long y\_val = 0;  
  
 Point avg\_ll = Point(-1, -1), avg\_lr = Point(-1, -1), avg\_ul = Point(-1, -1), avg\_ur = Point(-1, -1);  
 for (int bgr\_idx = 4; bgr\_idx < COLOR\_TABLE\_LEN; bgr\_idx++) {  
 /\* convert img and color to hsv, get inRange and bitwise AND with mask\*/  
 cvtColor(img, hsv\_img, CV\_BGR2HSV);  
 Scalar bgr = COLOR\_TABLE[bgr\_idx];  
 Mat *color*(1, 1, CV\_8UC(3), bgr);  
 cvtColor(color, hsv\_color, CV\_BGR2HSV);  
 inRange(hsv\_img, Scalar(hsv\_color*.at*<Vec3b>(0,0)[0]-10,90,90), Scalar(hsv\_color*.at*<Vec3b>(0,0)[0]+10,255,255), mask);  
 Mat res;  
 bitwise\_and(img,img,res,mask=mask);  
 bitwise\_or(res, or\_res, or\_res);  
  
 for (int i = 0; i < res*.rows*; i++) {  
 for (int k = 0; k < res*.cols*; k++) {  
 int b = (int)res*.at*<Vec3b>(i, k)[0];  
 int g = (int)res*.at*<Vec3b>(i, k)[1];  
 int r = (int)res*.at*<Vec3b>(i, k)[2];  
 if ((b > 0) || (g > 0) || (r > 0)) {  
 x\_val += k;  
 y\_val += i;  
 num\_pixels += 1;  
 }  
 }  
 }  
 }  
 Point center = Point(-1, -1);  
 center*.x* = x\_val/((float)num\_pixels);  
 center*.y* = y\_val/((float)num\_pixels);  
 std::cout << center*.x* << " " << center*.y* << std::endl;  
  
 if ((center*.x* >= 0) && (center*.x* <=156)) {  
 std::cout << "LEFT" << std::endl;  
 }  
 else if ((center*.x* >= 356) && (center*.x* <= 512)){  
 std::cout << "RIGHT" << std::endl;  
 }  
 else if ((center*.y* >= 0) && (center*.y* <= 156)) {  
 std::cout << "UP" << std::endl;  
 }  
 else if ((center*.y* >= 356) && (center*.y* <= 512)) {  
 std::cout << "DOWN" << std::endl;  
 }  
 else {  
 std::cout << "CENTER" << std::endl;  
 }  
 return 0;  
}  
  
int main(int argc, char \*argv[]) {  
 if (argc < 2) {  
 std::cout << "invalid args." << std::endl;  
 }  
 else {  
 string input\_file = argv[1];  
 outlineBorder(input\_file);  
 }  
 return 0;  
}

**Makefile**

CC = g++  
CRYPTO = -lcrypto  
FLAGS = -ggdb -std=c++*11*  
  
all: template border capture locate stabilize normalize  
  
template: gen\_template.cpp  
 $(CC) $(FLAGS) `pkg-config --cflags opencv` -o template gen\_template.cpp `pkg-config --libs opencv` $(CRYPTO)  
  
border: gen\_border.cpp gen\_border.hpp  
 $(CC) $(FLAGS) `pkg-config --cflags opencv` -o border gen\_border.cpp `pkg-config --libs opencv` $(CRYPTO)  
  
capture: capture\_image.cpp  
 $(CC) $(FLAGS) `pkg-config --cflags opencv` -o capture capture\_image.cpp `pkg-config --libs opencv` $(CRYPTO)  
  
locate: locate\_border.cpp  
 $(CC) $(FLAGS) `pkg-config --cflags opencv` -o locate locate\_border.cpp `pkg-config --libs opencv` $(CRYPTO)  
  
stabilize: stable\_image.cpp  
 $(CC) $(FLAGS) `pkg-config --cflags opencv` -o stabilize stable\_image.cpp `pkg-config --libs opencv` $(CRYPTO)  
  
normalize: normalize\_image.cpp  
 $(CC) $(FLAGS) `pkg-config --cflags opencv` -o normalize normalize\_image.cpp `pkg-config --libs opencv` $(CRYPTO)

**devel2.py**

import subprocess  
import math  
import time  
import select  
import json  
import re  
import sys  
import os  
from threading import Thread  
from Queue import Queue, Empty  
  
# CONSTANTS  
HOLD\_TIME = *.149999999999999994*  
SLEEP\_TIME = *.5*  
M\_FT\_CONV = *3.2808*4  
FLYING\_ALTITUDE = *30* #ft  
FENCE\_OUTPUT\_FILE = "./fence.txt"  
LAT\_LON\_DIV = *10000000.0*  
LATITUDE\_CONV = *10000.0*/*90.0*\**3280.4*  
LONGITUDE\_CONV = *364537.40*2  
EMAIL\_ADDRESS = *None*  
THRESHOLD = *.75*  
UPPER = *255*  
OPTIONS = [*65*, *80*]  
  
# Processes/Pipes  
proc\_proxy = *None*  
proc\_cv = *None*  
proc\_ci = *None*  
proc\_GPIO = *None*  
proc\_APP = *None*  
pipes\_proxy = *None*  
pipes\_cv = *None*  
pipes\_ci = *None*  
pipes\_GPIO = *None*  
pipes\_APP = *None*  
t\_proxy = *None*  
t\_ci = *None*  
t\_cv = *None*  
t\_GPIO = *None*  
t\_APP = *None*  
q\_proxy = *None*  
q\_ci = *None*  
q\_cv = *None*  
q\_GPIO = *None*  
q\_APP = *None*  
  
