Android Powered Autonomous Robot

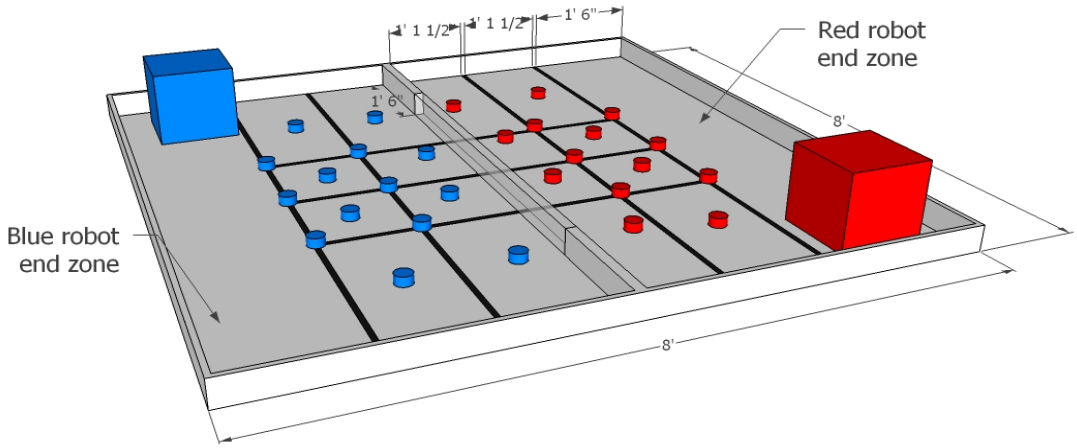
Senior Project

Dennis Cagle - BS Computer Engineering 2012
Zach Negrey - BS Computer Engineering 2012
Dr. John Seng

May 2012

Introduction

The goal of this Senior Project was to create an autonomous robot powered by an Android phone. The robot, named Marvin, was to compete at the Cal Poly Roborodentia 2012 robotics competition. The 2012 competition involved bringing painted cat food cans from a playing field back to an endzone. Each can returned scored 1 point plus an additional 2 points for each can stacked underneath it. After 1 minute or one team collected all the cans on their side, the center wall lifted and the robots were allowed to move to the other side and collect cans from the opposite side. These cans scored an additional point each if brought back to their own endzone.



[The Roborodentia 2012 field]

Design

Overview

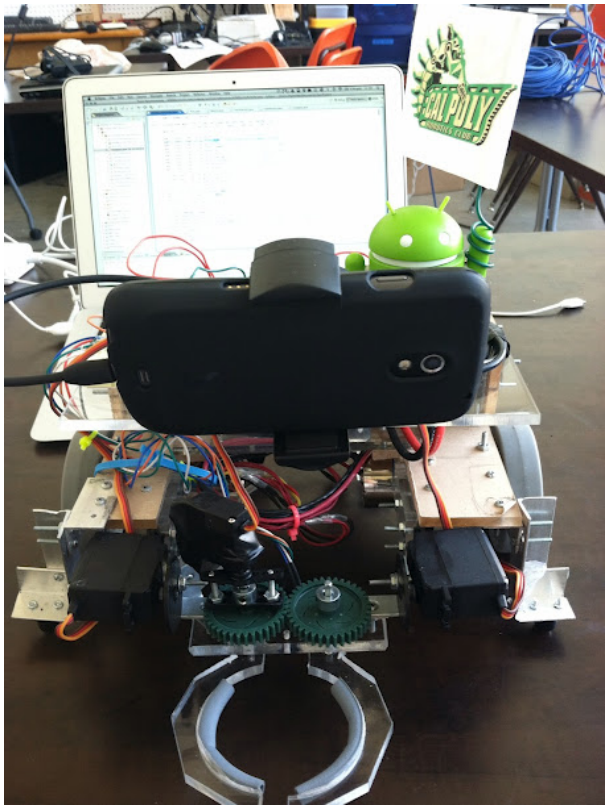
Designed originally to meet the requirements of Roborodentia 2012, Marvin is a 1 ft cubed sized robot capable of driving, picking up objects, and navigating via camera vision and compass directions. Marvin has a relatively simple mechanical design because the focus of the project was placed on integrating the Android phone (a Samsung Galaxy Nexus) with the hardware. Typically, Roborodentia robots use a custom microcontroller (MCU) based off an atmega processor as the “brain” of the bot. This MCU runs code in C and has various sensors and motors interfaced directly with the processor. For Marvin, we used the Android Open Accessory Development Kit (Android ADK) to interface an Android phone with a custom Arduino MCU (Arduino Mega) designed by Google. The Android ADK libraries allow the phone to communicate with the Arduino via some custom APIs, which then interfaces directly with the sensors and motors. The ADK includes open source code for the Android app and the Arduino which served as a starting point for the project. Marvin’s main code exists as a native Android 4.0 application written in Java and C code running on the Arduino Mega.

There were a few major benefits to the Android ADK approach. First, we were able to

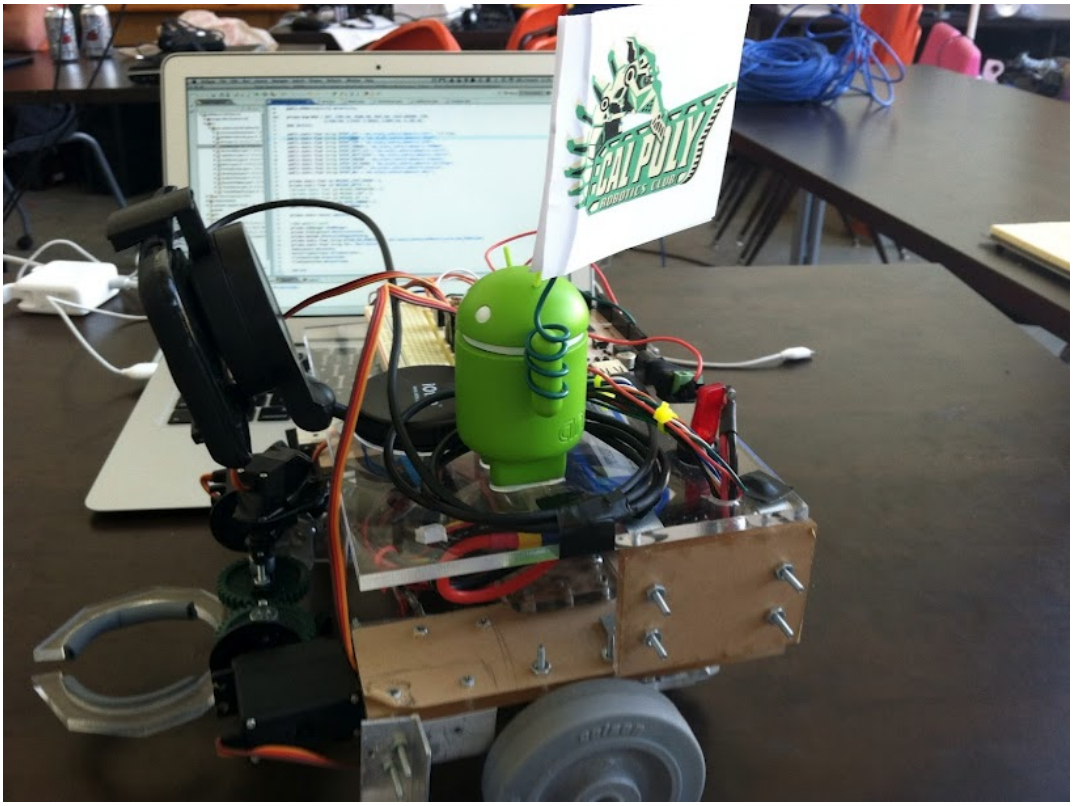
take advantage of the Nexus' processing power, camera, and compass. During the first quarter, Dennis focused on writing vision code that would track red or blue objects using the camera, while Zach focused on interfacing the phone with the Arduino using the ADK. By the end of the first quarter, we had a working prototype that allowed the camera to turn on various leds on the Arduino based on the position of a colored object.

Physical Design

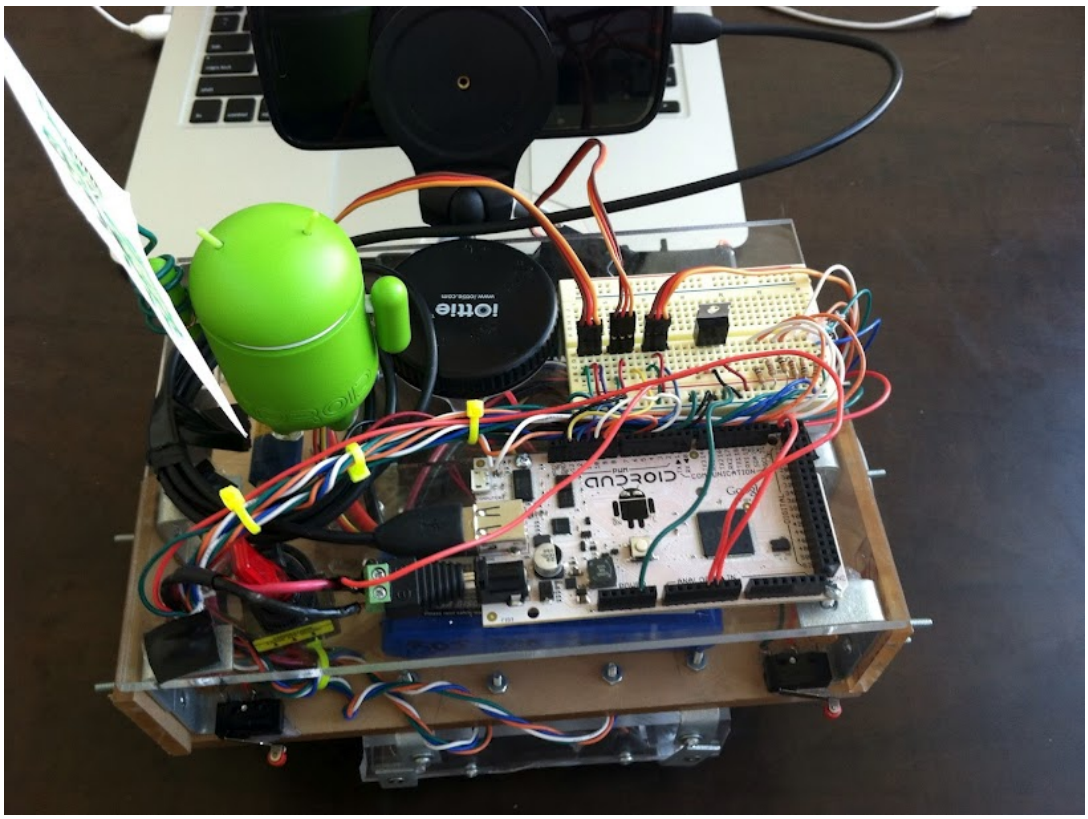
Once we had proven that the Android powered robot concept was possible, we moved on to actually creating the physical robot. The physical design went through a few iterations of simplification based on the funding, time, and the mechanical experience we had. In the end, we decided to limit our stacking capabilities in favor of the more simple design (see Appendix D).



[Marvin Front]

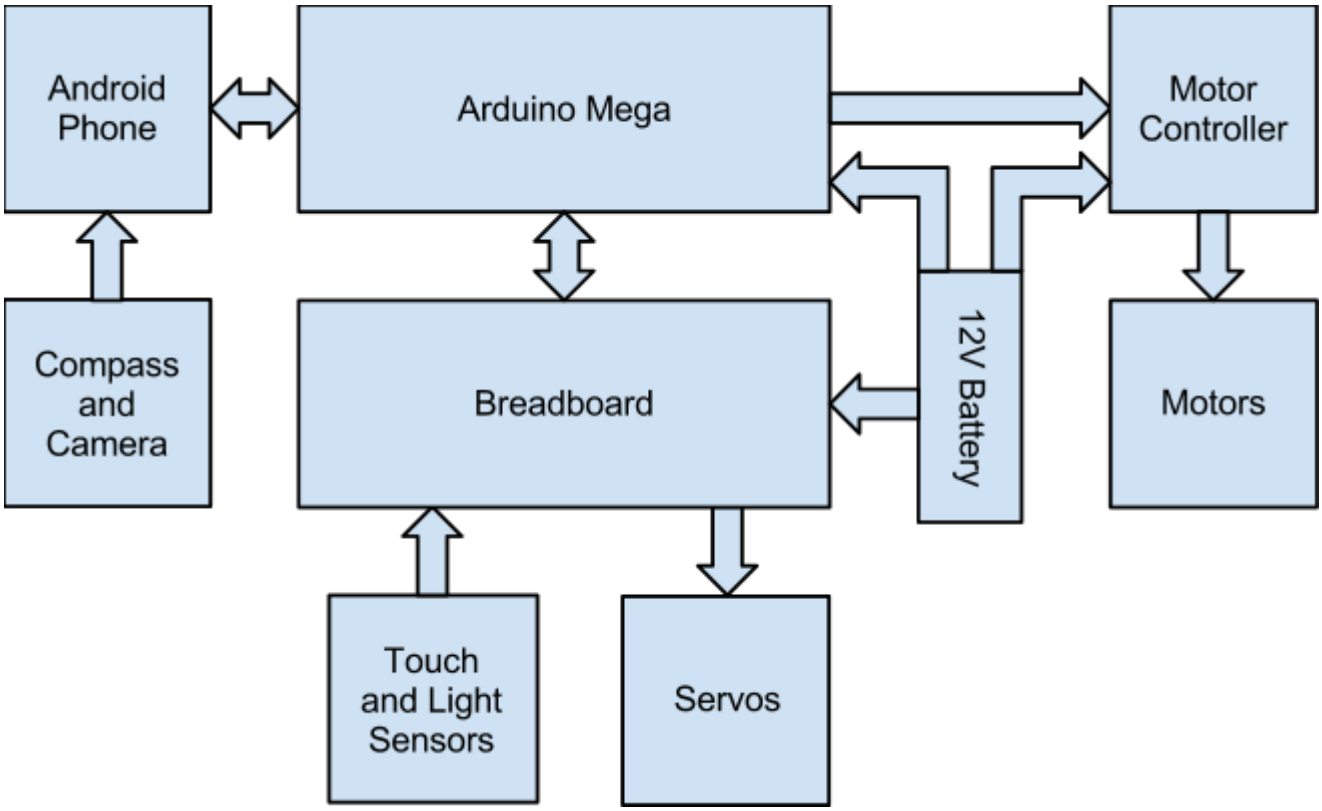


[Marvin Side]



[Marvin Top]

As you can see in the pictures above, Marvin is a multi-layered robot. The bottom layer holds the wheels, motors, motor controller, arm (servos and claw), IR sensors, and some ball casters for stabilization. The middle layer of plexiglas hosts the battery and the main physical structures. Finally, the top level of Marvin contained most of the electronics: the Arduino Mega, a breadboard to connect with various sensors, and the Android Nexus phone. A block diagram depicting the hardware can be seen below:



[Marvin Hardware Block Diagram]

Android Phone ⇔ Arduino Mega

A bidirectional USB connection connects the phone to the Arduino using the Android Open Accessory Development Kit. The Android phone uses the Open Accessory Framework to access its USB port while the Arduino uses the Serial library. The phone will send commands to the motors and servos by sending a special message to the Arduino Mega. The Arduino Mega sends back touch and light sensor values, but only as requested from the phone.

Arduino Mega ⇒ Motors

After the Arduino Mega receives a change motor speed request, it sends commands to the motors through a motor controller using the built in Servo library (the motor controller is set up to respond to servo commands). There is no feedback from the controller or the motors, it just makes a best guess attempt to set the speed.

