```
// opencv stuff
#include <unistd.h> // serial comms
#include <lccv.hpp>
#include <opencv2/opencv.hpp>
#include <stdio.h>

// file writing
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

using namespace cv;
using namespace cv::ml;
using namespace std;

#define BAUDRATE B115200
#define MODEMDEVICE "/dev/ttyAMA0"

//declare files
FileStorage read_data, read_labels;
Mat training_data, training_labels;

int uart0_filestream = -1;

void init(){
	uart0_filestream = open(MODEMDEVICE, O_RDWR | O_NOCTTY | O_NDELAY);
	//Open in non blocking read/write mode
	if (uart0_filestream == -1)
	{
		//ERROR - CAN'T OPEN SERIAL PORT
		printf("Error - Unable to open UART.  Ensure it is not in use by another
application\n");
		exit(-1);
	}
}

void send(char side,char victim){
	//----- TX BYTES -----
	char tx_buffer[20];
	sprintf(tx_buffer,"%c%c",side,victim);
	if (uart0_filestream != -1)
	{
		int err  = 0;
```

```
                err = write(uart0_filestream, &tx_buffer[0], strlen(tx_buffer));
//Filestream, bytes to write, number of bytes to write
                if (err < 0)
                {
                        printf("UART TX error\n");
                }else{
                        //printf("UART TX ok\n");
                }
                cout<<side<<" saw "<<victim<<endl;
        }
}

//compare contour function
bool contour_compare(const vector<Point> &a, const vector<Point> &b) {
        return contourArea(a) < contourArea(b);
}


int main(){
        // start the cams
        Mat img1, img2;
        lccv::PiCamera cam1;
        lccv::PiCamera cam2;

        //cam1
        cam1.options->camera = 0;
        cam1.options->video_width=320;//640;
        cam1.options->video_height=240;//480;
        cam1.options->framerate=30;
        cam1.options->verbose=true;
        cam1.startVideo();

        //cam2
        cam2.options->camera = 1;
        cam2.options->video_width=320;//640;
        cam2.options->video_height=240;//480;
        cam2.options->framerate=30;
        cam2.options->verbose=true;
        cam2.startVideo();

        //just make a window for it
        namedWindow("hello",cv::WINDOW_NORMAL);

        // open and READ in mat objects
```

```
	//opening the files w/ the data to put the data in the mats
	read_data = FileStorage("data.txt", FileStorage::READ);
	read_labels = FileStorage("labels.txt", FileStorage::READ);

	read_data["DATA"] >> training_data;
	read_labels["LABELS"] >> training_labels;

	//just converting the data into a float cuz knn needs it for some reason
	training_data.convertTo(training_data, CV_32F);
	training_labels.convertTo(training_labels, CV_32F);

	//print out nrows and ncols of the data
	cout << "training data size:   cols " << training_data.cols <<  "rows " <<
training_data.rows << endl;
	cout << "training labels size: cols " << training_labels.cols << "rows " <<
training_labels.rows << endl;

	//creating a pointer
	Ptr<KNearest> knn = KNearest::create();
   knn->train(training_data, ROW_SAMPLE, training_labels);

   while(true){
		if(!cam1.getVideoFrame(img1,99999999)){
			std::cout << "Camera error1" << std::endl;
			break;
		}
		Mat og_img1 = img1.clone();

		if(!cam2.getVideoFrame(img2,99999999)){
			std::cout << "Camera error2" << std::endl;
			break;
		}
		Mat og_img2 = img2.clone();

		//just convert the image to greyscale and threshhold it
		cvtColor(img1, img1, COLOR_BGR2GRAY);
		GaussianBlur(img1, img1, Size(9,9), 0, 0);
		threshold(img1, img1, 80, 255, THRESH_BINARY_INV+THRESH_OTSU);
		imshow("thresh", img1);

		cvtColor(img2, img2, COLOR_BGR2GRAY);
		GaussianBlur(img2, img2, Size(9,9), 0, 0);
		threshold(img2, img2, 80, 255, THRESH_BINARY_INV+THRESH_OTSU);
		imshow("thresh", img2);
```

```
		//just finding contours in the image
		vector<vector<Point>> contours1;
		findContours(img1, contours1, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
		sort(contours1.begin(), contours1.end(), contour_compare);

		vector<vector<Point>> contours2;
		findContours(img2, contours2, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
		sort(contours2.begin(), contours2.end(), contour_compare);
		//run through all contours
		for(auto& cnt : contours1){
			Rect br;
			//cout << "in countours" << endl;
			double a = contourArea(cnt);
			// filter with area
			if(!(a < 2000 && a > 300)){
				drawContours(og_img1, vector<vector<Point>>(1,cnt), -1,
Scalar(0, 0, 255), 1);
				//cout << "filtered out cuz of area thingy" << endl;
				continue;