# Other variables  
DEBUG = *True*  
is\_confirmed = *False*  
gpio\_high = *False*  
should\_terminate = *False*  
  
# Queueing  
def enqueue\_output(*out*, *queue*):  
 for line in iter(out.readline, b''):  
 queue.put(line)  
 out.close()  
  
def setup():  
 global proc\_proxy, proc\_cv, proc\_ci, proc\_GPIO, proc\_APP  
 global pipes\_proxy, pipes\_cv, pipes\_ci, pipes\_GPIO, pipes\_APP  
 global DEBUG  
 global t\_proxy, t\_cv, t\_ci, t\_GPIO, t\_APP  
 global q\_proxy, q\_cv, q\_ci, q\_GPIO, q\_APP  
  
 # for now assuming SITL device  
 if DEBUG:  
 '''  
 dronekit-sitl copter  
 '''  
 proc\_proxy = subprocess.Popen(["exec mavproxy.py --master=tcp:127.0.0.1:5760 --map"], *stdout*=subprocess.PIPE, *stdin*=subprocess.PIPE, *shell*=*True*)  
 else:  
 proc\_proxy = subprocess.Popen(["exec mavproxy.py --map"], *stdout*=subprocess.PIPE, *stdin*=subprocess.PIPE, *shell*=*True*)  
 pipes\_proxy = [proc\_proxy.stdin, proc\_proxy.stdout]  
 # queueing idea borrowed from <http://stackoverflow.com/questions/375427/non-blocking-read-on-a-subprocess-pipe-in-python> [21]  
 q\_proxy = Queue()  
 t\_proxy = Thread(*target*=enqueue\_output, *args*=(pipes\_proxy[*1*], q\_proxy))  
 t\_proxy.daemon = *True*  
 t\_proxy.start()  
  
 proc\_cv = subprocess.Popen(["exec bash"], *stdout*=subprocess.PIPE, *stdin*=subprocess.PIPE, *shell*=*True*)  
 pipes\_cv = [proc\_cv.stdin, proc\_cv.stdout]  
 q\_cv = Queue()  
 t\_cv = Thread(*target*=enqueue\_output, *args*=(pipes\_cv[*1*], q\_cv))  
 t\_cv.daemon = *True*  
 t\_cv.start()  
  
 proc\_ci = subprocess.Popen(["exec ../stabilize ./tmp\_capture.png"], *stdin*=subprocess.PIPE, *stdout*=subprocess.PIPE, *shell*=*True*)  
 pipes\_ci = [proc\_ci.stdin, proc\_ci.stdout]  
 q\_ci= Queue()  
 t\_ci = Thread(*target*=enqueue\_output, *args*=(pipes\_ci[*1*], q\_ci))  
 t\_ci.daemon = *True*  
 t\_ci.start()  
  
 proc\_GPIO = subprocess.Popen(["exec bash"], *stdin*=subprocess.PIPE, *stdout*=subprocess.PIPE, *shell*=*True*)  
 pipes\_GPIO = [proc\_GPIO.stdin, proc\_GPIO.stdout]  
 q\_GPIO = Queue()  
 t\_GPIO = Thread(*target*=enqueue\_output, *args*=(pipes\_GPIO[*1*], q\_GPIO))  
 t\_GPIO.daemon = *True*  
 t\_GPIO.start()  
  
 #if DEBUG:  
 # proc\_APP = subprocess.Popen(["exec bash"], stdin=subprocess.PIPE, stdout=subprocess.PIPE, shell=True)  
 #else:  
 proc\_APP = subprocess.Popen(["exec python ../gcm\_server.py"], *stdin*=subprocess.PIPE, *stdout*=subprocess.PIPE, *shell*=*True*)  
 pipes\_APP = [proc\_APP.stdin, proc\_APP.stdout]  
 q\_APP = Queue()  
 t\_APP = Thread(*target*=enqueue\_output, *args*=(pipes\_APP[*1*], q\_APP))  
 t\_APP.daemon = *True*  
 t\_APP.start()  
  
 return *True*  
  
# close pipes and kill processes  
def teardown():  
 global proc\_proxy, proc\_cv, proc\_ci, proc\_GPIO, proc\_APP  
 global pipes\_proxy, pipes\_cv, pipes\_ci, pipes\_GPIO, pipes\_APP  
 global t\_proxy, t\_cv, t\_ci, t\_GPIO, t\_APP  
  
 def closeup(*p*, *pipes*, *t*):  
 for pipe in pipes:  
 try:  
 pipe.close()  
 except:  
 pass  
 p.terminate()  
 try:  
 p.kill()  
 except Error:  
 pass  
  
 closeup(proc\_proxy, pipes\_proxy, t\_proxy)  
 closeup(proc\_cv, pipes\_cv, t\_cv)  
 closeup(proc\_ci, pipes\_ci, t\_ci)  
 closeup(proc\_GPIO, pipes\_GPIO, t\_GPIO)  
 closeup(proc\_APP, pipes\_APP, t\_APP)  
  
 return *True*  
  
# process image (remove borders, decode, or look for direction)  
def scan\_image(*mode*="template", *args*=*None*):  
 global pipes\_cv  
  
 if DEBUG:  
 return *None*  
 else:  
 if mode == "template":  
 pipes\_cv[*0*].write("../template ./tmp\_capture{0}.png {1} {2}\n").format(args["idx"], argx["lower"], args["upper"])  
 output = collect\_output("opencv", *keys*=["missing", "complete", "error", "invalid"], *debug*=*False*)  
 return output  
 elif mode == "decode":  
 pipes\_cv[*0*].write("../border N Y {0} {1} {2}\n".format(EMAIL\_ADDRESS, "./tmp\_capture{0}.png", "./tmp\_solution.png").format(args["idx"]))  
 output = collect\_output("opencv", *keys*=["invalid", "result"], *debug*=*False*)  
 return output  
 elif mode == "locate":  
 pipes\_cv[*0*].write("../locate ./border.png")  
 output = collect\_output("opencv", *keys*=["invalid", "LEFT", "RIGHT", "UP", "DOWN", "CENTER"], *debug*=*False*)  
 return output  
  