Arduino Mega ⇒ Servos

After the Arduino Mega receives a change servo position request, it sends commands to the servos through a breadboard connection using the built in Servo library (which sends a PWM signal on the control wire to adjust the position). No feedback is given from these servos.

Compass and Camera ⇒ Android Phone

The compass and the camera continuously send data to the Android application. The application then stores the data in a buffer and provides functions to give the data to the state machine when requested.

Light and Touch Sensors ⇒ Arduino Mega

Initially, we wanted the Arduino Mega to send information back to the Android phone when any interesting event happens (mostly a dramatic change in one of the sensors). Unfortunately, we weren't able to create an asynchronous system like the compass and camera used (mostly because we couldn't figure out how to make non-blocking reads using the Open Accessory framework). Instead, we have the Android Phone's state machine request information from the Arduino (breaking out of a blocking read call), which then grabs the relevant sensor's data and writes it to the USB connection.

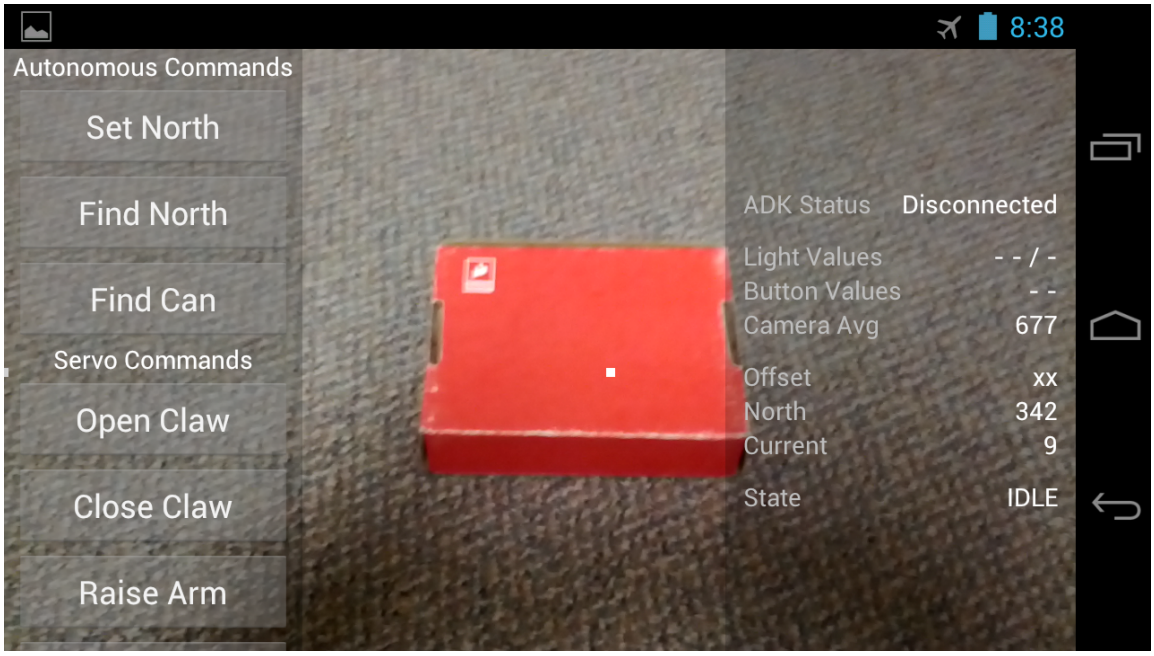
Hardware Troubles

During the early stages of development, we ran into a problem with the servos. They were directly connected to the Arduino Mega and didn't run through an external voltage regulator until after the competition. As a result, whenever the servos stalled, they drew just enough current to overwhelm the Arduino and cause it to reset. This happened fairly consistently whenever we ran into a wall (which happened quite a bit during the competition) and occasionally at random other times. It also occasionally prevented the phone from connecting to the board initially (sometimes we would have to retry 5 or 6 times to get a proper connection). Adding the external voltage regulator prevented these resets from happening and made the connection between the board and the phone much more reliable.

Software Design

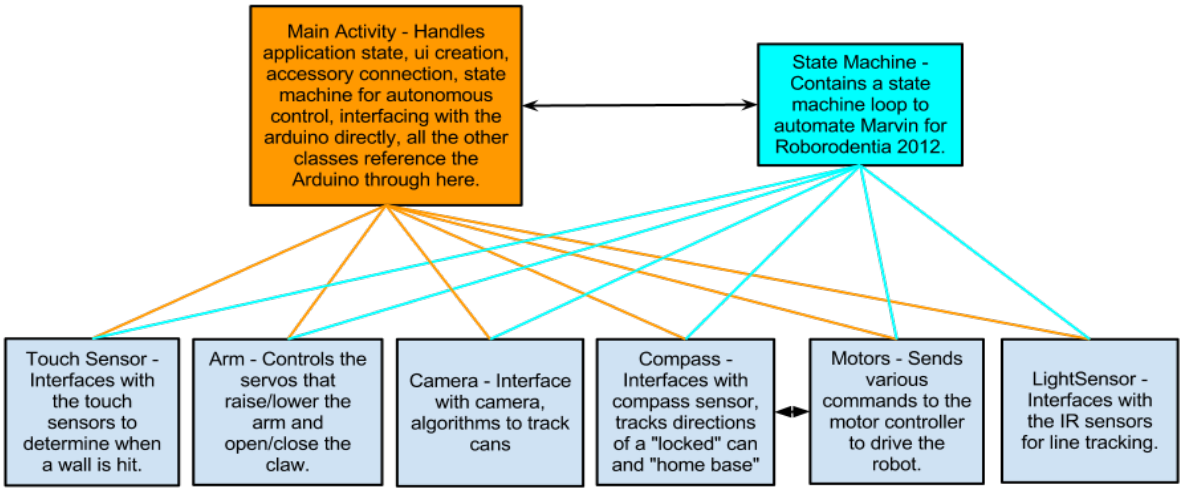
Marvin's software was split into two different programs. The Android application and the Arduino sketch. As has been previously mentioned, the base of both of these programs was Google's Android Open Accessory Development Kit. The ADK library contains the APIs needed to get two-way communication working between the phone and the Arduino.

Android Application



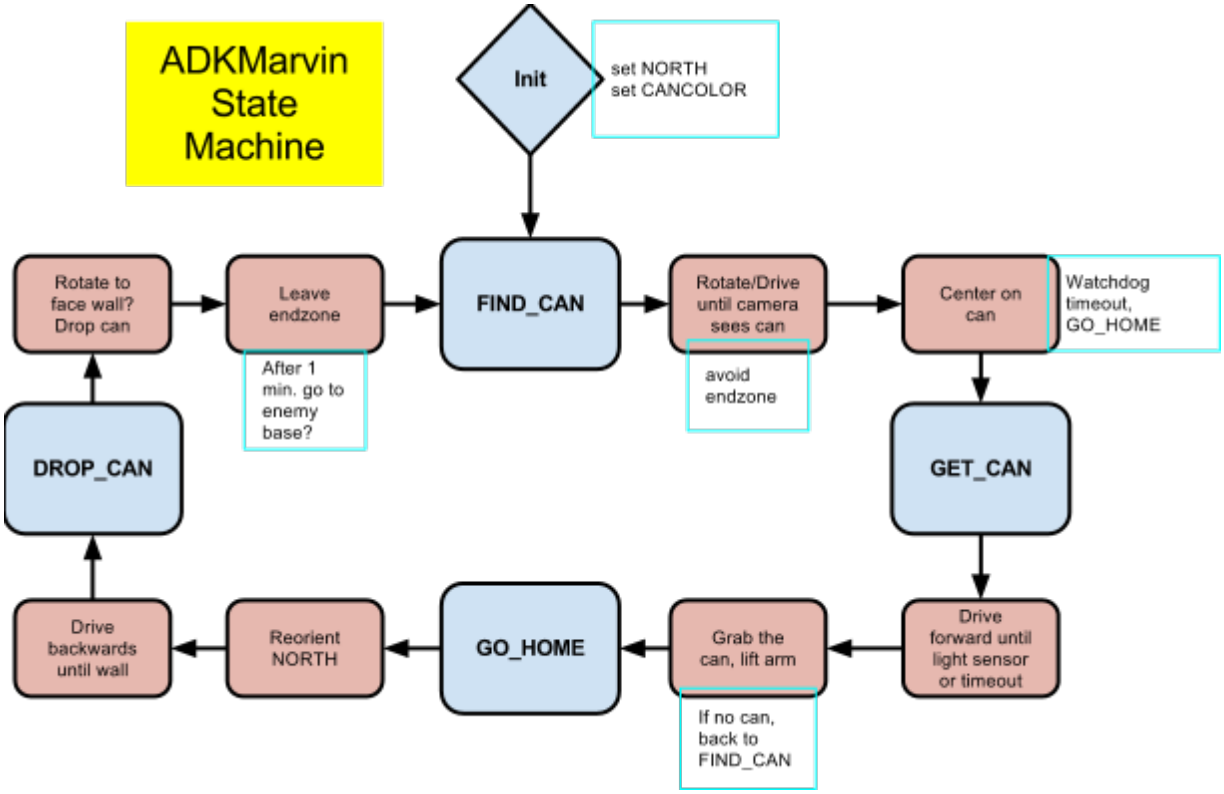
[Android Application Screenshot]

The phone app follows the standard protocols for an Android application, but is limited to 4.0+ devices because of the ADK library support. The diagram below gives a high level overview of the application:



[Android Application Software Overview]

During the competition, Marvin was required to be fully autonomous. This proved to be an interesting problem within the confines of Android, because apps typically follow a “reaction-based” design - a user interacts with a U/I that responds to those actions. Initially, the application was a screen of buttons that would directly make a call to a sensor, motor, or servo to generate a response. The autonomous logic of the application worked best as a state machine that would trigger these reaction-based responses. An overview of the state machine can be seen below:



[Marvin State Machine]

Algorithms:

Motors and Servo Control

The standard Arduino library contains a Servo class that automatically handles sending the appropriate control signals to the Servos (as well as our motor controller). We found that this library suited our needs without any sort of modifications, so we didn't use anything else.

Light and Touch Sensor Data Retrieval

The Arduino is responsible for collecting the sensor data. However, due to a limitation with the Open Accessory library for the Arduino, we weren't able to gather sensor data while simultaneously listening for incoming messages from the Android phone. To solve this, the Android phone sends a sensor data request message to the Arduino, which then receives the message (breaking out of a read call), gathers the sensor information, and writes it back to the Android phone.

Compass Data Retrieval

Currently, Android phones have three sensors used to obtain compass data. The first is the now deprecated Orientation sensor. Because this sensor has been deprecated for some time now, we decided not to use it and focus on the other two non-deprecated sensors: the Accelerometer and Magnetic Field sensors. The Android library contains functions (namely getRotationMatrix and remapCoordinateSystem) to automatically parse the data from these sensors into usable pitch, roll, and azimuth values. Since our robot only turns around the z-axis,

we just needed the azimuth value. The last known value is saved in a variable so that the state machine can poll this value as needed (the value is updated asynchronously).

Camera Algorithm

To find cans, several averages are taken on the Android phone. Camera data is continuously passed into the application through a system function that has a byte array as a parameter. OpenCV converts the YUV format of the array into an RGB array. Then, the program runs through each row and calculates an average x-coordinate for red and blue pixels. A red pixel is defined as a pixel that has a minimum amount of red and a maximum amount of green and blue, and similarly a blue pixel is defined as a pixel that has a minimum amount of blue and a maximum amount of red and green. After these averages are calculated, the total average for each row is calculated (separately for red and blue pixels). These averages can then be used to tell the robot which direction to turn to find a can. The reason this algorithm was chosen was because of its simplicity, speed, and reliability. Other algorithms we experimented with were too slow or didn't always give a direction to go. With the averaging algorithm, even if it went the wrong way at first, theoretically one can would get larger and take up more of the screen. This would make the average lean more and more towards that one can until we were able to grab it.

Conclusion and Future Work

The main requirement for this project was the ability to compete in Roborodentia. We were able to score some points during the competition by driving around and controlling the grabber, but there was still much work that needed to be done by the time of the competition. Most of this work was in the compass code. Without properly working compass code, we weren't able to use our camera due to the fact that we had no way to re-orient ourselves after we find a can. However, we continued work on Marvin after the competition and were eventually able to successfully complete all of the stages of our state machine algorithm. Marvin can now scan a field for a red can, lock onto the can, pick the can up, return it to the starting area, and repeat the process. Hopefully, Marvin will be able to compete in Roborodentia 2013.

We would like to continue work on Marvin by adding some system for wireless remote control (Bluetooth or Wifi). It would be interesting to look into streaming the video feed from the phone on Marvin to a different remote Android device that could then drive Marvin. Another interesting idea we would like to implement would be to send Marvin to a GPS coordinate autonomously.

Ultimately, this project was a great application of our Computer Engineering experience at Cal Poly. It contained design and implementation of hardware, electronic devices, and software. We applied our knowledge of materials engineering, physics, and the ability to work within system constraints to come up with a physical design. We were able to apply our Electrical Engineering knowledge to interface our sensors properly with the MCU and solve the voltage regulation issues. Finally, we applied both high level and firmware level software knowledge in the Android and Arduino applications, and successfully interconnected the systems together.

Bill of materials

Name	Supplier	Quantity	Price (Each)
5V Regulator	Radio Shack	1	$1.99
Large Servo	Sparkfun	2	$12.95
Medium Servo	Sparkfun	1	$10.95
Colson Wheels 3” x ⅞”	Robot Marketplace	2	$2.95
Door Weatherproofing	Home Depot	1	$4.98
L-Brackets	Home Depot	1	$5
Ball Casters	Sparkfun	4	$2.95
DC Barrel Jack Adapter	Sparkfun	1	$2.95
Arduino Mega	Amazon	1	$49.99
Motor Controller	Robotics Club	1	$60
Plexi Glass	Home Depot	2	$10
Various Machine Screws	Home Depot	1	$10
I/R Sensors	Sparkfun	3	$7
Total			**$220**

References

- Google Open Accessory Development Kit
  - https://developer.android.com/guide/topics/usb/adk.html
- USB Accessory Documentation
  - https://developer.android.com/guide/topics/usb/accessory.html
- Android Developer Documentation
  - https://developer.android.com/index.html
- QRB1134 Datasheet
  - http://www.datasheetcatalog.com/datasheets_pdf/Q/R/B/1/QRB1134.shtml
- OpenCV
  - http://opencv.willowgarage.com/wiki/
  - http://code.opencv.org/projects/opencv/wiki/OpenCV4Android

Appendices

a. Project Role assignments

Dennis Cagle was initially responsible for getting the camera code working and then for creating the initial design of the robot. After that was done, he moved to reprogramming