			}

			//make a bounding rect
			br = boundingRect(cnt);
			float ar = (float)br.height/br.width;

			//printf("ar: %.2f\n", ar);

			if(ar > 1.5 || ar < 0.5){
			drawContours(og_img1, vector<vector<Point>>(1,cnt), -1, Scalar(0, 255,
255), 1);
			continue;
			}

			// should be a letter by this point
			rectangle(og_img1, br, Scalar(0, 255, 0), 1);
			imshow("ogyk", og_img1);

			Mat blank = Mat::zeros(img1.size(), CV_8UC1);
			Mat letter;
			cout << "passes making rectangle" << endl;
			// redraw contour on a blank image
			drawContours(blank, vector<vector<Point>>(1, cnt), -1, 255, -1);
			imshow("blank", blank);
```

```
                        cout << "i got to the slicing" << endl;
                        cout << br.x << " " << br.y << " " << br.width << " " << br.height << endl;
                        // slice the letter so the rect is a little bigger
                        if(br.x < 10 || br.y < 10 || br.x + br.width + 10 > 319 || br.height + br.y + 10 >
239){
                                continue;
                        }
                        letter = blank(Rect(br.x - 10, br.y - 10, br.width + 20, br.height + 20));
                        imshow("letter", letter);
                        cout << "i passed the slicing" << endl;

                        RotatedRect rr = minAreaRect(cnt);
                        printf("ANGLE: %.2f\n", rr.angle);

                        cout << "center: " << rr.center << endl;

                        // warp affine stuff - make it some 90 degree angle
                        Mat rot_matrix = getRotationMatrix2D(Point2f(letter.cols/2.0F,
letter.rows/2.0F), rr.angle, 1.0);
                        Mat rotated_letter(Size(letter.size().height, letter.size().width),
letter.type());
                        warpAffine(letter, rotated_letter, rot_matrix, letter.size());
                        vector<Mat> fourWay;
                        fourWay.push_back(rotated_letter);
                        imshow("first", fourWay[0]);
                        for(int i = 1; i < 4; i++){
                                rotated_letter = fourWay[i-1].clone();
                                rot_matrix = getRotationMatrix2D(Point2f(rotated_letter.cols/2.0F,
rotated_letter.rows/2.0F), 90, 1.0);
                                warpAffine(rotated_letter, rotated_letter, rot_matrix,
rotated_letter.size());
                                //imshow("rotation" + i, rotated_letter);
                                fourWay.push_back(rotated_letter);
                                //imshow("fourWay" + i, fourWay[i]);
                        }
                        cout << "length" << fourWay.size();
                        for(int i = 0; i < 4; i++)
                                imshow("fourWay" + i, fourWay[i]);
                        //get letter contours and find the area which encapsulates the letter
                        vector<Mat> fourRes;

                        for(int i = 0; i < 4; i++){
                                vector<vector<Point>> letter_contours;
```

```
                findContours(fourWay[i], letter_contours, RETR_EXTERNAL,
CHAIN_APPROX_SIMPLE);
                //drawContours(fourWay[i], letter_contours, 0, 8);
                //imshow("fourWay" + i, fourWay[i]);
                sort(letter_contours.begin(), letter_contours.end(),
contour_compare);
                Rect final_letter = boundingRect(letter_contours[0]);
                Mat res = fourWay[i](final_letter);
                fourRes.push_back(res);
                //imshow("fourRes" + i, res);
        }
        //Mat res = rotated_letter.clone();

        //resize the letter and flatten it
        vector<Mat> flat;
        for(int i = 0; i < 4; i++){
                resize(fourRes[i], fourRes[i], Size(20,20));
                flat.push_back(fourRes[i].reshape(1,1));//res.total());
        }

        vector<Mat> ret(4);
        vector<vector<float>> neighbors(4);
        vector<vector<float>> distance(4);
        // pass to knn and get classifiction
        for(int i = 0; i < 4; i++){
                flat[i].convertTo(flat[i], CV_32F);
                //cout << "flat:  " << flat[i].size() << endl;
                //cout << "train: " << training_data.size() << endl;
                //cout << "what" << endl;
                knn->findNearest(flat[i], 3, ret[i], neighbors[i], distance[i]);
                cout << "broken";