def collect\_output(*proc*, *attempts*=*30*, *keys*=["GPS\_RAW\_INT"], *flag*="or", *debug*=*False*):  
 # gather output for timeout seconds, examine  
 q = *None*  
 if ("mav" in proc) or ("proxy" in proc):  
 global q\_proxy  
 q = q\_proxy  
 elif "opencv" in proc:  
 global q\_cv  
 q = q\_cv  
 elif "capture" in proc:  
 global q\_ci  
 q = q\_ci  
 elif "GPIO" in proc:  
 global q\_GPIO  
 q = q\_GPIO  
 elif "APP" in proc:  
 global q\_APP  
 q = q\_APP  
  
 used\_attempts = *0*  
 while (used\_attempts < attempts):  
 try:  
 line = q.get\_nowait()  
 print 'Discovered: '+line  
 found = *False*  
 if flag == "and":  
 found = *True*  
 for key in keys:  
 if not (key in line):  
 found = *False*  
 break  
 else:  
 found = *False*  
 for key in keys:  
 if key in line:  
 found = *True*  
 break  
 if found:  
 return line  
 except:  
 pass  
 used\_attempts += *1*  
 return *None*  
  
# query for gps status on drone, return information decoded  
def get\_curr\_latlon():  
 global pipes\_proxy  
  
 output = *None*  
 while output is *None*:  
 pipes\_proxy[*0*].write("status GPS\_RAW\_INT\n")  
 output = collect\_output("proxy")  
  
 if output is *None*:  
 print "unable to retrieve gps corodinates..try again?"  
 else:  
 gps\_info = output.split("GPS\_RAW\_INT")[*1*]  
 gps\_info\_str = json.dumps(gps\_info)  
 gps\_info\_arr = (filter(*None*, re.split("[{, \!?:}\"\n]+", gps\_info\_str)))[:-*1*]  
 gps\_info\_dict = {}  
 for i in xrange(*0*, len(gps\_info\_arr), *2*):  
 gps\_info\_dict[gps\_info\_arr[i]] = gps\_info\_arr[i+*1*]  
  
 curr\_lat, curr\_lon = int(gps\_info\_dict['lat'])/LAT\_LON\_DIV, int(gps\_info\_dict['lon'])/LAT\_LON\_DIV  
 return (curr\_lat, curr\_lon)  
  
# issue GPS location for drone to travel to  
def go\_to(*lat*, *lon*, *fn*=*None*, *fn\_args* = *None*, *ret*=*False*, *curr\_lat*=*None*, *curr\_lon*=*None*):  
 global pipes\_proxy  
  
 pipes\_proxy[*0*].write("guided {0} {1} {2}\n".format(lat, lon, int(math.ceil(FLYING\_ALTITUDE/M\_FT\_CONV))))  
  
 output = *None*  
 if not (fn is *None*):  
 if not (fn\_args is *None*):  
 output = fn(fn\_args)  
 else:  
 output = fn()  
  
 if ret and (not (curr\_lat is *None*) and (not curr\_lon is *None*)):  
 time.sleep(*5*)  
 pipes\_proxy[*0*].write("guided {0} {1} {2}\n".format(curr\_lat, curr\_lon, int(math.ceil(FLYING\_ALTITUDE/M\_FT\_CONV))))  
  
 return output  
  
# set the drone in guided and arm  
def arm\_throttle():  
 global pipes\_proxy  
  
 pipes\_proxy[*0*].write("mode guided\n")  
 pipes\_proxy[*0*].write("arm throttle\n")  
  
# takeoff to flying altitude (beginning/threat avoidance)  
def takeoff(*arm*=*True*):  
 global pipes\_proxy  
  
 # optionally arm the throttle  
 if arm:  
 arm\_throttle()  
 time.sleep(*5*)  
  
 pipes\_proxy[*0*].write("takeoff {0}\n".format(FLYING\_ALTITUDE/M\_FT\_CONV))  
  
 print "completed takeoff commands"  
 time.sleep(*30*)  
  
# set the drone into landing mode (not RTL)  
def land():  
 global pipes\_proxy  
  
 pipes\_proxy[*0*].write("mode land\n")  
 time.sleep(*10*)  
  
 print "completed landing commands"  
  
# look for the pad's direction  
def initial\_search():  
 global pipes\_proxy  
  
 pipes\_proxy[*0*].write("mode guided\n")  
  
 # establish coordinates to evaluate  
 curr\_lat, curr\_lon = get\_curr\_latlon()  
 lat\_u = curr\_lat+(*50*/(LATITUDE\_CONV))  
 lat\_d = curr\_lat-(*50*/(LATITUDE\_CONV))  
 lon\_r = curr\_lon+(*50*/(LONGITUDE\_CONV))  
 lon\_l = curr\_lon-(*50*/(LONGITUDE\_CONV))  
 best\_dir = *None*  
  
 # chceck if the direction is useful  
 def move\_helper(*direction*):  
 best\_dir = *None*  
 for i in xrange(*50*):  
 output = scan\_image(*mode*="locate")  
 if not (output is *None*):  
 best\_dir = direction  
 time.sleep(*0.05*)  
 return best\_dir  
  