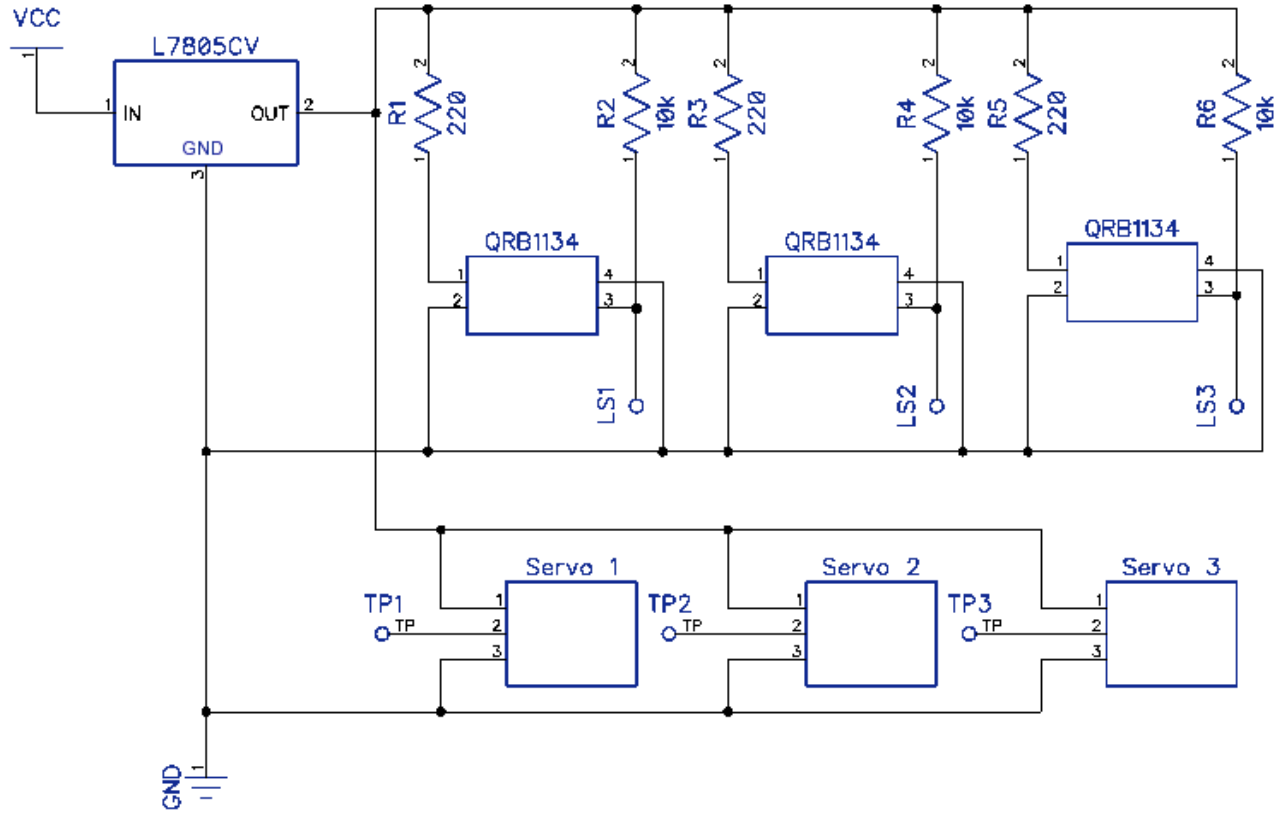
the Arduino to control the servos and sensors. Finally, he wrote Android compass code, and designed and implemented the final U/I for the application.

Zach Negrey was responsible for getting the Android ADK (Phone to Arduino) connection working and creating the initial Android application, and then redesigned the front grabber arm design. Towards the end of the project, he moved to working on the final state machine code.

b. Project personas

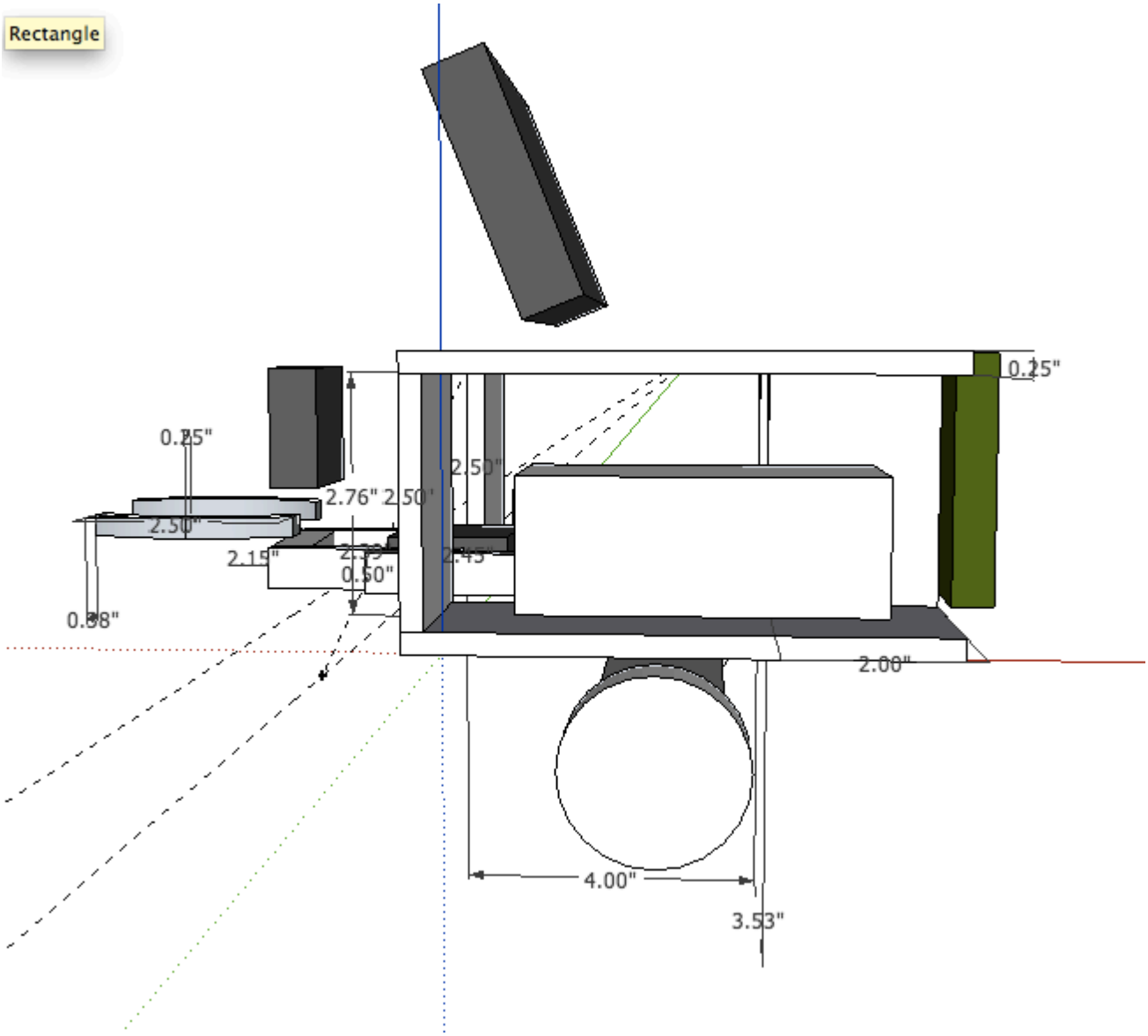
The target user for this system would be somebody who wants a decent robot to control using an Android phone. Ideally, the sensors and motor calls would be broken up into easy to use classes, although the user would need some technical knowledge to modify the code for any other types of sensors. Members of the Cal Poly Robotics club would be ideal users for this project.

c. Circuit diagrams, layouts, CAD drawings, etc.

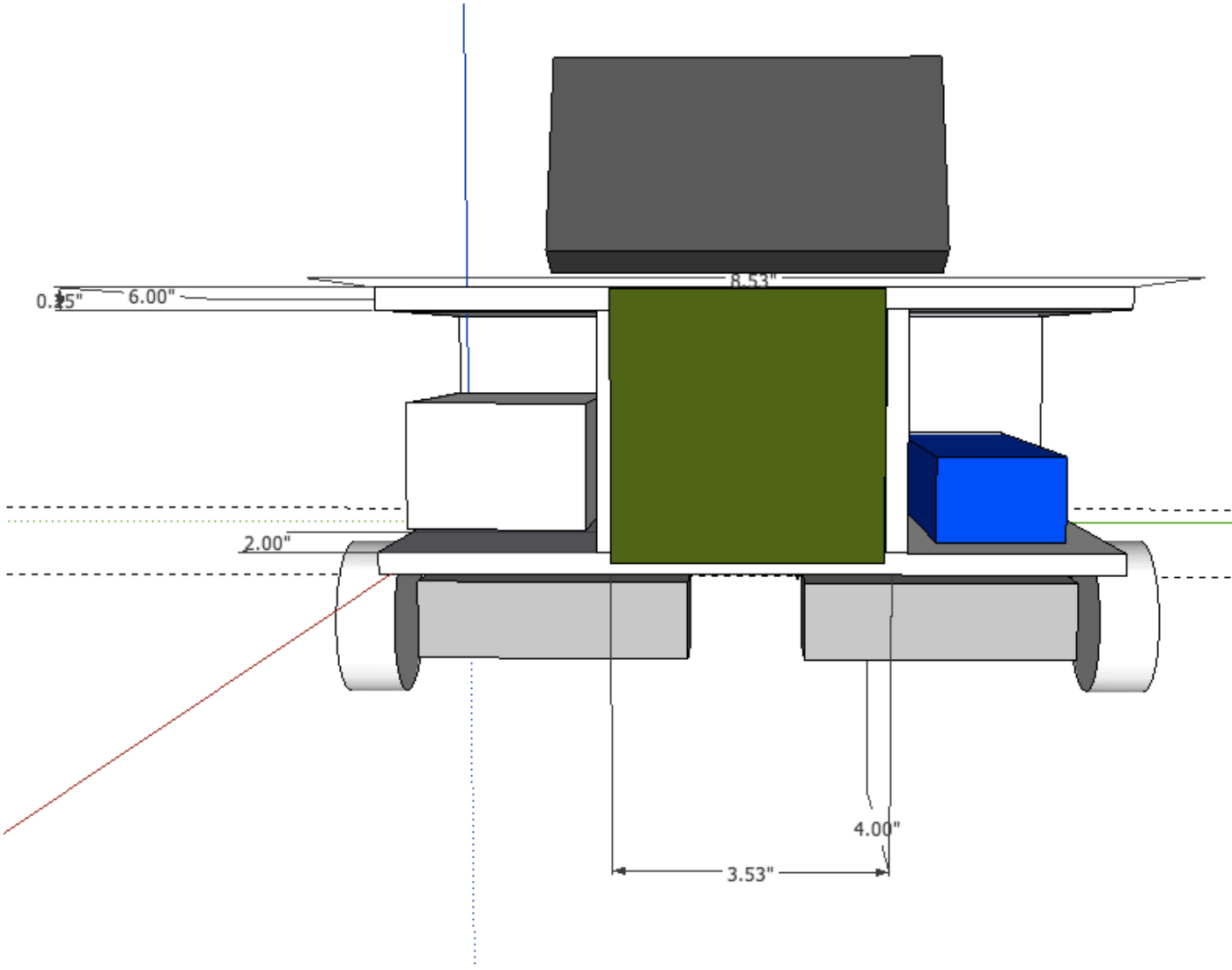


[Circuit Diagram for Light Sensors and Servos]

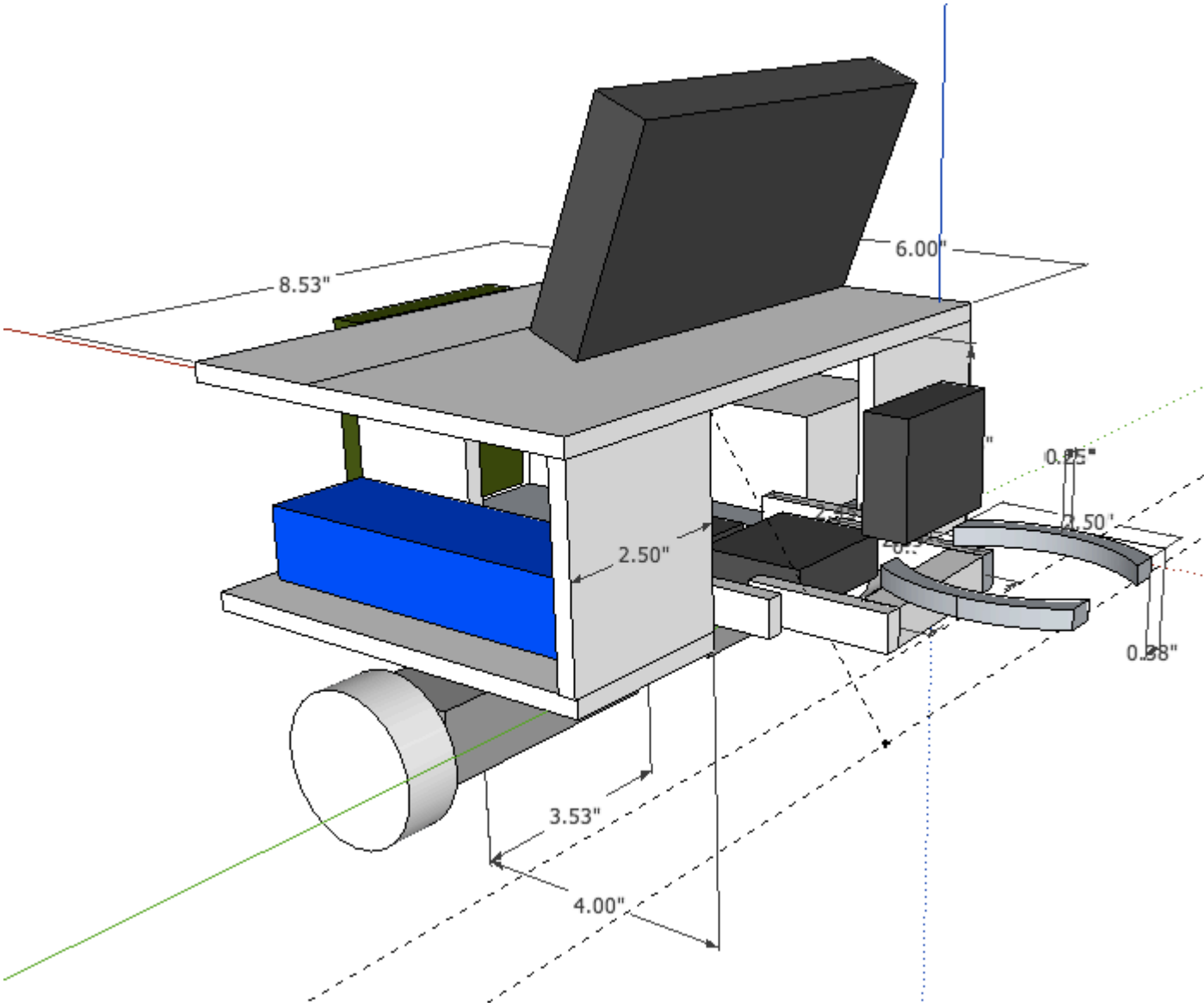
The light sensors required a current-limiting resistor to the input and a pull-up resistor on the output. They ran off the external voltage regulator along with the servos. The touch sensors were just plugged directly into the Arduino with the internal pull-ups enabled.



[Side view of the final design]



[Back view of the final design]



[Overall picture of final design]

d. Team Member Summary

Dennis Cagle

During winter quarter, I tested the camera code. At first, I based the code off of an OpenCV sample that didn't work. On my Droid X phone, the screen just remained black while Zach's Galaxy Nexus phone just crashed when attempting to start the application. I found out that (apparently quite a few) Android phones require a surface to draw a camera preview to get any data from the camera back (even if the application doesn't need to draw the preview). After learning this, I modified the sample code to work with both of our phones and began thinking about algorithms to detect cans. I decided that the simplest way was to just use a bunch of averages. By the end of winter quarter, I had this algorithm working pretty well by testing it on our old Roborodentia 2011 bot. I also came up with a rough design for the robot itself, although no construction was done this quarter.