                //cout << endl << "holy crap ret: " << ret[i] << "  neighbors: " <<
neighbors[i][0] << "  distance: " << distance[i][0] << endl << endl;
        }
        int xyz = 0;
        float minDist = 2000000000;
        for(int i = 0; i < 4; i++){
                float curr = distance[i][0];
                if(curr < minDist){
                        minDist = curr;
                        xyz = i;
                }
        }
```

```
                        char value = ret[xyz].at<float>(0,0);
                        if(minDist < 4000000){
                                printf("\n\nGUESSED A %c - %f - cam: %d on the LEFT\n\n",
value, minDist, 1);
                                send('L', value);
                        }
                        else
                                cout << "\nToo far from a letter - " << 1 << endl;
                        //printf("Closest distance %d\n", (int)distance[xyz][0]);
                        cout << "closest" << (int)minDist << endl;
                        /*char value = ret.at<float>(0,0);
                        if(distance[0] < 4000000)
                                printf("GUESSED A %c\n", value);
                        else
                                cout << "Too far from a letter" << endl;
                        printf("Closest distance %d\n", (int)distance[0]);*/
                }

                for(auto& cnt : contours2){
                        Rect br;
                        //cout << "in countours" << endl;
                        double a = contourArea(cnt);
                        // filter with area
                        if(!(a < 2000 && a > 300)){
                                drawContours(og_img2, vector<vector<Point>>(1,cnt), -1,
Scalar(0, 0, 255), 1);
                                //cout << "filtered out cuz of area thingy" << endl;
                                continue;
                        }

                        //make a bounding rect
                        br = boundingRect(cnt);
                        float ar = (float)br.height/br.width;

                        //printf("ar: %.2f\n", ar);

                        if(ar > 1.5 || ar < 0.5){
                        drawContours(og_img2, vector<vector<Point>>(1,cnt), -1, Scalar(0, 255,
255), 1);
                        continue;
                        }

                        // should be a letter by this point
                        rectangle(og_img2, br, Scalar(0, 255, 0), 1);
```

```
                        imshow("ogyk", og_img2);

                        Mat blank = Mat::zeros(img2.size(), CV_8UC1);
                        Mat letter;
                        cout << "passes making rectangle" << endl;
                        // redraw contour on a blank image
                        drawContours(blank, vector<vector<Point>>(1, cnt), -1, 255, -1);
                        imshow("blank", blank);

                        cout << "i got to the slicing" << endl;
                        cout << br.x << " " << br.y << " " << br.width << " " << br.height << endl;
                        // slice the letter so the rect is a little bigger
                        if(br.x < 10 || br.y < 10 || br.x + br.width + 10 > 319 || br.height + br.y + 10 >
239){
                                continue;
                        }
                        letter = blank(Rect(br.x - 10, br.y - 10, br.width + 20, br.height + 20));
                        imshow("letter", letter);
                        cout << "i passed the slicing" << endl;

                        RotatedRect rr = minAreaRect(cnt);
                        printf("ANGLE: %.2f\n", rr.angle);

                        cout << "center: " << rr.center << endl;

                        // warp affine stuff - make it some 90 degree angle
                        Mat rot_matrix = getRotationMatrix2D(Point2f(letter.cols/2.0F,
letter.rows/2.0F), rr.angle, 1.0);
                        Mat rotated_letter(Size(letter.size().height, letter.size().width),
letter.type());
                        warpAffine(letter, rotated_letter, rot_matrix, letter.size());
                        vector<Mat> fourWay;
                        fourWay.push_back(rotated_letter);
                        imshow("first", fourWay[0]);
                        for(int i = 1; i < 4; i++){
                                rotated_letter = fourWay[i-1].clone();
                                rot_matrix = getRotationMatrix2D(Point2f(rotated_letter.cols/2.0F,