 # try all four directions in cross  
 output\_u = go\_to(lat\_u, curr\_lon, move\_helper, "up", *True*, curr\_lat, curr\_lon)  
 time.sleep(*10*)  
 output\_d = go\_to(lat\_d, curr\_lon, move\_helper, "down", *True*, curr\_lat, curr\_lon)  
 time.sleep(*10*)  
 output\_r = go\_to(curr\_lat, lon\_r, move\_helper, "right", *True*, curr\_lat, curr\_lon)  
 time.sleep(*10*)  
 output\_l = go\_to(curr\_lat, lon\_l, move\_helper, "left", *True*, curr\_lat, curr\_lon)  
 time.sleep(*10*)  
  
 if not (output\_u is *None*):  
 best\_dir = output\_u  
 elif not (output\_d is *None*):  
 best\_dir = output\_d  
 elif not (output\_r is *None*):  
 best\_dir = output\_r  
 elif not (output\_l is *None*):  
 best\_dir = output\_l  
  
 print "drone initial flight path completed ", best\_dir  
 return best\_dir  
  
def move\_towards\_landing(*direction*):  
 global pipes\_proxy, pipes\_cv  
  
 # establish directional lat/lon  
 curr\_lat, curr\_lon = get\_curr\_latlon()  
 lat\_u = curr\_lat+(*20*/(LATITUDE\_CONV))  
 lat\_d = curr\_lat-(*20*/(LATITUDE\_CONV))  
 lon\_r = curr\_lon+(*20*/(LONGITUDE\_CONV))  
 lon\_l = curr\_lon-(*20*/(LONGITUDE\_CONV))  
  
 # repositino the drone  
 if (direction == "LEFT"):  
 output = go\_to(curr\_lat, lon\_l)  
 time.sleep(*10*)  
 elif (direction == "RIGHT"):  
 output = go\_to(curr\_lat, lon\_r)  
 time.sleep(*10*)  
 elif (direction == "UP"):  
 output = go\_to(lat\_u, curr\_lon)  
 time.sleep(*10*)  
 elif (direction == "DOWN"):  
 output = go\_to(lat\_d, curr\_lon)  
 time.sleep(*10*)  
 elif (direction == "CENTER"):  
 print "already in the center of the pad"  
 return *1*  
  
 print "moved towards landing location ", direction  
 return *0*  
  
# GPIG pin handler, will just check if a 1 is output indicating threat detected (Thread)  
def check\_GPIO():  
 global pipes\_GPIO, gpio\_high  
  
 while (*1*):  
 value = collect\_output("GPIO", *keys*=["1"])  
 if not (value is *None*):  
 gpio\_high = *True*  
 should\_terminate = *True*  
 if should\_terminate:  
 break  
  
# App handler, tell app to send request, break upon confirmation (Thread)  
def check\_APP():  
 global pipes\_APP, is\_confirmed, should\_terminate  
  
 pipes\_APP[*0*].write("send\n")  
 while (*1*):  
 output = collect\_output("APP", *keys*=["confirm"])  
 if not (output is *None*):  
 print 'done'  
 is\_confirmed = *True*  
 should\_terminate = *True*  
 if should\_terminate:  
 break  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 with open("../email.txt") as email\_file:  
 for line in email\_file:  
 EMAIL\_ADDRESS = line  
 break  
  
 setup()  
 time.sleep(*20*)  
 takeoff()  
 best\_dir = initial\_search()  
 if best\_dir is *None*:  
 pass  
 else:  
 output = move\_to\_landing(best\_dir)  
 while (output != *1*):  
 best\_dir = scan\_image(*mode*="locate")  
 output = move\_to\_landing(best\_dir)  
 # try decoding  
 is\_landing = *False*  
 output = scan\_image(*mode*="template", *args*={"idx": *0*, "lower": OPTIONS[*0*], "upper": UPPER})  
 if ("complete" in output):  
 result = scan\_image(*mode*="decode", *args*={"idx": *0*}).split("result")[*1*]  
 if int(result) > THRESHOLD:  
 is\_landing = *True*  
 land()  
 if not is\_landing:  
 output = scan\_image(*mode*="template", *args*={"idx": *1*, "lower": OPTIONS[*1*], "upper": UPPER})  
 if ("complete" in output):  
 result = scan\_image(*mode*="decode", *args*={"idx": *1*}).split("result")[*1*]  
 if int(result) > THRESHOLD:  
 is\_landing = *True*  
 land()  
 if not is\_landing:  
 pass  
 # TODO abort msg  
 land()  
  
 # start the GPIO/APP threads  
 #thread\_app = Thread(target=check\_GPIO, args=[])  
 #thread\_app.daemon = True  
 #thread\_msg = Thread(target=check\_APP, args=[])  
 #thread\_msg.daemon = True  
 check\_APP()  
 gpio\_high = *False*  
 is\_confirmed = *False*  
  
 print 'reaching stopping point'  
  
 last\_checked = time.time()  
 while(*1*):  
 # keep throttle armed in case we need to take off quickly  
 if (abs(time.time() - last\_checked)) > *4.0*:  
 arm\_throttle()  
 if (gpio\_high):  
 takeoff(*arm*=*False*)  
 elif (is\_confirmed):  
 land()  
 # TODO shine\_gpio led?  
 print 'tearing down now'  
 teardown()

**verification\_testbench.py**

import subprocess  
import os  
import sys  
import time  
from threading import Thread  
from Queue import Queue, Empty  
  
UPPER = *255*  
OPTIONS = [*50*, *65*, *80*]  
  
proc\_cv = *None*  
pipes\_cv = *None*  
t\_cv = *None*  
q\_cv = *None*  
  
# Queueing  
def enqueue\_output(*out*, *queue*):  
 for line in iter(out.readline, b''):  
 queue.put(line)  
 out.close()  
  
# create the opencv shell to issue commands to  
def setup():  
 global proc\_cv, pipes\_cv, t\_cv, q\_cv  
  
 proc\_cv = subprocess.Popen(['bash'], *stdout*=subprocess.PIPE, *stdin*=subprocess.PIPE)  
 pipes\_cv = [proc\_cv.stdin, proc\_cv.stdout]  
  
 q\_cv = Queue()  
 t\_cv = Thread(*target*=enqueue\_output, *args*=(pipes\_cv[*1*], q\_cv))  
 t\_cv.daemon = *True*  
 t\_cv.start()  
 return *True*  
  