At the beginning of spring quarter, both of us changed our focus to building the

hardware. We constructed files in Solidworks and finally used the laser cutter to begin construction. Once the robot was mostly assembled, I moved on to integrating our remaining non-Android sensors and programming the Arduino. The Google ADK board came with a demo firmware that we had been using up to this point, but we found that it wasn't enough to control our robot so we changed the code. The biggest issue here was coming up with a system to always read data from the Android phone while still monitoring the non-Android sensors. Normally, Arduinos provide a non-blocking way to check if data is available, but the Open Accessory framework library blocked when trying to use this call. We couldn't figure out a way around this in time, so we just had the Android phone request non-Android sensor data every now and then.

Zach Negrey

During the Winter quarter, I focused on creating a base Android application that would work with the Arduino Mega board. Most of this work involved researching various documentation from Google and tutorials online. Google actually provides an open source example application for use with the ADK board, so I based our MarvinADK application off that design. Once the app could communicate with the ADK board, I set out to create more custom abstractions that we could use in the future to control our various sensors. Toward the end of the quarter, I worked on adding Dennis' vision code into the app and got the vision code to use the created abstractions to light up leds on the ADK board.

The physical hardware build took us much longer then we had anticipated. I spent many hours working plexi glass into the different shapes we required, screwing various components onboard, and then wiring up the electronics for the system. The most delicate part of the bot ended up being the arm because the servos had to be perfectly aligned not only physically, but in the software as well. I went through three different revisions of an arm before we finally had something that would pick up and hold a can.

Once Marvin took physical shape, I set about individually testing each of the sensors/ motors/servos individually to generate working classes that we could use later for the state machine. After each part worked on its own, I began designing the state machine for the autonomous run during the competition. Unfortunately, the entire system did not come together reliably enough the morning of Roborodentia, so I made a last minute scrap to manually code in what was essentially the autonomous state machine code. It worked decently during testing, but in the actual competition, the power issues mentioned in the report would cause the manual run to re-init and get confused.

After the competition, I worked with Dennis to tweak the hardware, and re-write much of the Android application. One of the persistent issues we had with the ADK board APIs is that they could only be run from the main activity of the application. While I had originally written broadcast receivers to pass messages between the various classes/activities of the application, we ended up having the main activity control the ADK required commands and ended up with much cleaner code.