rotated_letter.rows/2.0F), 90, 1.0);
                                warpAffine(rotated_letter, rotated_letter, rot_matrix,
rotated_letter.size());
                                //imshow("rotation" + i, rotated_letter);
                                fourWay.push_back(rotated_letter);
                                //imshow("fourWay" + i, fourWay[i]);
                        }
```

```
                        cout << "length" << fourWay.size();
                        for(int i = 0; i < 4; i++)
                                imshow("fourWay" + i, fourWay[i]);
                        //get letter contours and find the area which encapsulates the letter
                        vector<Mat> fourRes;

                        for(int i = 0; i < 4; i++){
                                vector<vector<Point>> letter_contours;
                                findContours(fourWay[i], letter_contours, RETR_EXTERNAL,
CHAIN_APPROX_SIMPLE);
                                //drawContours(fourWay[i], letter_contours, 0, 8);
                                //imshow("fourWay" + i, fourWay[i]);
                                sort(letter_contours.begin(), letter_contours.end(),
contour_compare);
                                Rect final_letter = boundingRect(letter_contours[0]);
                                Mat res = fourWay[i](final_letter);
                                fourRes.push_back(res);
                                //imshow("fourRes" + i, res);
                        }
                        //Mat res = rotated_letter.clone();

                        //resize the letter and flatten it
                        vector<Mat> flat;
                        for(int i = 0; i < 4; i++){
                                resize(fourRes[i], fourRes[i], Size(20,20));
                                flat.push_back(fourRes[i].reshape(1,1));//res.total());
                        }

                        vector<Mat> ret(4);
                        vector<vector<float>> neighbors(4);
                        vector<vector<float>> distance(4);
                        // pass to knn and get classifiction
                        for(int i = 0; i < 4; i++){
                                flat[i].convertTo(flat[i], CV_32F);
                                //cout << "flat:  " << flat[i].size() << endl;
                                //cout << "train: " << training_data.size() << endl;
                                //cout << "what" << endl;
                                knn->findNearest(flat[i], 3, ret[i], neighbors[i], distance[i]);
                                //cout << endl << "holy crap ret: " << ret[i] << "  neighbors: " <<
neighbors[i][0] << "  distance: " << distance[i][0] << endl << endl;
                        }
                        int xyz = 0;
                        float minDist = 2000000000;
                        for(int i = 0; i < 4; i++){
```

```
                                float curr = distance[i][0];
                                if(curr < minDist){
                                        minDist = curr;
                                        xyz = i;
                                }
                        }

                        char value = ret[xyz].at<float>(0,0);
                        if(minDist < 4000000){
                                printf("\n\nGUESSED A %c - %f - %d on the RIGHT\n\n", value,
minDist, 2);
                                send('R', value);

                        }
                        else
                                cout << "\nToo far from a letter " << 2 << endl;
                        //printf("Closest distance %d\n", (int)distance[xyz][0]);
                        cout << "closest" << (int)minDist << endl;
                        /*char value = ret.at<float>(0,0);
                        if(distance[0] < 4000000)
                                printf("GUESSED A %c\n", value);
                        else
                                cout << "Too far from a letter" << endl;
                        printf("Closest distance %d\n", (int)distance[0]);*/
                }
        char key = (char)cv::waitKey(1);
        if(key == 'q'){
                std::cout << "BYE" << std::endl;
                break;
        }
        }
        read_data.release();
        read_labels.release();
        cam1.stopVideo();
        cam2.stopVideo();
        cv::destroyAllWindows();


        return 0;
}
```