# close opencv pipes and exit the shell  
def teardown():  
 global proc\_cv, pipes\_cv, t\_cv, q\_cv  
  
 for pipe in pipes\_cv:  
 try:  
 pipe.close()  
 except:  
 pass  
  
 proc\_cv.kill()  
 return *True*  
  
# look through the output (non-blocking using queue)  
def collect\_output(*attempts*=*30*, *keys*=[], *flag*="or", *debug*=*False*):  
 global q\_cv  
  
 q = q\_cv  
 used\_attempts = *0*  
 while (used\_attempts < attempts):  
 try:  
 line = q.get\_nowait()  
 if debug:  
 print 'Discovered: '+line  
 found = *False*  
 if flag == "and":  
 found = *True*  
 for key in keys:  
 if not (key in line):  
 found = *False*  
 break  
 else:  
 found = *False*  
 for key in keys:  
 if key in line:  
 found = *True*  
 break  
 if found:  
 return line  
 except Exception as e:  
 pass  
 return *None*  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 setup()  
 testfile = "./tests.txt"  
  
 # if testfile specified, override default  
 if (len(sys.argv) > *1*):  
 testfile = sys.argv[*1*]  
  
 tmp\_dest = "./tmp\_out"  
  
 with open(testfile, 'r') as input\_file:  
 successes = *0*  
 failures = *0*  
 threshold = -*1.0*  
 firstLine = *True*  
 test\_idx = *0*  
 for line in input\_file:  
 # read in threshold for success values  
 if firstLine:  
 threshold = float(line)  
 firstLine = *False*  
 else:  
 # run test case  
 print "Running Test {0}".format(test\_idx)  
 (input\_image, email\_addr, should\_normalize) = line.replace("\n", "").split("|")  
 input\_image = input\_image.lstrip().rstrip()  
 email\_addr = email\_addr.lstrip().rstrip()  
 should\_normalize = int(should\_normalize)  
 print "Beginning Phase 0"  
 # start clock  
 t0 = time.time()  
 failed = [*True* for x in xrange(len(OPTIONS))]  
 highest\_acc = -*1*  
  
 # try removing borders and rotating (multiple options for threshold for outliers)  
 for opt\_idx in xrange(len(OPTIONS)):  
 pipes\_cv[*0*].write("./template {0} {1} {2} {3}\n".format(input\_image, tmp\_dest+str(opt\_idx)+".png", OPTIONS[opt\_idx], UPPER))  
 phase0\_out = collect\_output(*attempts*=*100*, *keys*=["missing", "invalid", "completed", "failed"], *flag*="or", *debug*=*False*)  
 if (phase0\_out is *None*) or ("missing" in phase0\_out) or ("invalid" in phase0\_out) or ("failed" in phase0\_out):  
 continue  
 else:  
 failed[opt\_idx] = *False*  
 if should\_normalize:  
 pipes\_cv[*0*].write("./normalize {0} {1}\n".format(tmp\_dest+str(opt\_idx)+".png", tmp\_dest+str(opt\_idx)+".png"))  
 # stop clock  
 print "Phase 0 Time: {0}".format(time.time()-t0)  
 # start secondary clock  
 t1 = time.time()  
 for opt\_idx in xrange(len(OPTIONS)):  
 if (failed[opt\_idx] is *False*):  
 # try decoding  
 pipes\_cv[*0*].write("./border N Y {0} {1} {2}\n".format(email\_addr, input\_image, tmp\_dest+str(opt\_idx)+".png"))  
 phase1\_out = collect\_output(["invalid", "result"], "or")  
 # accuracy of removing borders with this threshold  
 if "invalid" in phase1\_out:  
 continue  
 result = phase1\_out.split("result")[*1*]  
 os.remove(tmp\_dest+str(opt\_idx)+".png")  
 observed = float(result)  
  
 if (observed > highest\_acc):  
 highest\_acc = observed  
 print highest\_acc  
 # stop clock  
 print "Phase 1 Time: {0}".format(time.time()-t1)  
 print "Total Time: {0}".format(time.time()-t0)  
  
 # if able to decode with high enough accruacy this is success  
 if (highest\_acc > threshold):  
 successes += *1*  
 else:  
 failures += *1*  
 test\_idx += *1*  
 print "Correct: {0}, Failures: {1}, Accuracy: {2}".format(successes, failures, (float(successes)/(successes + failures)))  
  
 teardown()

**live\_verification\_locate.py**

import subprocess  
import sys  
from threading import Thread  
from Queue import Queue, Empty  
from time import sleep  
import cv2  
  
proc\_cv = *None*  
proc\_ci = *None*  
pipes\_cv = *None*  
pipes\_ci = *None*  
t\_cv = *None*  
q\_cv = *None*  
  
# Queueing  
def enqueue\_output(*out*, *queue*):  
 for line in iter(out.readline, b''):  
 queue.put(line)  
 out.close()  
  
# create the opencv shell to issue commands to  
def setup():  
 global proc\_cv, proc\_ci  
 global pipes\_cv, proc\_ci  
 global t\_cv, q\_cv  
  
 proc\_cv = subprocess.Popen(['bash'], *stdout*=subprocess.PIPE, *stdin*=subprocess.PIPE)  
 pipes\_cv = [proc\_cv.stdin, proc\_cv.stdout]  
  
 proc\_ci = subprocess.Popen(["exec ./stabilize ./test\_capture.png"], *stdin*=subprocess.PIPE, *stdout*=subprocess.PIPE, *shell*=*True*)  
  
 q\_cv = Queue()  
 t\_cv = Thread(*target*=enqueue\_output, *args*=(pipes\_cv[*1*], q\_cv))  
 t\_cv.daemon = *True*  
 t\_cv.start()  
 return *True*  
  