e. Code

Android App:

```
/* Senior Project Winter 2012
 * Robodentia 2012
 * Create an Android powered robot
 * Dennis Cagle and Zach Negrey
 */

// ADB:
// adb connect <mac>
```

```
// adb install app.apk
// adb logcat -s "marvin"

package edu.calpoly.android.adkmarvin;

import java.io.FileDescriptor;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Timer;
import java.util.TimerTask;

//import com.android.future.usb.UsbAccessory;
//import com.android.future.usb.UsbManager;
import android.graphics.Rect;
import android.hardware.Sensor;
import android.hardware.SensorManager;
import android.hardware.usb.UsbAccessory;
import android.hardware.usb.UsbManager;
import android.support.v4.app.ServiceCompat;
import android.support.v4.content.LocalBroadcastManager;

import android.app.Activity;
import android.app.PendingIntent;
import android.app.Service;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.ParcelFileDescriptor;
import android.util.Log;
import android.view.View;
import android.view.Window;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class ADKMarvinActivity extends Activity implements Runnable {

   private Button btn_openClaw, btn_closeClaw, btn_raiseArm, btn_lowerArm;
   private Button btn_stop, btn_forward, btn_backward, btn_left, btn_right;
   private Button btn_set_north, btn_find_north, btn_find_can;
   public TextView tv1, lightValues, cameraValues, currentState;
   public TextView tv_returnAngle, tv_northAngle, tv_currentAngle;
   public ADKMarvinActivity mainActivity;

   private enum MODE { INIT, FIND_CAN, FOUND_CAN, HAVE_CAN, LEAVE_ENDZONE, STOP,
                                                        H_FIND_CAN, H_STACK, H_TOBASE, H_DROP_CAN, H_LINE_UP};
   MODE BOTSTATE;

   // Broadcast Receiver:
   public static final String INTENT_ROTATE_LEFT = "edu.calpoly.android.adkmarvin.rLeft";
   public static final String INTENT_ROTATE_RIGHT = "edu.calpoly.android.adkmarvin.rRight";
   public static final String INTENT_CAMERA_VALUE = "edu.calpoly.android.adkmarvin.cvalue";

   private static final int MESSAGE_LIGHT_SENSOR = 1;
   private static final int MESSAGE_SWITCH = 2;
   //private static final int MESSAGE_TEMPERATURE = 2;
   //private static final int MESSAGE_LIGHT = 3;
   //private static final int MESSAGE_JOY = 4;
```

```
public static final byte LED_SERVO_COMMAND = 2;
public static final byte RELAY_COMMAND = 3;

private static Context appContext;

//ADK specific stuff:
private UsbManager mUsbManager;
private PendingIntent mPermissionIntent;
private boolean mPermissionRequestPending;
private static final String ACTION_USB_PERMISSION = "edu.calpoly.android.ADKMarvin.action.USB_PERMISSION";
private static final String TAG = "MarvinDroid";
UsbAccessory mAccessory;
ParcelFileDescriptor mFileDescriptor;
FileInputStream mInputStream;
FileOutputStream mOutputStream;
//SerialKeepAlive keepAliveThread;

// Driver references
Arm arm;
Motors motors;
LightSensor lightSensor;
TouchSensor btns;
Compass compass;
StateMachine stateMachine;
CameraData camera;

public long cameraValue = 0;

boolean seeacan = false;
int capturedcans = 0;
boolean timerExpired = false;

/* START OVERHEAD FOR BROADCAST RECEIVER */
/*LocalBroadcastManager mLocalBroadcastManager;
BroadcastReceiver mReceiver;

@Override
protected void onDestroy() {
        super.onDestroy();
        //unregisterBroadcastReceiver();
}

protected void unregisterBroadcastReceiver(){
        mLocalBroadcastManager.unregisterReceiver(mReceiver);
        //stopService(new Intent(ADKMarvinActivity.this, LocalService.class));

}

protected void registerBroadcastReceiver() {
        // We are going to watch for interesting local broadcasts.
        mLocalBroadcastManager = LocalBroadcastManager.getInstance(this);

       IntentFilter filter = new IntentFilter();
       filter.addAction(INTENT_ROTATE_LEFT);
       filter.addAction(INTENT_ROTATE_RIGHT);
       filter.addAction(INTENT_CAMERA_VALUE);
       mReceiver = new BroadcastReceiver() {
       @Override
       public void onReceive(Context context, Intent intent) {
                // FSM here.
                String action = intent.getAction();
                if (action.equals(INTENT_ROTATE_LEFT))
                {
                        motors.rotateLeft();
                }
                else if (action.equals(INTENT_ROTATE_RIGHT))
```

```
                {
                        motors.rotateRight();
                }
                else if (action.equals(INTENT_CAMERA_VALUE))
                {
                        cameraValue = intent.getLongExtra("value", 0);
                        //Log.d("marvin", "main: "+cameraValue);
                }
        }
        };
        mLocalBroadcastManager.registerReceiver(mReceiver, filter);
}*/


/* END OVERHEAD FOR BROADCAST RECEIVER */

        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        //setContentView(R.layout.opencv_main);
        //registerBroadcastReceiver();
        //start the broadcast
        //startService(new Intent(ADKMarvinActivity.this, LocalService.class));
        //mLocalBroadcastManager.sendBroadcast(new Intent(CUSTOM_INTENT));
        // LocalBroadcastManager.getInstance(this).sendBroadcast(new Intent(CUSTOM_INTENT));
        Log.v("marvin", "launched");
        appContext = getApplicationContext();
        mainActivity = this;
        BOTSTATE = MODE.INIT;
        initScreen();
        initBot();
        connectADK();
        //startOpenCV();
        //motors.delay(2000);
        //hardrun();
        }


        @Override
public void onResume() {
        super.onResume();
        //registerBroadcastReceiver();

        // ADK start:
        Intent intent = getIntent();
        if (mInputStream != null && mOutputStream != null) {
                return;
        }

        UsbAccessory[] accessories = mUsbManager.getAccessoryList();
        UsbAccessory accessory = (accessories == null ? null : accessories[0]);
        if (accessory != null) {
                if (mUsbManager.hasPermission(accessory)) {
                        openAccessory(accessory);
                } else {
                        synchronized (mUsbReceiver) {
                                if (!mPermissionRequestPending) {
                                        mUsbManager.requestPermission(accessory,
                                                        mPermissionIntent);
                                        mPermissionRequestPending = true;
                                }
                        }
                }
```

```
		} else {
			Log.d(TAG, "mAccessory is null");
		}
		/*
		Log.d(TAG, "before");
		// Start the compass:
		mSensorManager.registerListener(mCompass, mSensor, SensorManager.SENSOR_DELAY_GAME);
		Log.d(TAG, "registeredCompass");
		*/
		// Start the compass and state machines
		compass.startCompass();
		stateMachine.restart();
		BOTSTATE = MODE.INIT;
		//hardrun();
	}

		@Override
		public void onPause()
		{
		// If the app exits...panic.
		motors.stop();
		arm.lower();
		compass.stopCompass(this);
		// Stop the compass:
		//mSensorManager.unregisterListener(mCompass);
		if (mUsbReceiver != null)
			this.unregisterReceiver(mUsbReceiver);

		// Stop the worker threads
		stateMachine.cancel();
		closeAccessory();
		super.onPause();
		//unregisterBroadcastReceiver();
		}


		public void initScreen()
	{
		setContentView(R.layout.btnlayout);
		btn_openClaw = (Button) this.findViewById(R.id.openclaw);
		btn_closeClaw = (Button) this.findViewById(R.id.closeclaw);
		btn_raiseArm = (Button) this.findViewById(R.id.raisearm);
		btn_lowerArm = (Button) this.findViewById(R.id.lowerarm);
		btn_forward = (Button) this.findViewById(R.id.forward);
		btn_backward = (Button) this.findViewById(R.id.backward);
		btn_left = (Button) this.findViewById(R.id.left);
		btn_right = (Button) this.findViewById(R.id.right);
		btn_stop = (Button) this.findViewById(R.id.stop);
		btn_set_north = (Button) this.findViewById(R.id.setNorth);
		btn_find_north = (Button) this.findViewById(R.id.findNorth);
		btn_find_can = (Button) this.findViewById(R.id.findCan);
		tv_returnAngle = (TextView) this.findViewById(R.id.returnAngle);
		tv_northAngle = (TextView) this.findViewById(R.id.northAngle);
		tv_currentAngle = (TextView) this.findViewById(R.id.currentAngle);
		currentState = (TextView) this.findViewById(R.id.currentState);

		btn_openClaw.setOnClickListener(new OnClickListener() {
			@Override public void onClick(View view) {
				Log.v("marvin", "open claw");
				arm.open_claw();
				//mystack();
				//hardrun();
				//startOpenCV();
		}});
		btn_closeClaw.setOnClickListener(new OnClickListener() {
			@Override public void onClick(View view) {
```

```
                        Log.v("marvin", "close claw");
                        arm.close_claw();
});
btn_raiseArm.setOnClickListener(new OnClickListener() {
        @Override public void onClick(View view) {
                        arm.raise();
});
btn_lowerArm.setOnClickListener(new OnClickListener() {
        @Override public void onClick(View view) {
                        arm.lower();
});
btn_forward.setOnClickListener(new OnClickListener() {
        @Override public void onClick(View view) {
                        motors.forward();
});
btn_backward.setOnClickListener(new OnClickListener() {
        @Override public void onClick(View view) {
                        motors.back();
});
btn_stop.setOnClickListener(new OnClickListener() {
        @Override public void onClick(View view) {
                        motors.stop();
});
btn_left.setOnClickListener(new OnClickListener() {
        @Override public void onClick(View view) {
                        motors.rotateLeft();
});
btn_right.setOnClickListener(new OnClickListener() {
        @Override public void onClick(View view) {
                        motors.rotateRight();
});
btn_set_north.setOnClickListener(new OnClickListener() {
        @Override public void onClick(View view) {
                        //mainActivity.orientNorth();
                        //mainActivity.findNorth();
                        compass.setNorth();
});
btn_find_north.setOnClickListener(new OnClickListener() {
        @Override public void onClick(View view) {
                        stateMachine.requestRotate(0);
});
// This button will launch the OpenCV activity, which overrides this GUI with
//   the camera activity's GUI. It also starts the autonomous competition routines.
btn_find_can.setOnClickListener(new OnClickListener() {
        @Override public void onClick(View view) {
                        stateMachine.beginAutonomous();
});
tv1 = (TextView) this.findViewById(R.id.adkstatus);
lightValues = (TextView) this.findViewById(R.id.lightValues);
cameraValues = (TextView) this.findViewById(R.id.cameraValue);
camera = (CameraData) this.findViewById(R.id.cameraView);
camera.setCallback(new CameraCallback()
{
        @Override
        public void cameraChanged(ArrayList<Rect> objects)
        {
                if (objects.size() > 0)
                {
                        stateMachine.updateCameraTargets(
                                        objects.get(0).centerX(), objects.get(0).centerX());
                        cameraValue = objects.get(0).centerX();
                        cameraValues.post(new Runnable()
                        {
                                public void run()
                                {
                                        cameraValues.setText("" + cameraValue);
```

```
                                                }
                                        });
                                }
                        }
                });
        }

                public void initBot()
                {
                 // Instantiate the driver objects
                 arm = new Arm(this);
                 motors = new Motors(this);
                 lightSensor = new LightSensor(this);
                 btns = new TouchSensor(this);
                 compass = new Compass(this, this);
                 // Initialize the servos and motors
                 arm.lower();
                 arm.open_claw();
                 motors.stop();
                 // Create (but don't start) the state machine
                 stateMachine = new StateMachine(motors,compass, this, arm, btns, lightSensor);
                }

                public void startOpenCV()
                {
                 Toast.makeText(this, "OpenCV", Toast.LENGTH_SHORT).show();
                 Intent myIntent = new Intent(appContext, Sample1Java.class);
                 this.startActivity(myIntent);
                }

                /* ADK SPECIFIC COMMANDS : */


                public void connectADK()
        {

                 mUsbManager = (UsbManager) getSystemService(Context.USB_SERVICE);
                 mPermissionIntent = PendingIntent.getBroadcast(this, 0, new Intent(
                                         ACTION_USB_PERMISSION), 0);
                 IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
                 filter.addAction(UsbManager.ACTION_USB_ACCESSORY_DETACHED);
                 registerReceiver(mUsbReceiver, filter);

                 if (getLastNonConfigurationInstance() != null) {
                                 mAccessory = (UsbAccessory) getLastNonConfigurationInstance();
                                 //mAccessory = (UsbAccessory) intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
                                 openAccessory(mAccessory);
                 }
                 BOTSTATE = MODE.INIT;
                 arm.lower();
                 motors.stop();
        }

        private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
                 @Override
                 public void onReceive(Context context, Intent intent) {
                                 String action = intent.getAction();
                                 if (ACTION_USB_PERMISSION.equals(action)) {
                                                 synchronized (this) {
                                                                 //UsbAccessory accessory = UsbManager.getAccessory(intent);
                                                                 UsbAccessory accessory = (UsbAccessory)
intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
                                                                 if (intent.getBooleanExtra(
                                                                                         UsbManager.EXTRA_PERMISSION_GRANTED, false)) {
                                                                         //tv2.setText("openAccessory");
                                                                         openAccessory(accessory);
```

```
                                        tv1.setText("Connected");
                                } else {
                                        Log.d(TAG, "permission denied for accessory "
                                                        + accessory);
                                        //tv2.setText("permission denied");
                                }
                                mPermissionRequestPending = false;
                        }
                } else if (UsbManager.ACTION_USB_ACCESSORY_DETACHED.equals(action)) {
                        //UsbAccessory accessory = UsbManager.getAccessory(intent);
                        UsbAccessory accessory = (UsbAccessory) intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
                        if (accessory != null && accessory.equals(mAccessory)) {
                                closeAccessory();
                                tv1.setText("Disconnected");
                        }
                }
        }
};

private void openAccessory(UsbAccessory accessory) {
        mFileDescriptor = mUsbManager.openAccessory(accessory);
        if (mFileDescriptor != null) {
                mAccessory = accessory;
                FileDescriptor fd = mFileDescriptor.getFileDescriptor();
                mInputStream = new FileInputStream(fd);
                mOutputStream = new FileOutputStream(fd);
                Thread thread = new Thread(null, this, "ADKMarvin");
                thread.start();
                //keepAliveThread = new SerialKeepAlive(mOutputStream); // Spam this port with a useless message
                //keepAliveThread.start();
                Log.d(TAG, "accessory opened");
                tv1.setText("Connected");
                //tv1.setText("ADK Connected");
                Toast.makeText(this, "ADK Connected", Toast.LENGTH_SHORT).show();
        } else {
                Log.d(TAG, "accessory open fail");
                tv1.setText("Error");
                //tv1.setText("ADK FAILURE to connect");
        }
}

private void closeAccessory() {
        //tv1.setText("ADK session Closed!");
        Toast.makeText(this, "ADK Session Closed", Toast.LENGTH_SHORT).show();
        try {
                if (mFileDescriptor != null) {
                        mFileDescriptor.close();
                }
        } catch (IOException e) {
        } finally {
                mFileDescriptor = null;
                mAccessory = null;
        }
        motors.stop();
        arm.lower();
        tv1.setText("Closed");
}

/* run() gets values directly from the arduino and handles in mHandler */
public void run() {
        int ret = 0;
        byte[] buffer = new byte[16384];
        int i;

        while (ret >= 0) {
                try {
```

```
                ret = mInputStream.read(buffer);
            } catch (IOException e) {
                break;
            }

            i = 0;
            while (i < ret) {
                int len = ret - i;

                switch (buffer[i]) {
                case 0x1:
                    /*
                    if (buffer[i + 1] != 2)
                        Log.v("marvin", "got light msg: " + buffer[i] + ", " + buffer[i + 1] + ", " + buffer[i + 2]);
                    */
                    if (len >= 3) {
                        Message m = Message.obtain(mHandler, MESSAGE_LIGHT_SENSOR);
                        m.obj = new LightMsg(buffer[i + 1], buffer[i + 2]);
                        mHandler.sendMessage(m);
                        i += 3;
                    }
                    break;
                case 0x2:
                    if (len >= 3) {
                        Message m = Message.obtain(mHandler, MESSAGE_SWITCH);
                        m.obj = new LightMsg(buffer[i + 1], buffer[i + 2]);
                        mHandler.sendMessage(m);
                        mHandler.post(new Runnable()
                        {
                            public void run()
                            {
                                Toast.makeText(appContext, "Button Sensor Tripped",
Toast.LENGTH_SHORT).show();
                            }
                        });
                    }
                    i += 3;
                    break;

                /*case 0x5:
                    if (len >= 3) {
                        Message m = Message.obtain(mHandler, MESSAGE_LIGHT);
                        m.obj = new LightMsg(composeInt(buffer[i + 1],
                                buffer[i + 2]));
                        mHandler.sendMessage(m);
                    }
                    i += 3;
                    break;*/

                default:
                    Log.d(TAG, "unknown msg: " + buffer[i]);
                    i++; // Unknown message, move to next character
                    //i = len;
                    mHandler.post(new Runnable()
                    {
                        public void run()
                        {
                            Toast.makeText(appContext, "Unknown message",
Toast.LENGTH_SHORT).show();
                        }
                    });
                    break;
                }
            }

        }
```

```
}

private int composeInt(byte hi, byte lo) {
	int val = (int) hi & 0xff;
	val *= 256;
	val += (int) lo & 0xff;
	return val;
}

Handler mHandler = new Handler() {
	@Override
	public void handleMessage(Message msg) {
		switch (msg.what) {
		case MESSAGE_LIGHT_SENSOR:
			LightMsg ls = (LightMsg) msg.obj;
			handleLightSensorMessage(ls);
			break;
		case MESSAGE_SWITCH:
			LightMsg swm = (LightMsg) msg.obj;
			handleButtonMessage(swm);
			break;
		}
	}
};

public class LightMsg {
	private byte sw;
	private byte state;

	public LightMsg(byte sw, byte state) {
		this.sw = sw;
		this.state = state;
	}

	public byte getSw() {
		return sw;
	}

	public byte getState() {
		return state;
	}
}

private void handleLightSensorMessage(LightMsg lm)
{
	// Pass this information to the LightSensor class
	byte[] states = lightSensor.handleLightSensorMessage(lm, stateMachine);
	lightValues.setText(states[0] + " " + states[1] + " / " + states[2]);
}

private void handleButtonMessage(LightMsg lm)
{
	//Log.v("marvin", "btn press");
	btns.onTouchEvent(lm, stateMachine);
}

public void sendCameraMessage(int value)
{
	this.cameraValue = value;
}

//sendCommand to the arduino
public void sendCommand(byte command, byte target, int value) {
	byte[] buffer = new byte[3];
	if (value > 255)
		value = 255;
```

```
		buffer[0] = command;
		buffer[1] = target;
		buffer[2] = (byte) value;
		if (mOutputStream != null && buffer[1] != -1) {
			synchronized (mOutputStream)
			{
				try {
					if (command == LED_SERVO_COMMAND)
					{
						//Toast.makeText(this, "DKA: target"+((Byte)target).toString()+" value:"+((Integer)
value).toString(), Toast.LENGTH_SHORT).show();
					}
					mOutputStream.write(buffer);
				} catch (IOException e) {
					Log.e(TAG, "write failed", e);
				}
			}
		}
	}

	/* END ADK SPECIFIC COMMANDS */
}


// State machine for Marvin:

package edu.calpoly.android.adkmarvin;

import java.util.Timer;
import java.util.TimerTask;

import android.util.Log;

public class StateMachine
{
	private enum STATE { IDLE, FIND_CAN, GET_CAN, GO_HOME, DROP_CAN, EXIT, ROTATE_ONCE };
	private STATE state = STATE.IDLE;

	// References to other classes
	private Motors motors;
	private Compass compass;
	private Arm arm;
	private ADKMarvinActivity adkActivity;
	private TouchSensor touch;
	private LightSensor light;

	// Thread variables
	private Timer watchdog;
	private TimerTask task;
	private StateMachineThread thread;

	private long oldCameraVal = 0;

	// Variables to keep track of sensor events
	private boolean hasCan = false;
	private boolean onWall = false;
	private int linesPassed = 0;
	private int cameraRedTarget = -1;
	private int cameraBlueTarget = -1;
	private boolean beginCalled = false;
	private boolean rotateRequested = false;
	private int desiredRotateVal = 0;

	// init the class and references to other classes
	public StateMachine(Motors motors, Compass compass, ADKMarvinActivity actin, Arm arm, TouchSensor touch, LightSensor light)
```

```
{
	adkActivity = actin;
	this.motors = motors;
	this.compass = compass;
	this.arm = arm;
	this.touch = touch;
	this.light = light;
}

/* State machine public helper methods */

public void requestRotate(int angle)
{
	desiredRotateVal = 0;
	rotateRequested = true;
}



// This is called by the camera to tell us we found a can
public void foundCan(boolean found)
{
	// This information will be used by the state machine if it cares
	hasCan = found;
}

// This is called by the touch sensors to let us know we hit a wall
public void foundWall()
{
	motors.stop();
	// Delay for a bit to allow PID, compass, and motors to catch up
	try { Thread.sleep(50); } catch (InterruptedException ie) { }
	// Reset the compass orientation no matter what state we are in
	compass.setNorth();
	// Also inform the state machine in case it decides to look for this event
	//   (this should be auto reset by the state machine, no need for main program to reset)
	onWall = true;
}

// called by the line sensors
public void passedLine()
{
	if (compass.getCurrHeading() < 90 || compass.getCurrHeading() > 270)
	{
		// If we are driving towards opponents endzone, increment linesPassed
		linesPassed++;
	}
	else
	{
		// If we are driving towards our endzone, decrement linesPassed instead
		linesPassed--;
	}
}

// called by the camera
public void updateCameraTargets(int redTarget, int blueTarget)
{
	cameraRedTarget = redTarget;
	cameraBlueTarget = blueTarget;
}

/* End state machine public helper methods */

// Stop everything
public void cancel()
{
```

```
		// Stop the motors and terminate the threads
		motors.stop();
		thread.cancel();
		watchdog.cancel();
}


public void restart()
{
		// Recreate the state machine and timer threads
		thread = new StateMachineThread(motors, compass, arm);
		thread.start();
		watchdog = new Timer("Watchdog");
		// For now go back to base on restart
		// state = STATE.GO_HOME;
}

public void beginAutonomous()
{
		compass.setNorth();
		// Give marvin 10 seconds to find a can...or else!
		//resetWatchdog(10000);
		beginCalled = true;
}

/**
 * Sets up a timer to force the state machine into the GO_HOME state if it
 *   isn't reset before a timeout occurs.
 *
 * @param millis The time (in ms) to wait until forcing a return to the STOP state.
 */
private void resetWatchdog(long millis)
{
		if (millis > 0)
		{
				// Cancel any previous tasks
				if (task != null)
				{
						task.cancel();
						watchdog.purge();
				}
				// Force the state to the STOP state if any state takes too long
				task = new TimerTask()
				{
						public void run()
						{
								arm.close_claw();
								Log.v("marvin", "Watchdog expired, moving to GO_HOME state");
								compass.lockTo(0);
								try { Thread.sleep(10); } catch (InterruptedException ie) { }
								state = STATE.GO_HOME;
						}
				};
				watchdog.schedule(task, millis);
		}
		else
		{
				// If millis is 0 or negative, cancel any previous task
				task.cancel();
				watchdog.purge();
		}
}

// Inner class to handle the actual state machine:
private class StateMachineThread extends Thread
{
```

```
private Motors motors;
private Compass compass;
private Arm arm;
public StateMachineThread(Motors motors, Compass compass, Arm arm)
{
	this.motors = motors;
	this.compass = compass;
	this.arm = arm;
}
// rotate
@Override
public void run()
{
	while (state != STATE.EXIT)
	{
		switch (state)
		{
		case IDLE:
			// Start the autonomous routines if beginAutonomous() was called
			if (beginCalled)
			{
				state = STATE.FIND_CAN;
				Log.d("marvin", "FIND_CAN");
				beginCalled = false;
			}
			// If rotateRequested was called (for example, from a button press), go to a special ROTATE_ONCE
state
			if (rotateRequested)
			{
				state = STATE.ROTATE_ONCE;
			}
			// Poll the sensors to aid in debugging process
			light.requestData((byte)0x1);
			light.requestData((byte)0x2);
			light.requestData((byte)0x3);
			touch.requestButtonData();
			break;
		case ROTATE_ONCE:
			// This state should just be used if the state should return to IDLE after finishing
			if (rotateTo(desiredRotateVal))
			{
				// We're done, go to STOP (which goes back to IDLE)
				motors.stop();
				state = STATE.FIND_CAN;
				desiredRotateVal = 0;
				rotateRequested = false;
			}
			break;
		case FIND_CAN:
			// Use the camera values to find a can
			// Avoid endzone
			// Drive randomly?
			// Rotate bot:
			if (oldCameraVal != cameraRedTarget)
			{
				Log.d("marvin", "camerRedTarget: "+cameraRedTarget);
				// Some magic camera numbers (around the middle of the screen)
				if (cameraRedTarget >= 900)
				{
					Log.d("marvin", "ROTATE RIGHT");
					//motors.rotate(1100 - cameraRedTarget / 2);
					motors.rotateRight();
				}
				else if (cameraRedTarget <= 600)
				{
					Log.d("marvin", "ROTATE LEFT");
```

```
                        //motors.rotate(cameraRedTarget / 2 - 900);
                        motors.rotateLeft();
                }
                else
                {
                        motors.stop();
                        // Delay for a bit to allow PID and motors to catch up before continuing
                        try { Thread.sleep(50); } catch (InterruptedException ie) { }
                        if (cameraRedTarget > 600 && cameraRedTarget < 900)
                        {
                                // We have indeed found a can, just drive forwards for a while
                                Log.v("marvin", "Found can, moving to GET_CAN state");
                                // Lock the compass to where we see the can:
                                compass.lock();
                                try { Thread.sleep(10); } catch (InterruptedException ie) { }
                                arm.open_claw();
                                // Give the bot 5 seconds to grab the can
                                resetWatchdog(3500);
                                hasCan = false;
                                //desiredRotateVal = compass.getReturnAngle();
                                state = STATE.GET_CAN;
                                Log.v("marvin", "GET_CAN");
                        }
                }
                oldCameraVal = cameraRedTarget;
        }
        break;
case GET_CAN:
        // Grab the error from the desired angle from the compass and pass it to the motors
        int result = compass.getReturnAngle();
        motors.trueForward(result);
        // Check light sensor up front for can
        light.requestData((byte)3);
        // Watchdog will timeout.
        if (hasCan)
        {
                hasCan = false;
                Log.v("marvin", "marvin hasCan");
                motors.stop();
                // reset watchdog (10 seconds to get home...or else!)
                //resetWatchdog(10000);
                // We see the can! Grab it!
                arm.close_claw();
                //arm.raise();
                // Clear watchdog
                //resetWatchdog(0);
                // face the opponents base
                compass.lockTo(0);
                //compass.lock();
                try { Thread.sleep(10); } catch (InterruptedException ie) { }
                state = STATE.GO_HOME;
                Log.v("marvin", "GO_HOME");
        }
        // Delay for a bit to allow PID and motors to catch up before continuing
        try { Thread.sleep(1); } catch (InterruptedException ie) { }
        break;
case GO_HOME:
        // Go back to starting side and hit the wall
        int bresult = compass.getReturnAngle();
        motors.trueBackward(bresult);
        touch.requestButtonData();
        if (onWall)
        {
                Log.v("marvin", "onWall");
                onWall = false;
                motors.stop();
```

```
                                        // drop the can
                                        arm.lower();
                                        state = STATE.DROP_CAN;
                                }

                                break;
                        case DROP_CAN:
                                motors.forward();
                                for (int i=0; i<25; i++)
                                        motors.rotateLeft();
                                arm.open_claw();
                                motors.back();
                                for (int i=0; i<10; i++)
                                        motors.rotateRight();
                                state = STATE.ROTATE_ONCE;
                                break;
                        } //end switch
                        adkActivity.currentState.post(new Runnable()
                        {
                                public void run()
                                {
                                        adkActivity.currentState.setText(state.name());
                                }
                        });
                } // end while
        }
        // Rotate the bot to a desired angle:
        public boolean rotateTo(int angle)
        {
                boolean atDesiredAngle = false;
                int compassResult = -1;
                Log.v("marvin", "rotateTo: "+angle);
                // Give the bot 6 seconds to turn.
                //resetWatchdog(6000);
                // THIS NEEDS A TIMEOUT!!!!
                // Tell the compass where we want to go
                compass.lockTo(angle);
                //while (!atDesiredAngle)
                //{
                        // Ask compass which way to turn
                        compassResult = compass.getReturnAngle();
                        // Rotate the motors in that direction
                        motors.rotate(compassResult);
                        Log.v("marvin", "rotating motors: "+compassResult);
                        //if (compassResult > 178 && compassResult < 182)
                        if (compassResult > -5 && compassResult < 5)
                        {
                                Log.v("marvin", "Found angle??");
                                motors.stop();
                                // Let all the sensors catch up
                                try { Thread.sleep(50); } catch (InterruptedException ie) { }
                                // Are we really facing the right direction?
                                compassResult = compass.getReturnAngle();
                                if (compassResult > -5 && compassResult < 5)
                                {
                                        Log.v("marvin", "Yes, we really found it. exit");
                                        compass.unlock();
                                        return true;
                                        //atDesiredAngle = true;
                                }
                        }
                        // Delay for a bit to allow PID, compass, and motors to catch up
                        try { Thread.sleep(1); } catch (InterruptedException ie) { }
                //}
                return false;
        }
```

```
		public void cancel()
		{
			state = STATE.EXIT;
		}
	}
}

// Arm

package edu.calpoly.android.adkmarvin;

import android.util.Log;
import android.widget.Toast;

public class Arm {

	private ADKMarvinActivity mActivity;
	private static final byte ARM_L = 16, // Servo 1 - PWM11 - purple
	                          ARM_R = 17, // Servo 2 - PWM12 - blue
	                          CLAW = 18,  // Servo 3 - PWM13 - yellow
	                          SERVO_COMMAND = 2,
	                          LOW = 2,
	                          HIGH = 3;
	private byte arm_pos = LOW;
	private static final int init_arm_l = 5;
	private static final int init_arm_r = 250;
	private static final int init_top = 90;
	private static final int stack_top = 45;

	public Arm(ADKMarvinActivity activity)
	{
		mActivity = activity;
	}

	public void lower()
	{
		int top = init_top;
		int lpos=init_arm_l+top, rpos=init_arm_r-top;
		if (arm_pos == HIGH)
		{
			for (int i=0; i<top; i+=2)
			{
				move_servo(ARM_L, lpos - i);
				move_servo(ARM_R, rpos + i);
			}
		}
		arm_pos = LOW;
	}

	public void stack()
	{
		lower();
		open_claw();
	}

	public void raise()
	{
		int lpos=init_arm_l, rpos=init_arm_r, top=init_top;
		if (arm_pos == LOW)
		{
			for (int i=0; i<top; i+=2)
			{
				move_servo(ARM_L, lpos + i);
				move_servo(ARM_R, rpos - i);
			}
		}
```

```
		arm_pos = HIGH;
	}

	public void open_claw()
	{
		move_servo(CLAW, 160);
	}

	public void close_claw()
	{
		for(int i=160; i>9; i-=3)
		{
			move_servo(CLAW, i);
		}
	}

	private void move_servo(byte which_servo, int pos)
	{
		mActivity.sendCommand(SERVO_COMMAND, which_servo, pos);
	}

}

package edu.calpoly.android.adkmarvin;

import android.content.Context;
import android.graphics.Path.Direction;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.util.Log;

public class Compass implements SensorEventListener
{
	private float[] accelerometer = new float[3];
	private float[] geomagnetic = new float[3];

	private int NORTH = 5000;
	private int lockOrientation = 5000;
	private int curOrientation = 5000;
	private Boolean setNorthBool = false;

	public int toNorthAngle = -1;

	// This is a debugging variable to automatically print a Toast message
	private Context tempContext;
	private ADKMarvinActivity adkActivity;

	private boolean unreliableWarn[] = { false, false };

	public Compass(Context context, ADKMarvinActivity activity)
	{
		//Log.d("marvin", "Create Compass Class");
		tempContext = context;
		adkActivity = activity;
	}

	public int getNorth()
	{
		return NORTH;
	}

	public void setNorth()
	{
		//NORTH = curOrientation;
```

```
		setNorthBool = true;
}

public int getCurrHeading()
{
		return curOrientation;
}

public void startCompass()
{
		/*SensorManager manager = (SensorManager)tempContext.getSystemService(Context.SENSOR_SERVICE);
		java.util.List<Sensor> sensors = manager.getSensorList(Sensor.TYPE_MAGNETIC_FIELD);
		if (sensors.size() > 0)
		{
			manager.registerListener(this, sensors.get(0), SensorManager.SENSOR_DELAY_NORMAL);
		}
		sensors = manager.getSensorList(Sensor.TYPE_ACCELEROMETER);
		if (sensors.size() > 0)
		{
			manager.registerListener(this, sensors.get(0), SensorManager.SENSOR_DELAY_NORMAL);
		}*/
		SensorManager sensorManager = (SensorManager)adkActivity.getSystemService(Context.SENSOR_SERVICE);
		Sensor sensor = sensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION);
		sensorManager.registerListener(this, sensor, SensorManager.SENSOR_DELAY_GAME);
		setNorthBool = true;
}

public void stopCompass(Context context)
{
		//Stop updating the information if this activity is no longer in use.
		SensorManager manager = (SensorManager)context.getSystemService(Context.SENSOR_SERVICE);
		manager.unregisterListener(this);
		curOrientation = 5000;
		unlock();
}

public void lock()
{
		lockOrientation = curOrientation;
}

public void lockTo(int newOrientation)
{
		lockOrientation = newOrientation;
}

//TODO: Make this return a value from -180 to 180 to simplify the PID controller
public int getReturnAngle()
{
		if (lockOrientation != 5000)
		{
			int angleDif = curOrientation - lockOrientation;
			if (angleDif > 180)
				return angleDif - 360;
			if (angleDif < -180)
				return angleDif + 360;
			//if (angleDif < 0)
			//	return angleDif + 360;
			//Log.d("marvin", "angleDif "+angleDif);
			return angleDif;
		}
		// Warn user but only if compass isn't turned off
		//if (curOrientation != 5000)
			//Log.e("Compass", "Lock must be called before getReturnAngle()!");
		return -1;
}
```

```
	public void unlock()
	{
		lockOrientation = 5000;
	}

	@Override
	public void onSensorChanged(SensorEvent event)
	{
		curOrientation = (int)event.values[0];
		if (setNorthBool)
		{
			NORTH = curOrientation;
			Log.d("marvin", "NORTH FOUND!: "+NORTH);
			adkActivity.tv_northAngle.setText("" + NORTH);
			setNorthBool = false;
		}
		else if (NORTH != 5000)
		{
			//int result = curOrientation - NORTH;
			curOrientation -= NORTH;
			if (curOrientation < 0)
				curOrientation += 360;
			//Log.d("marvin", "curOrientation: "+result);
			adkActivity.tv_currentAngle.setText("" + curOrientation);
			//adkActivity.tv_returnAngle.setText("" + result);
		}
		/*if (NORTH != 5000)
		{
			// Interpret the sensor values
			curOrientation = (int)Math.toDegrees(actualOrientation[0]) - NORTH;
			if (curOrientation < 0)
				curOrientation += 360;
			if (setNorthBool)
			{
				NORTH = (int)Math.toDegrees(actualOrientation[0]);
				adkActivity.tv_northAngle.setText("" + NORTH);
				setNorthBool = false;
			}
			//Log.d("marvin", "curOrientation: "+curOrientation);
			adkActivity.tv_currentAngle.setText("" + curOrientation);
			adkActivity.tv_returnAngle.setText("" + (int)Math.toDegrees(actualOrientation[0]));
		}
		else if (NORTH == 5000)
		{
			NORTH = (int)Math.toDegrees(actualOrientation[0]);
			Log.d("marvin", "NORTH FOUND!: "+NORTH);
			adkActivity.tv_northAngle.setText("" + NORTH);
		}*/

	}

	@Override
	public void onAccuracyChanged(Sensor sensor, int accuracy) { }
}

package edu.calpoly.android.adkmarvin;

import java.util.Timer;
import java.util.TimerTask;

import android.content.Intent;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;
import android.widget.Toast;
```

```
public class LightSensor
{
   private ADKMarvinActivity mActivity;
   private byte[] states = new byte[3];
   private Timer syncTimer = new Timer("SyncTimer");
   private boolean expired = true;
   private int orientation = 2;

   public static final String INTENT_ENDZONE = "edu.calpoly.android.adkmarvin.iEndzone";
   public static final String INTENT_FOUND_CAN = "edu.calpoly.android.adkmarvin.iFoundCan";

   // Debugging variable
   private int lines_passed = 0;

   public LightSensor(ADKMarvinActivity activity)
   {
            mActivity = activity;
   }

   public void orientationUpdated(int orientation)
   {
            if (orientation == 2)
                     this.orientation = 2;
            else if (orientation == 3)
                     this.orientation = 3;
            else
                     this.orientation = 0;
   }

   public void requestData(byte sensor)
   {
            mActivity.sendCommand((byte)0x3, sensor, 0);
   }

   public byte[] handleLightSensorMessage(ADKMarvinActivity.LightMsg l, StateMachine m)
   {
            //String msg = ""+l.getState();
            //Log.v("marvin", msg);
            if (l.getSw() == 2)
            {
                     states[2] = l.getState();
                     // Check can sensor values and update the state machine
                     if (states[2] == 0)
                     {
                              m.foundCan(true);
                              /*
                              Intent i = new Intent();
                              i.setAction(INTENT_FOUND_CAN);
                              LocalBroadcastManager.getInstance(mActivity).sendBroadcast(i);
                              */
                     }
                     else
                     {
                              m.foundCan(false);
                     }
            }
            else
            {
                     states[l.getSw()] = l.getState();
                     //if (l.getState() == 1)
                     //{
                              if (states[0] == 1 && states[1] == 1)
                              {
                                       Log.v("marvin", "passed a line");
                                       m.passedLine();
                                       /*
```

```
                                        Intent i = new Intent();
                                        i.setAction(INTENT_ENDZONE);
                                        LocalBroadcastManager.getInstance(mActivity).sendBroadcast(i);
                                        */
                                }
                        //}
                }
                return states;

                /*
                if (l.getState() == 1)
                {
                        // If a black line is detected on one of the sensors
                        if (expired)
                        {
                                // Start a timer to count 50 milliseconds
                                expired = false;
                                states[0] = l.getSw();
                                syncTimer.schedule(new TimerTask()
                                {
                                        public void run()
                                        {
                                                expired = true;
                                        }
                                }, 50);
                        }
                        else
                        {
                                // Check if the other sensor was the one tripped
                                if (l.getSw() != states[0])
                                {
                                        expired = true;
                                        if (orientation == 2)
                                                lines_passed++;
                                        else if (orientation == 3)
                                                lines_passed--;
                                        if (lines_passed <= 0)
                                        {
                                                Intent i = new Intent();
                                                i.setAction(INTENT_ENDZONE);
                                                LocalBroadcastManager.getInstance(mActivity).sendBroadcast(i);
                                                lines_passed = 0;
                                        }
                                }
                        }
                }
                */
        }
}

package edu.calpoly.android.adkmarvin;

import android.app.Service;
import android.content.Intent;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.support.v4.app.ServiceCompat;
import android.support.v4.content.LocalBroadcastManager;

public class LocalService extends Service { //static?
        LocalBroadcastManager mLocalBroadcastManager;
        int mCurUpdate;

        public static final String CUSTOM_INTENT = "edu.calpoly.android.adkmarvin.breceive";
```

```
        static final int MSG_UPDATE = 1;

        Handler mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
        switch (msg.what) {
        case MSG_UPDATE: {
        mCurUpdate++;
        Intent intent = new Intent(CUSTOM_INTENT);
        intent.putExtra("value", mCurUpdate);
        mLocalBroadcastManager.sendBroadcast(intent);
        Message nmsg = mHandler.obtainMessage(MSG_UPDATE);
        mHandler.sendMessageDelayed(nmsg, 1000);
        } break;
        default:
        super.handleMessage(msg);
        }
        }
        };

        @Override
        public void onCreate() {
        super.onCreate();
        mLocalBroadcastManager = LocalBroadcastManager.getInstance(this);
        }

        public int onStartCommand(Intent intent, int flags, int startId) {
        // Tell any local interested parties about the start.
        mLocalBroadcastManager.sendBroadcast(new Intent(CUSTOM_INTENT));

        // Prepare to do update reports.
        mHandler.removeMessages(MSG_UPDATE);
        Message msg = mHandler.obtainMessage(MSG_UPDATE);
        mHandler.sendMessageDelayed(msg, 1000);
        return ServiceCompat.START_STICKY;
        }

        @Override
        public void onDestroy() {
        super.onDestroy();

        // Tell any local interested parties about the stop.
        mLocalBroadcastManager.sendBroadcast(new Intent(CUSTOM_INTENT));

        // Stop doing updates.
        mHandler.removeMessages(MSG_UPDATE);
        }

        @Override
        public IBinder onBind(Intent intent) {
        return null;
        }
}

package edu.calpoly.android.adkmarvin;

import android.util.Log;

public class Motors {

   // Motor wires! Red = LEFT
   // Green = Right!

   private ADKMarvinActivity mActivity;
   private static final byte //MOTOR_L = 15, // Servo 1 - PWM10 - white
                                                                        //MOTOR_R = 14, // Servo 2 - PWM9 - orange
```

```
                                                    MOTOR_L = 0x1,
                                                    MOTOR_R = 0x2,
                                                    SERVO_COMMAND = 2;

public Motors(ADKMarvinActivity activity)
{
        mActivity = activity;
}

/* Smoothing PID control */

// PID controller found from
        //   http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/
private long lastTime;
private double errSum, lastErr;
private static final double kp = 0.02, ki = 0.02, kd = 0.01;

public double computePower(double error)
{
        // How long since we last calculated
        long now = System.currentTimeMillis();
        double timeChange = now - lastTime;

        // Compute all the working error variables
        //double error = targetAngle - currentAngle;
        errSum += (error * timeChange);

        // Keep errSum from getting too large
        if (errSum > 3000)
                errSum = 3000;
        else if (errSum < -3000)
                errSum = -3000;

        double dErr = (error - lastErr) / timeChange;

        // Compute PID Output
        double Output = kp * error + ki * errSum + kd * dErr;

        // Prevent the Output from moving too fast
        if (Output > 1000)
                Output = 1000;
        if (Output < -1000)
                Output = -1000;

        // Remember some variables for next time
        lastErr = error;
        lastTime = now;

        return Output / 100;
}

/* End Smoothing PID control */

byte tempResult;

public void trueForward(int error)
{
        /*byte result = (byte)computePower(error);
        tempResult = result;
        //Log.d("marvin", "L "+(15 - result)+" R "+(-15 - result));
        mActivity.tv_returnAngle.post(new Runnable()
        {
                public void run()
                {
                        mActivity.tv_returnAngle.setText("Result: "+tempResult+"          ");
                }
```

```
		});*/
		Log.d("marvin", "trueforward: "+(error));
		if (error > 8)
			error = 8;
		else if (error < -8)
			error = -8;

		move_servo(MOTOR_L, 128 + 15 + error);
		move_servo(MOTOR_R, 128 -17  + error);
		delay(50);
}

public void trueBackward(int error)
{
		/*byte result = (byte)computePower(error);
		tempResult = result;
		//Log.d("marvin", "L "+(15 - result)+" R "+(-15 - result));
		mActivity.tv_returnAngle.post(new Runnable()
		{
			public void run()
			{
				mActivity.tv_returnAngle.setText("Result: "+tempResult+"            ");
			}
		});*/
		Log.d("marvin", "truebackward "+(error));
		if (error > 8)
			error = 8;
		else if (error < -8)
			error = -8;

		move_servo(MOTOR_L, 128 - 15 + error);
		move_servo(MOTOR_R, 128 + 17 + error);
		delay(50);
}

public void rotate(int error)
{
		//byte result = (byte)computePower(error);
		//tempResult = result;
		//Log.d("marvin", "L "+(15 - result)+" R "+(-15 - result));