# close opencv pipes and exit the shell  
def teardown():  
 global proc\_cv, proc\_camera  
 global pipes\_cv, pipes\_camera  
 global t\_cv, q\_cv  
  
 for pipe in pipes\_cv:  
 try:  
 pipe.close()  
 except:  
 pass  
 for pipe in pipes\_camera:  
 try:  
 pipe.close()  
 except:  
 pass  
  
 proc\_cv.kill()  
 proc\_camera.kill()  
 return *True*  
  
# look through the output (non-blocking using queue)  
def collect\_output(*attempts*=*30*, *keys*=[], *flag*="or", *debug*=*False*):  
 global q\_cv  
  
 q = q\_cv  
 used\_attempts = *0*  
 while (used\_attempts < attempts):  
 try:  
 line = q.get\_nowait()  
 if debug:  
 print 'Discovered: '+line  
 found = *False*  
 if flag == "and":  
 found = *True*  
 for key in keys:  
 if not (key in line):  
 found = *False*  
 break  
 else:  
 found = *False*  
 for key in keys:  
 if key in line:  
 found = *True*  
 break  
 if found:  
 return line  
 except Exception as e:  
 pass  
 return *None*  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 setup()  
 input\_image = "./test\_capture.png"  
 test\_idx = *0*  
  
 sleep(*2*)  
 while(*1*):  
 # run test case  
 print "Running Test {0}".format(test\_idx)  
 print "Beginning Phase 0"  
  
 # try removing borders and rotating (multiple options for threshold for outliers)  
 pipes\_cv[*0*].write("./locate {0}\n".format(input\_image))  
 inp = cv2.imread(input\_image)  
 phase0\_out = collect\_output(*attempts*=*100*, *keys*=["invalid", "LEFT", "RIGHT", "UP", "DOWN", "CENTER", "failed"], *flag*="or", *debug*=*False*)  
 if (phase0\_out is *None*) or ("invalid" in phase0\_out) or ("failed" in phase0\_out):  
 continue  
 else:  
 print phase0\_out  
 try:  
 cv2.imshow("Captured Image", inp)  
 cv2.waitKey(*0*)  
 cv2.destroyAllWindows()  
 except:  
 print "skipping"  
  
 sleep(*2*)  
  
 test\_idx += *1*  
 teardown()

**live\_verification\_decode.py**

import subprocess  
import os  
import sys  
from threading import Thread  
from Queue import Queue, Empty  
from time import sleep  
import numpy as np  
import cv2  
import time  
  
UPPER = *255*  
OPTIONS = [*50*, *65*] #80]  
  
proc\_cv = *None*  
proc\_ci = *None*  
pipes\_cv = *None*  
pipes\_ci = *None*  
t\_cv = *None*  
q\_cv = *None*  
  
# Queueing  
def enqueue\_output(*out*, *queue*):  
 for line in iter(out.readline, b''):  
 queue.put(line)  
 out.close()  
  
# create the opencv shell to issue commands to  
def setup():  
 global proc\_cv, proc\_ci  
 global pipes\_cv, proc\_ci  
 global t\_cv, q\_cv  
  
 proc\_cv = subprocess.Popen(['bash'], *stdout*=subprocess.PIPE, *stdin*=subprocess.PIPE)  
 pipes\_cv = [proc\_cv.stdin, proc\_cv.stdout]  
  
 proc\_ci = subprocess.Popen(["exec ./stabilize ./test\_capture.png"], *stdin*=subprocess.PIPE, *stdout*=subprocess.PIPE, *shell*=*True*)  
  
 q\_cv = Queue()  
 t\_cv = Thread(*target*=enqueue\_output, *args*=(pipes\_cv[*1*], q\_cv))  
 t\_cv.daemon = *True*  
 t\_cv.start()  
 return *True*  
  
# close opencv pipes and exit the shell  
def teardown():  
 global proc\_cv, proc\_camera  
 global pipes\_cv, pipes\_camera  
 global t\_cv, q\_cv  
  
 for pipe in pipes\_cv:  
 try:  
 pipe.close()  
 except:  
 pass  
 for pipe in pipes\_camera:  
 try:  
 pipe.close()  
 except:  
 pass  
  
 proc\_cv.kill()  
 proc\_camera.kill()  
 return *True*  
  
# look through the output (non-blocking using queue)  
def collect\_output(*attempts*=*30*, *keys*=[], *flag*="or", *debug*=*False*):  
 global q\_cv  
  
 q = q\_cv  
 used\_attempts = *0*  
 while (used\_attempts < attempts):  
 try:  
 line = q.get\_nowait()  
 if debug:  
 print 'Discovered: '+line  
 found = *False*  
 if flag == "and":  
 found = *True*  
 for key in keys:  
 if not (key in line):  
 found = *False*  
 break  
 else:  
 found = *False*  
 for key in keys:  
 if key in line:  
 found = *True*  
 break  
 if found:  
 return line  
 except Exception as e:  
 pass  
 return *None*  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 setup()  
 threshold = *.60*  
 email\_addr = "hoaglan2@illinois.edu"  
 tmp\_dest1 = "./tmp\_out1-"  
 tmp\_dest2 = "./tmp\_out2-"  
 input\_image = "./test\_capture.png"  
 successes = *0*  
 failures = *0*  
 test\_idx = *0*  
  
 if len(sys.argv) < *3*:  
 print "using default success threshold of .60 and email of hoaglan2@illinois.edu"  
  
 else:  
 threshold = sys.argv[*1*]  
 email\_addr = sys.argv[*2*]  
  
 sleep(*2*)  
 while(*1*):  
 # run test case  
 print "Running Test {0}".format(test\_idx)  
 print "Beginning Phase 0"  
 # start clock  
 start\_time = time.time()  
 failed = [*True* for x in xrange(len(OPTIONS))]  
 highest\_acc = -*1*  
  