		/*mActivity.tv_returnAngle.post(new Runnable()
		{
			public void run()
			{
				mActivity.tv_returnAngle.setText("" + tempResult);
			}
		});*/
		Log.d("marvin", "move_servo: "+(128+error));
		if (error > 10)
			error = 10;
		else if (error < -10)
			error = -10;
		move_servo(MOTOR_L, 128 + error);
		move_servo(MOTOR_R, 128 + error);
		delay(100);
}

/* Individual hard-coded motor directions */

public void stop()
{
		move_servo(MOTOR_L, 128); //128
		move_servo(MOTOR_R, 128); //128
		mActivity.tv_returnAngle.post(new Runnable()
		{
			public void run()
```

```
                {
                        mActivity.tv_returnAngle.setText("xx");
                }
        });
}

public void forward()
{
        //stop();
        move_servo(MOTOR_L, 128 + 20);
        move_servo(MOTOR_R, 128 - 20);
        delay(1000);
        stop();
}

public void back()
{
        //stop();
        move_servo(MOTOR_L, 128 - 30);
        move_servo(MOTOR_R, 128 + 29);
        delay(1500);
        stop();
}

public void rotateRight()
{
        //stop();
        move_servo(MOTOR_L, 128 - 15);
        move_servo(MOTOR_R, 128 - 15);
        delay(50);
        stop();
        //delay(75);
}

public void rotateLeft()
{
        //stop();
        move_servo(MOTOR_L, 128 + 15);
        move_servo(MOTOR_R, 128 + 15);
        delay(50);
        stop();
        //delay(75);
}

public void driftRight()
{
        stop();
        move_servo(MOTOR_L, 128 + 33);
        move_servo(MOTOR_R, 128 - 27);
}

public void driftLeft()
{
        move_servo(MOTOR_L, 128 + 27);
        move_servo(MOTOR_R, 128 - 33);
}

public void delay(long ms)
{
        // Drive for milliseconds
        try {
                Thread.sleep(ms);
        } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
```

```
}

private void move_servo(byte which_servo, int pos)
{
	mActivity.sendCommand(SERVO_COMMAND, which_servo, pos);
}


public void hforward()
{
	move_servo(MOTOR_L, 128 + 21);
	move_servo(MOTOR_R, 128 - 20);
	delay(4500);
	stop();
}

public void fcforward()
{
	move_servo(MOTOR_L, 128 + 20);
	move_servo(MOTOR_R, 128 - 20);
	delay(3200);
	stop();
}

public void stackforward()
{
	move_servo(MOTOR_L, 128 + 11);
	move_servo(MOTOR_R, 128 - 10);
	delay(1200);
	stop();
}

public void left90()
{
	move_servo(MOTOR_L, 128 + 10);
	move_servo(MOTOR_R, 128 + 10);
	delay(2000);
	stop();
}

public void scootchback()
{
	move_servo(MOTOR_L, 128 - 14);
	move_servo(MOTOR_R, 128 + 16);
	delay(1500);
	stop();
}

public void right90()
{
	move_servo(MOTOR_L, 128 - 20);
	move_servo(MOTOR_R, 128 - 20);
	delay(900);
	stop();
}

public void right180()
{
	move_servo(MOTOR_L, 128 - 10);
	move_servo(MOTOR_R, 128 - 10);
	delay(3400);
	stop();
}

public void wallforward()
{
```

```
		move_servo(MOTOR_L, 128 + 16);
		move_servo(MOTOR_R, 128 - 15);
		delay(1000);
		stop();
	}

	public void scootchforward()
	{
		move_servo(MOTOR_L, 128 + 16);
		move_servo(MOTOR_R, 128 - 15);
		delay(1000);
		stop();
	}

	public void center()
	{
		move_servo(MOTOR_L, 128 + 19);
		move_servo(MOTOR_R, 128 - 20);
		delay(4000);
		stop();
	}

	/* End individual motor directions */
}

package edu.calpoly.android.adkmarvin;

import java.util.ArrayList;

public class DecisionMaker
{
	public static enum Decision { GO_LEFT, GO_RIGHT };
	private static enum State { INIT, SEARCH, RETURN };
	private static State curState = State.INIT;
	private static int phase = 0;

	private static ArrayList<TrackedObjectList.TrackedObject> objectList;
	private static int compassVal;

	public static void updateObjects(ArrayList<TrackedObjectList.TrackedObject> objects)
	{
		objectList = objects;
	}

	public static void updateCompass(int value)
	{
		compassVal = value;
	}

	public static Decision makeDecision()
	{
		Decision curDecision = Decision.GO_LEFT;
		for (TrackedObjectList.TrackedObject obj : objectList)
		{
			if (obj instanceof TrackedObjectList.RedCan)
			{
				if (obj.getDimensions().centerX() < 600)
				{
					curDecision = Decision.GO_LEFT;
				}
				else
				{
					curDecision = Decision.GO_RIGHT;
				}
			}
			else if (phase == 1)
```

```
                {
                        // Give priority to blue cans in second phase of competition
                        if (obj instanceof TrackedObjectList.BlueCan)
                        {
                                if (obj.getDimensions().centerX() < 600)
                                {
                                        curDecision = Decision.GO_LEFT;
                                }
                                else
                                {
                                        curDecision = Decision.GO_RIGHT;
                                }
                        }
                }
        }
        return curDecision;
  }
}

package edu.calpoly.android.adkmarvin;

import java.io.IOException;
import java.util.List;

import android.content.Context;
import android.hardware.Camera;
import android.hardware.Camera.PreviewCallback;
import android.util.AttributeSet;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class PreviewSurface extends SurfaceView implements SurfaceHolder.Callback
{
  private static final String TAG = "OpenCV";
  private Camera mCamera;
  private int mFrameWidth;
  private int mFrameHeight;

  public PreviewSurface(Context context, AttributeSet attr)
  {
        super(context, attr);
        if (!isInEditMode())
        {
                SurfaceHolder fHolder = getHolder();
                fHolder.addCallback(this);
                fHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
        }
  }

        public int getFrameWidth()
        {
        return mFrameWidth;
        }

        public int getFrameHeight()
        {
        return mFrameHeight;
        }

        public void setPreviewCallback(PreviewCallback callback)
        {
         mCamera.setPreviewCallback(callback);
        }

  public void surfaceChanged(SurfaceHolder holder, int format, int width, int height)
```

```
{
        if (!isInEditMode())
        {
        Log.i(TAG, "PreviewSurface - surfaceChanged");
        if (mCamera != null)
        {
        Camera.Parameters params = mCamera.getParameters();
        List<Camera.Size> sizes = params.getSupportedPreviewSizes();
        mFrameWidth = width;
        mFrameHeight = height;

        // selecting optimal camera preview size
        {
        double minDiff = Double.MAX_VALUE;
        for (Camera.Size size : sizes)
        {
                if (Math.abs(size.height - height) < minDiff)
                {
                mFrameWidth = size.width;
                mFrameHeight = size.height;
                minDiff = Math.abs(size.height - height);
                }
        }
        }

        params.setPreviewSize(mFrameWidth, mFrameHeight);
        mCamera.setParameters(params);
        try
        {
                mCamera.setPreviewDisplay(holder);
                        }
        catch (IOException e)
                        {
                                Log.e(TAG, "mCamera.setPreviewDisplay fails: " + e);
                        }
        mCamera.startPreview();
        }
        }
}

public void surfaceCreated(SurfaceHolder holder)
{
        if (!isInEditMode())
        {
        Log.i(TAG, "PreviewSurface - surfaceCreated");
        mCamera = Camera.open();
        if (mCamera != null)
        {
                mCamera.setPreviewCallback(new PreviewCallback()
                {
                        public void onPreviewFrame(byte[] data, Camera camera)
                        {
                                // Keep trying to send the frame data to CameraDataBase, even while unavailable.
                                CameraDataBase view = (CameraDataBase)
(getRootView().findViewById(R.id.cameraView));
                                if (view != null)
                                        view.onPreviewFrame(data, mFrameWidth, mFrameHeight, camera);
                        }
                });
        }
        }
}

public void surfaceDestroyed(SurfaceHolder holder)
{
        if (!isInEditMode())
```

```
                {
                Log.i(TAG, "surfaceDestroyed");
                if (mCamera != null)
                {
                synchronized (this)
                {
                mCamera.stopPreview();
                mCamera.setPreviewCallback(null);
                mCamera.release();
                mCamera = null;
                }
                }
                }
    }
}

package edu.calpoly.android.adkmarvin;


import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.Window;

public class Sample1Java extends Activity
{
                private static final String TAG = "Sample::Activity";

                private MenuItem mItemPreferences;

                private static Context appContext;
                //Context appContext;
                public static final String CUSTOM_INTENT = "edu.calpoly.android.adkmarvin.breceive";

                public Sample1Java()
                {
                Log.i("marvin", "Instantiated new " + this.getClass());
                }

                /** Called when the activity is first created. */
                @Override
                public void onCreate(Bundle savedInstanceState)
                {
                Log.i(TAG, "onCreate");
                super.onCreate(savedInstanceState);
                appContext = getApplicationContext();
                Log.v("marvin", "init opencv br and send:");

                 requestWindowFeature(Window.FEATURE_NO_TITLE);
                setContentView(R.layout.opencv_main);

                }

                @Override
                public boolean onCreateOptionsMenu(Menu menu)
                {
                Log.i(TAG, "onCreateOptionsMenu");
                mItemPreferences = menu.add("Settings");
                return true;
                }
```

```
	@Override
	public boolean onOptionsItemSelected(MenuItem item)
	{
	Log.i(TAG, "Menu Item selected " + item);
	if (item == mItemPreferences)
		Log.i(TAG, "Preferences window selected");
	return true;
	}

	public void test()
	{
	 Log.v("marvin", "test");
	}
}

package edu.calpoly.android.adkmarvin;

import java.io.IOException;
import java.io.OutputStream;

/**
 * Spams a serial port with useless information to keep reads() on
 *   an Arduino from blocking too long.
 */
public class SerialKeepAlive extends Thread
{
   private OutputStream outputStream;
   private byte[] dummy = { (byte)0xFF, 0, 0 };

   public SerialKeepAlive(OutputStream mOutputStream)
   {
	outputStream = mOutputStream;
   }

   @Override
   public void run()
   {
	while (true)
	{
		if (outputStream != null)
		{
			try
			{
				synchronized (outputStream)
				{
					outputStream.write(dummy);
				}
				try { Thread.sleep(10); } catch (InterruptedException ie) { }
			}
			catch (IOException e)
			{
				android.util.Log.e("marvin", "no longer keeping serial port alive");
				break;
			}
		}
	}
   }
}

package edu.calpoly.android.adkmarvin;

import java.util.ArrayList;

import android.content.Context;
import android.content.Intent;
import android.graphics.Point;
```

```
import android.graphics.Rect;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;

public class TrackedObjectList
{
   private static final int STEP = 50;

   private static final int SINGLE_OBJECT = 0;
   private static final int SINGLE_OBJECT_WEIGHTED = 1;
   private static final int MULTI_OBJECT = 2;
   private static final int MULTI_OBJECT_REC = 3;
   private static final int TRACKING_TYPE = SINGLE_OBJECT;

   private ArrayList<RedCan> redcans;
   private ArrayList<BlueCan> bluecans;

   public static final String INTENT_CAMERA_VALUE = "edu.calpoly.android.adkmarvin.cvalue";

   Context myContext;

   // A "Red" pixel should have a red value > RED[0],
   //   green value < RED[1], and blue value < RED[2]
   private static final int[] RED = { 50, 25, 25 };
   // A "Blue" pixel should have a red value < BLUE[0],
   //   green value < BLUE[1], and blue value > BLUE[2]
   private static final int[] BLUE = { 25, 25, 30 };

   public abstract class TrackedObject
   {
            public abstract Rect getDimensions();
   }

   public class BlueCan extends TrackedObject
   {
            private Rect objSize;
            public BlueCan(Rect bounds)
            {
                     objSize = bounds;
            }
            public Rect getDimensions()
            {
                     return objSize;
            }
   }

   public class RedCan extends TrackedObject
   {
            private Rect objSize;
            private Rect maxSize;
            public RedCan(Rect bounds)
            {
                     objSize = bounds;
            }
            public RedCan(byte[] frame, boolean[][] processed, Point initialPoint, Rect maxSize)
            {
                     this.maxSize = maxSize;
                     objSize = new Rect(initialPoint.x, initialPoint.y, 1, 1);
                     findRedCan(frame, processed, initialPoint.x, initialPoint.y, objSize);
            }
            public Rect getDimensions()
            {
                     return objSize;
            }
            private boolean findRedCan(byte[] frame, boolean[][] processed, int x, int y, Rect objSize)
            {
```

```
                int n = 0;
                if (x < maxSize.right && x >= maxSize.left)
                {
                        if (y < maxSize.bottom && y >= maxSize.top)
                        {
                                if (!processed[x][y])
                                {
                                        processed[x][y] = true;
                                        n = y * maxSize.width() + x;
                                        // Check if red pixel
                                        if (frame[n] > RED[0])
                                        {
                                                if (frame[n + 1] >= RED[1])
                                                        return false;
                                                if (frame[n + 2] >= RED[2])
                                                        return false;
                                        }
                                        /*
                                        // Check if blue pixel
                                        if (frame[n + 2] > BLUE[2])
                                        {
                                                if (frame[n] >= BLUE[0])
                                                        return false;
                                                if (frame[n] >= BLUE[1])
                                                        return false;
                                        }
                                        */
                                        // Recursively call this function on each neigboring pixel, modifying
                                        //   objSize each time
                                        if (findRedCan(frame, processed, x + STEP, y, objSize))
                                                objSize.union(x + STEP, y);
                                        if (findRedCan(frame, processed, x - STEP, y, objSize))
                                                objSize.union(x - STEP, y);
                                        if (findRedCan(frame, processed, x, y + STEP, objSize))
                                                objSize.union(x, y + STEP);
                                        if (findRedCan(frame, processed, x, y - STEP, objSize))
                                                objSize.union(x, y - STEP);
                                        return true;
                                }
                        }
                }
                return false;
        }
}

public TrackedObjectList(Context context)
{
        myContext = context;
        redcans = new ArrayList<RedCan>();
        bluecans = new ArrayList<BlueCan>();
        //stateMachine = tempstateMachine;
        if (TRACKING_TYPE == SINGLE_OBJECT || TRACKING_TYPE == SINGLE_OBJECT_WEIGHTED)
        {
                // These modes never add new objects, so initialize them here
                redcans.add(new RedCan(new Rect(0, 0, 10, 10)));
                bluecans.add(new BlueCan(new Rect(0, 0, 10, 10)));
        }
}

// Right now, this just returns a list of all red cans tracked
public ArrayList<TrackedObject> getTrackedObjects()
{
        ArrayList<TrackedObject> list = new ArrayList<TrackedObject>();
        for (RedCan rcan : redcans)
                list.add(rcan);
        //for (BlueCan bcan : bluecans)
```

```
                //list.add(bcan);
        return list;
}

public int findObjects(byte[] frame, int width, int height, int pixelSize)
{
        switch (TRACKING_TYPE)
        {
        case SINGLE_OBJECT:
                return findSingleObject(frame, width, height, pixelSize);
        case SINGLE_OBJECT_WEIGHTED:
                break;
        case MULTI_OBJECT:
                return findMultiObjects(frame, width, height, pixelSize);
        case MULTI_OBJECT_REC:
                return findMultiObjectsRec(frame, width, height, pixelSize);
        }
        throw new IllegalArgumentException("TRACKING_TYPE constant not valid");
}

private int findSingleObject(byte[] frame, int width, int height, int pixelSize)
{
        // RGB conversion variables
        int r, g, b, n;
        // Red can variables
        long r_avg = 0;
        long r_totAvg = 0;
        int r_xSamples = 0;
        int r_ySamples = 0;
        // Blue can variables
        long b_avg = 0;
        long b_totAvg = 0;
        int b_xSamples = 0;
        int b_ySamples = 0;
        for (int y = 0; y < height; y++)
        {
                for (int x = 0; x < width; x++)
                {
                        // Cast as unsigned bytes (which isn't possible in java, so just use an int)
                        n = (y * width + x) * pixelSize;
                        r = frame[n] & 0xFF;
                        g = frame[n + 1] & 0xFF;
                        b = frame[n + 2] & 0xFF;
                        if (r > RED[0] && g < RED[1] && b < RED[2])
                        {
                                // Update the red row average numerator
                                r_avg += x;
                                r_xSamples++;
                        }
                        else if (r < BLUE[0] && g < BLUE[1] && b > BLUE[2])
                        {
                                // Update the blue row average numerator
                                b_avg += x;
                                b_xSamples++;
                        }
                }
                if (r_avg > 0)
                {
                        // Calculate the current total red can average
                        r_totAvg += r_avg / r_xSamples;
                        r_ySamples++;
                }
                if (b_avg > 0)
                {
                        // Calculate the current total blue can average
                        b_totAvg += b_avg / b_xSamples;
```

```
                        b_ySamples++;
                }
                // Reset before moving to the next row
                r_avg = 0;
                r_xSamples = 0;
                b_avg = 0;
                b_xSamples = 0;
        }
        // Get the average of the row averages
        if (r_ySamples > 0)
                r_totAvg /= r_ySamples;
        if (b_ySamples > 0)
                b_totAvg /= b_ySamples;
        // In this mode, redcans (and bluecans) only have one element
        Rect temp = redcans.get(0).getDimensions();
        temp.set((int)r_totAvg - 5, height / 2 - 5,
                        (int)r_totAvg + 5, height / 2 + 5);
        temp = bluecans.get(0).getDimensions();
        temp.set((int)b_totAvg - 5, height / 2 - 5,
                        (int)b_totAvg + 5, height / 2 + 5);
        // Broadcast the r_totAvg to the adk activity
        //String msg = "trackedobjlist: " + r_totAvg;
        //Log.v("marvin", msg);
        // Send a broadcast to ADKMarvinActivity to turn on the led.
        //stateMachine.redVal = r_totAvg;
        Intent i = new Intent(INTENT_CAMERA_VALUE);
        i.putExtra("value", r_totAvg);
        LocalBroadcastManager.getInstance(myContext).sendBroadcast(i);
        return 2;
}
private int findMultiObjects(byte[] frame, int width, int height, int pixelSize)
{
        ArrayList<Rect> rects = new ArrayList<Rect>();
        Rect temp = new Rect();
        boolean connected;
        int n = 0;
        // Create rectangles for the first row
        for (int y = 0; y < height; y++)
        {
                for (int x = 0; x < width; x++)
                {
                        n = (y * width + x) * pixelSize;
                        if (frame[n] > RED[0] && frame[n + 1] < RED[1] && frame[n + 2] < RED[2])
                        {
                                // Update temp to be a 2x2 rectangle covering this point
                                //   as well as potential past neighbors
                                temp.left = x - 1;
                                temp.top = y;
                                temp.right = x;
                                temp.bottom = y - 1;
                                connected = false;
                                for (Rect r : rects)
                                {
                                        if (Rect.intersects(r, temp))
                                        {
                                                r.union(temp);
                                                connected = true;
                                        }
                                }
                                if (!connected)
                                        rects.add(temp);
                        }
                }
        }
        redcans.clear();
        for (Rect r : rects)
```

```
            {
                redcans.add(new RedCan(r));
            }
            return rects.size();
    }

    private int findMultiObjectsRec(byte[] frame, int width, int height, int pixelSize)
    {
            boolean[][] found = new boolean[width][height];
            int n, objectsFound = 0;
            for (int y = 0; y < height; y++)
            {
                for (int x = 0; x < width; x++)
                {
                    n = (y * width + x) * pixelSize;
                    if (frame[n] > RED[0] && frame[n + 1] < RED[1] && frame[n + 2] < RED[2])
                    {
                        redcans.add(new RedCan(frame, found,
                                new Point(x, y), new Rect(0, 0, width, height)));
                        objectsFound++;
                    }
                }
            }
            return objectsFound;
    }
}

package edu.calpoly.android.adkmarvin;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;

public class AccessoryAttached extends Activity{

    @Override
    protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            Intent intent = new Intent(AccessoryAttached.this, ADKMarvinActivity.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK
                    | Intent.FLAG_ACTIVITY_CLEAR_TOP);
            try {
                startActivity(intent);
            } catch (ActivityNotFoundException e) {
                Log.e("ADKMARVIN", "unable to start ADKMarvin activity", e);
            }
            finish();
    }

}

// ui:
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="horizontal" >

            <edu.calpoly.android.adkmarvin.CameraData
            android:id="@+id/cameraView"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />

            <edu.calpoly.android.adkmarvin.PreviewSurface
```

```
android:id="@+id/fakeCameraView"
android:layout_width="fill_parent"
android:layout_height="fill_parent" />

<LinearLayout
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="horizontal" >

<ScrollView
android:id="@+id/scrollView1"
android:layout_width="wrap_content"
android:layout_height="fill_parent"
android:background="#30000000" >

<LinearLayout
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:orientation="vertical"
android:paddingLeft="5dip"
android:paddingRight="5dip" >

<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:gravity="center"
android:text="Autonomous Commands"
android:textColor="#FFFFFFFF" />

<Button
android:id="@+id/setNorth"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Set North" />

<Button
android:id="@+id/findNorth"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Find North" />

<Button
android:id="@+id/findCan"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:padding="10dip"
android:text="Find Can" />

<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:gravity="center"
android:text="Servo Commands"
android:textColor="#FFFFFFFF" />

<Button
android:id="@+id/openclaw"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Open Claw" />

<Button
android:id="@+id/closeclaw"
```