 # try removing borders and rotating (multiple options for threshold for outliers)  
 for opt\_idx in xrange(len(OPTIONS)):  
 pipes\_cv[*0*].write("./template {0} {1} {2} {3}\n".format(input\_image, tmp\_dest1+str(opt\_idx)+".png", OPTIONS[opt\_idx], UPPER))  
 phase0\_out = collect\_output(*attempts*=*100*, *keys*=["missing", "invalid", "completed", "failed"], *flag*="or", *debug*=*False*)  
 if (phase0\_out is *None*) or ("missing" in phase0\_out) or ("invalid" in phase0\_out) or ("failed" in phase0\_out):  
 continue  
 else:  
 failed[opt\_idx] = *False*  
 # perform normalization  
 pipes\_cv[*0*].write("./normalize {0} {1}\n".format(tmp\_dest1+str(opt\_idx)+".png", tmp\_dest2+str(opt\_idx)+".png"))  
 phase1\_out = collect\_output(*attempts*=*100*, *keys*=["invalid", "success"], *flag*="or", *debug*=*False*)  
 try:  
 original = cv2.imread(input\_image)  
 except:  
 continue  
 end\_time = time.time()  
 print "Time to complete {0}: {1} seconds".format(test\_idx, (end\_time - start\_time))  
 cropped = []  
 normalized = []  
 for opt\_idx in xrange(len(OPTIONS)):  
 if (failed[opt\_idx] is *False*):  
 try:  
 n\_cropped = cv2.imread(tmp\_dest1+str(opt\_idx)+".png")  
 n\_normalized = cv2.imread(tmp\_dest2+str(opt\_idx)+".png")  
 n\_cropped = cv2.resize(n\_cropped, (*200*, *200*))  
 n\_normalized = cv2.resize(n\_normalized, (*200*, *200*))  
 if n\_normalized is *None*:  
 print tmp\_dest2+str(opt\_idx)+".png"  
 cropped.append(n\_cropped)  
 normalized.append(n\_normalized)  
 except:  
 pass  
 if len(cropped) > *1*:  
 cropped\_display = np.hstack(tuple(cropped))  
 print 'klo'  
 normalized\_display = np.hstack(tuple(normalized))  
 cv2.imshow("Original Image", original)  
 cv2.imshow("Cropped Images", cropped\_display)  
 cv2.imshow("Normalized Images", normalized\_display)  
 cv2.waitKey(*0*)  
 cv2.destroyAllWindows()  
 else:  
 sleep(*2*)  
  
 for opt\_idx in xrange(len(OPTIONS)):  
 if (failed[opt\_idx] is *False*):  
 # try decoding  
 pipes\_cv[*0*].write("./border N Y {0} {1} {2}\n".format(email\_addr, input\_image, tmp\_dest2+str(opt\_idx)+".png"))  
 phase1\_out = collect\_output(["invalid", "result"], "or")  
 # accuracy of removing borders with this threshold  
 if "invalid" in phase1\_out:  
 continue  
 os.remove(tmp\_dest1+str(opt\_idx)+".png")  
 os.remove(tmp\_dest2+str(opt\_idx)+".png")  
 result = phase1\_out.split("result")[*1*]  
 observed = float(result)  
  
 if (observed > highest\_acc):  
 highest\_acc = observed  
 print highest\_acc  
  
 # if able to decode with high enough accruacy this is success  
 if (highest\_acc > threshold):  
 successes += *1*  
 else:  
 failures += *1*  
 test\_idx += *1*  
 print "Correct: {0}, Failures: {1}, Accuracy: {2}".format(successes, failures, (float(successes)/(successes + failures)))  
  
 teardown()

**Gcm\_server.py**

#Source code taken and modified from [15] <https://gist.github.com/vietkute02/9680901>

#!/usr/bin/python  
import sys, json, xmpp, random, string  
from subprocess import PIPE, Popen  
from threading import Thread  
from Queue import Queue, Empty  
import datetime  
  
SERVER = 'gcm.googleapis.com'  
PORT = *5235*  
USERNAME = "632877030162"  
PASSWORD = "AIzaSyBnar8mPqv6CmZ8gqOQBtUOOyq28nw2cwY"  
REGISTRATION\_ID = "d3Z--G\_Ilq0:APA91bE5qwn034sII-Sm5IkZdsUWfNfBVQVjbcNKDDGcicjI9G7SCWmeuri7o\_D99NlymrbLN-JlbyKcSrUfR2iW\_-1jShrIILnX\_2e0QBsrgBPHb0\_\_pALCc85oHhX82U7iuUv0kDKi"  
#REGISTRATION\_ID = "fkDwv4iOT74:APA91bFSC8ANDGgnimDl6DpWXV4og8cXl54Oj3xluTPmgs8dOI43iU7sCwEMmOljNqoILsHmD3VsZ3WGd-iuTa0H-ab0oWZOJTbqztysUzURwvpCdM02ww7BdcWqsS4fHhPAPG\_W9leB"  
ON\_POSIX = 'posix' in sys.builtin\_module\_names  
  
unacked\_messages\_quota = *100*  
send\_queue = []  
  
#Determine current UTC time  
def UtcNow():  
 now = datetime.datetime.utcnow()  
 return (now - datetime.datetime(*1970*, *1*, *1*)).total\_seconds()  
  
# Return a random alphanumerical id  
def random\_id():  
 rid = ''  
 for x in range(*8*): rid += random.choice(string.ascii\_letters + string.digits)  
 return rid  
  
#Determine communication time if received message and print "confirm" message  
def message\_callback(*session*, *message*):  
 server\_time = UtcNow()  
 gcm = message.getTags('gcm')  
 if gcm:  
 gcm\_json = gcm[*0*].getData()  
 msg = json.loads(gcm\_json)  
 if("message\_type" not in msg):  
 app\_time = int(msg["data"]["time\_test"])  
 print("Commnication took : " + str((server\_time-app\_time)) + " seconds")  
 print("confirm")  
 print('\n')  
  
#Package data into json to send to app  
def send(*json\_dict*):  
 template = ("<message><gcm xmlns='google:mobile:data'>{1}</gcm></message>")  
 client.send(xmpp.protocol.Message(  
 *node*=template.format(client.Bind.bound[*0*], json.dumps(json\_dict))))  
  
#Process messages in queue  
def flush\_queued\_messages():  
 global unacked\_messages\_quota  
 while len(send\_queue) and unacked\_messages\_quota > *0*:  
 send(send\_queue.pop(*0*))  
 unacked\_messages\_quota -= *1*  
  
#client = xmpp.Client('gcm.googleapis.com', debug=['socket'])  
  
#Set up xmpp server  
client = xmpp.Client('gcm.googleapis.com', *debug*=[])  
client.connect(*server*=(SERVER,PORT), *secure*=*1*, *use\_srv*=*False*)  
auth = client.auth(USERNAME, PASSWORD)  
if not auth:  
 print 'Authentication failed!'  
 sys.exit(*1*)  
  
#Map server to message call back function  
client.RegisterHandler('message', message\_callback)  
  
#Enqueue sends to queue  
def enqueue\_output(*out*, *queue*):  
 for line in iter(out.readline, b''):  
 queue.put(line)  
 out.close()  
  
q = Queue()  
t = Thread(*target*=enqueue\_output, *args*=(sys.stdin, q))  
t.daemon = *True* # thread dies with the program  
t.start()  
  
  
#Contiously runs server, sending messages when "send" command is inputed by user  
while *True*:  
 client.Process(*1*)  
 flush\_queued\_messages()  
  
 # read line without blocking  
 try: line = q.get\_nowait() # or q.get(timeout=.1)  
 except Empty:  
 continue  
  
 if("send" in line):  
 send\_queue.append({'to': REGISTRATION\_ID,  
 'message\_id': 'reg\_id',  
 'data': {'message\_destination': 'RegId',  
 'message\_id': random\_id(),  
 'time\_test': str(UtcNow())}})

**gpio.c**

#**include** <stdio.h>

#**include** <stdlib.h>

#**include** <stdint.h>

#**include** <pthread.h>

#**include** <unistd.h>

#**include** <string.h>

#**include** <time.h>

#**include** <wiringPi.h>

#**include** <wiringPiI2C.h>

#**include** <wiringSerial.h>

#**include** <lcd.h>

#**include** <iostream>

#**include** <thread>

void **check**(const int doppler, const int pir1, const int pir2);

bool **analogCheck**(const int doppler);

// Pin number declarations for ODROID-C1+

const int pirLeftFront = *2*;

const int pirLeftBack = *3*;

const int pirRightFront = *4*;

const int pirRightBack = *5*;

const int outputVar = *27*;

#**define** **PORT\_ADC1** 1

#**define** **PORT\_ADC0** 0

bool outputVal = *false*;

int **main**(void)

{

wiringPiSetup();

pinMode(pirLeftFront, INPUT);

pinMode(pirLeftBack, INPUT);

pinMode(pirRightFront, INPUT);

pinMode(pirRightBack, INPUT);

pinMode(outputVar, OUTPUT);

digitalWrite(outputVar, LOW);

digitalWrite(pirLeftFront, LOW);

digitalWrite(pirLeftBack, LOW);

digitalWrite(pirRightFront, LOW);

digitalWrite(pirRightBack, LOW);

bool pLF, pLB, pRF, pRB, dL, dR;

while(1)

{

pLF = digitalRead(pirLeftFront);

pLB = digitalRead(pirLeftBack);

pRF = digitalRead(pirRightFront);

pRB = digitalRead(pirRightBack);

dL = analogCheck(PORT\_ADC1);

dR = analogCheck(PORT\_ADC0);

std::cout << dR << std::endl;

std::cout << pLF << " " << pLB << " " << pRF << " " << pRB << " " << dL << " " << dR << '\n';

// if(pLF == true || pLB == true || dL == true)

// {

// std::cout << "potential threat, left" << '\n';

// check(PORT\_ADC1, pirLeftFront, pirLeftBack);

// }

if(pRF == true || pRB == true || dR == true)

{

std::cout << "potential threat, right" << '\n';

check(PORT\_ADC0, pirRightFront,pirRightBack);

}

if(outputVal == true)

{

std::cout << "entering infinite loop" << std::endl;

while(1){

std::cout << "in the loop" << std::endl;

digitalWrite(outputVar, HIGH);

delay(500);

digitalWrite(outputVar, LOW);

delay(500);

digitalWrite(outputVar, HIGH);

delay(500);

digitalWrite(outputVar, LOW);

delay(500);

digitalWrite(outputVar, HIGH);

digitalWrite(outputVar, LOW);

}//break;

}

}

}

void **check**(const int doppler, const int pir1, const int pir2)

{

bool loopDecide;

if(**analogCheck**(doppler) == true)

loopDecide = false;

else

loopDecide = true;

time\_t start, end;

time(&start);

end = start;

if(loopDecide == false)

{

while(**difftime**(end, start) < 2)

{

if(**digitalRead**(pir1) == HIGH || digitalRead(pir2) == HIGH)

{

outputVal = true;

break;

}

time(&end);

}

}

else

{

while(**difftime**(end,start) < 2)

{

if(**analogRead**(doppler) == true)

{

outputVal = true;

break;

}

time(&end);

}

}

}

bool **analogCheck**(const int doppler)

{

int adcVal;

adcVal = analogRead(doppler);

//**printf**("%d\n", adcVal);

std::cout << adcVal\*1.8/1024.0 << '\n';

if(adcVal\*1.8/1024.0 > 1.6){return true;}

return false;

}