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Close Claw" />

<Button
android:id="@+id/raisearm"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Raise Arm" />

<Button
android:id="@+id/lowerarm"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Lower Arm" />

<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:gravity="center"
android:text="Drive commands"
android:textColor="#FFFFFFFF" />

<Button
android:id="@+id/stop"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="STOP" />

<Button
android:id="@+id/forward"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Forward" />

<Button
android:id="@+id/backward"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Back" />

<Button
android:id="@+id/right"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Right" />

<Button
android:id="@+id/left"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Left" />
</LinearLayout>
</ScrollView>

<!--
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
android:layout_width="fill_parent"
android:layout_height="wrap_content" >

<Button
android:id="@+id/openclaw"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Open Claw" />

<Button
android:id="@+id/closeclaw"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Close Claw" />

<Button
android:id="@+id/raisearm"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Raise Arm" />

<Button
android:id="@+id/lowerarm"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Lower Arm" />
</LinearLayout>

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="wrap_content" >

<Button
android:id="@+id/stop"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="STOP" />

<Button
android:id="@+id/forward"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Forward" />

<Button
android:id="@+id/backward"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Back" />

<Button
android:id="@+id/right"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Right" />

<Button
```

```
android:id="@+id/left"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Left" />
</LinearLayout>

<TextView
android:id="@+id/lightValues"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="TextView" />

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="wrap_content" >

<Button
android:id="@+id/setNorth"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Set North" />

<Button
android:id="@+id/findNorth"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="10dp"
android:text="Find North" />

<Button
android:id="@+id/findCan"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="10dip"
android:text="Find Can" />
</LinearLayout>
-->

<LinearLayout
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:gravity="right" >

<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="wrap_content"
android:layout_height="fill_parent"
android:background="#30000000"
android:gravity="center"
android:orientation="vertical"
android:paddingLeft="10dip"
android:paddingRight="10dip" >

<TableRow
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:paddingBottom="10dip" >

<TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="ADK Status" />
```

```
<TextView
            android:id="@+id/adkstatus"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="right"
            android:text="Disconnected"
            android:textColor="#FFFFFFFF" />
</TableRow>

<TableRow
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:paddingBottom="10dip" >

<TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Light Sensors" />

<TextView
            android:id="@+id/lightValues"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="right"
            android:text="- - / -"
            android:textColor="#FFFFFFFF" />
</TableRow>

<TableRow
android:layout_width="match_parent"
android:layout_height="wrap_content" >

<TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Offset" />

<TextView
            android:id="@+id/returnAngle"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="right"
            android:text="xx"
            android:textColor="#FFFFFFFF" />
</TableRow>

<TableRow
android:layout_width="match_parent"
android:layout_height="wrap_content" >

<TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="North" />

<TextView
            android:id="@+id/northAngle"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="right"
            android:text="xx"
            android:textColor="#FFFFFFFF" />
</TableRow>

<TableRow
android:layout_width="match_parent"
```

```
android:layout_height="wrap_content"
android:paddingBottom="10dip" >

<TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Current" />

<TextView
        android:id="@+id/currentAngle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="right"
        android:text="xx"
        android:textColor="#FFFFFFFF" />
</TableRow>

<TableRow
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:paddingBottom="10dip" >

<TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Camera Avg" />

<TextView
        android:id="@+id/cameraValue"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="right"
        android:text="xx"
        android:textColor="#FFFFFFFF" />
</TableRow>
<TableRow
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:paddingBottom="10dip" >

<TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="State" />

<TextView
        android:id="@+id/currentState"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="right"
        android:text="EXIT"
        android:textColor="#FFFFFFFF" />
</TableRow>
</TableLayout>
</LinearLayout>
</LinearLayout>

<!--
Required for the camera to give any data back

<edu.calpoly.android.adkmarvin.PreviewSurface
android:id="@+id/fakeCameraView"
android:layout_width="fill_parent"
android:layout_height="fill_parent" />
-->
```

```
</FrameLayout>

// Libraries:
// Google APIs (Android 4.0)
// android-support-v4.jar
// OpenCV - 2.3.1
```

Arduino Mega

```
#include <Wire.h>
#include <Servo.h>

#include <Max3421e.h>
#include <Usb.h>
#include <AndroidAccessory.h>

#define  SERVO1        11
#define  SERVO2        12
#define  SERVO3        13

#define  MOTOR_0       10
#define  MOTOR_1        9

#define  LINE_0        A0
#define  LINE_1        A1
#define  LINE_CAN      A2

#define  BUTTON1       A6
#define  BUTTON2       A7

AndroidAccessory acc("Google, Inc.",
"DemoKit",
"DemoKit Arduino Board",
"1.0",
"http://www.android.com",
"0000000012345678");

Servo servos[5];

void setup();
void loop();

void init_buttons()
{
  pinMode(BUTTON1, INPUT);
  pinMode(BUTTON2, INPUT);

  // enable the internal pullups on the buttons
  digitalWrite(BUTTON1, HIGH);
  digitalWrite(BUTTON2, HIGH);
}

void init_lineFollowers()
{
  pinMode(LINE_0, INPUT);
  pinMode(LINE_1, INPUT);
  pinMode(LINE_CAN, INPUT);

  // Enable the internal pull-ups on the line followers
  digitalWrite(LINE_0, HIGH);
  digitalWrite(LINE_1, HIGH);
  digitalWrite(LINE_CAN, HIGH);
}

// Global variables
```

```
byte b1, b2;
int ls1, ls2, ls3;

void setup()
{
 Serial.begin(115200);
 Serial.print("\r\nStart");

 init_buttons();
 init_lineFollowers();

 servos[0].attach(SERVO1);
 servos[0].write(map(5, 0, 255, 0, 180));
 servos[1].attach(SERVO2);
 servos[1].write(map(250, 0, 255, 0, 180));
 servos[2].attach(SERVO3);
 servos[2].write(90);
 servos[3].attach(MOTOR_0);
 servos[3].write(90);
 servos[4].attach(MOTOR_1);
 servos[4].write(90);

 b1 = digitalRead(BUTTON1);
 b2 = digitalRead(BUTTON2);
 ls1 = digitalRead(LINE_0);
 ls2 = digitalRead(LINE_1);
 ls3 = digitalRead(LINE_CAN);

 acc.powerOn();
}

void loop()
{
 byte msg[3];
 int ls;
 byte b;

 // Check if there are any messages coming in and send
 //   any important changes to the Android phone
 if (acc.isConnected())
 {
  int len = acc.read(msg, sizeof(msg), 1);

  if (len > 0)
  {
   // assumes only one command per packet
   if (msg[0] == 0x2)
   {
    if (msg[1] == 0x1)
     servos[3].write(map(msg[2], 0, 255, 0, 180));
    else if (msg[1] == 0x2)
     servos[4].write(map(msg[2], 0, 255, 0, 180));
    else if (msg[1] == 0x10)
     servos[0].write(map(msg[2], 0, 255, 0, 180));
    else if (msg[1] == 0x11)
     servos[1].write(map(msg[2], 0, 255, 0, 180));
    else if (msg[1] == 0x12)
     servos[2].write(map(msg[2], 0, 255, 0, 180));
   }
   else if (msg[0] == 0x3)
   {
    if (msg[1] == 0x1)
    {
     int ls = analogRead(LINE_0);

     if ((ls > 512 && !ls1) || (ls <= 512 && ls1))
```

```
  {
    ls1 = ls > 512;

    if (acc.isConnected())
    {
      msg[0] = 0x1;
      msg[1] = 0;
      msg[2] = ls1 ? 1 : 0;
      acc.write(msg, 3);
    }

    if (ls1)
      Serial.println("Light sensor 1 now detecting black");
    else
      Serial.println("Light sensor 1 now detecting white");
  }
}
else if (msg[1] == 0x2)
{
  ls = analogRead(LINE_1);

  if ((ls > 512 && !ls2) || (ls <= 512 && ls2))
  {
    ls2 = ls > 512;

    if (acc.isConnected())
    {
      msg[0] = 0x1;
      msg[1] = 1;
      msg[2] = ls2 ? 1 : 0;
      acc.write(msg, 3);
    }

    if (ls2)
      Serial.println("Light sensor 2 now detecting black");
    else
      Serial.println("Light sensor 2 now detecting white");
  }
}
else if (msg[1] == 0x3)
{
  ls = analogRead(LINE_CAN);

  if ((ls > 990 && !ls3) || (ls <= 990 && ls3))
  {
    ls3 = ls > 990;
    if (acc.isConnected())
    {
      msg[0] = 0x1;
      msg[1] = 2;
      msg[2] = ls3 ? 1 : 0;
      acc.write(msg, 3);
    }
  }
}
else if (msg[1] == 0x4)
{
  boolean b = digitalRead(BUTTON1);
  if (b != b1)
  {
    b1 = b;

    if (acc.isConnected())
    {
      msg[0] = 0x2;
      msg[1] = 0;
```

```
          msg[2] = b1;
          acc.write(msg, 3);
         }

         if (b1 == LOW)
          Serial.println("Button 1 pressed");
         else
          Serial.println("Button 1 released");
        }
       }
       else if (msg[1] == 0x5)
       {
        b = digitalRead(BUTTON2);
        if (b != b2)
        {
         b2 = b;

         if (acc.isConnected())
         {
          msg[0] = 0x2;
          msg[1] = 1;
          msg[2] = b;
          acc.write(msg, 3);
         }

         if (b2 == LOW)
          Serial.println("Button 2 pressed");
         else
          Serial.println("Button 2 released");
        }
       }
      }
     }
    }
    else
    {
     // reset outputs to default values on disconnect

     servos[0].write(map(5, 0, 255, 0, 180));
     servos[1].write(map(250, 0, 255, 0, 180));
     servos[2].write(90);
     servos[3].write(90);
     servos[4].write(90);
    }

    // Delay 10 milliseconds before looping again
    delay(10);
}
```