Kinect Kicker

University of Florida
EEL 4665/5666
Intelligent Machines Design Laboratory

By Naveen Dhawan

Instructors: Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz
TA's: Devin Hughes Tim Martin
Ryan Stevens Josh Weaver

Naveen Dhawan
April 19, 2011

Table of Contents

1.0 Abstracts

The Kinect Kicker is designed to kick certain colored balls and demonstrate other intelligent behaviors with the use of the Microsoft Kinect sensor whose original purpose is to provide a controllerless experience to persons who own a XBOX 360 console. The Kinect contains the following senesors: accelerometer, a depth image sensing camara, a RGB camera and an array of audio sensors. It also has a motor that helps tilt the camera up and down. The purpose of this project is to prove that the Kinect Sensor is a viable sensor that can be integrated to robotics seamlessly. During the writing of this report many have began work on the Kinect with various applications and few have been with the use of Robotics. This report details one method how one can integrate the Kinect Sensor to a robotics application, including information on how to program and controller that can process the data the Kinect provides.

For this project, the robot utilizes the camera's depth and RGB sensors for obstacle avoidance and ball detection to act out behaviors preprogrammed in the controller such as kicking. The controller chosen is the Beagleboard C4 model, a board that utilizes the TI OMAP3530 built for mobile applications, it is powerful enough to run the Kinect through a USB port and process the data returned with the use of the Angstrom Operating System. The Kinect sits on top of the robot with the circuitry and microcontroller found in the middle layer of the robot. The bottom layer contains the actuation peripherals for moving and kicking.

2.0 Executive Summary

In the past year from the writing of the report, the Microsoft Kinect sensor has garnered much support from the open source community especially in the ease of integrating it to projects that require sensors. Through drivers built by members of the open source community we can now have access to the range of detail of 3D objects in all x, y and z directions. This brings a lot of possibility, as it's far easier to discern data with an extra dimension rather than an image, which only consists of 2D information.

In this project I will construct a robot that integrates the Kinect Sensor to complete a simple task. The task I have chosen is to use the sensors depth range sensor in order to properly avoid obstacles and then use the RGB camera to discern a special colored ball in which it will approach to kick. The robot will also utilizes bump sensors for obstacle avoidance due to the Kinect's cone of invisibility to see to the left and the right of where obstacles maybe. The main purpose of this report is to show the feasibility of using the Kinect and its applications to robotics.

Integrating the Kinect with the Beagleboard, at first was a daunting task because from taking on the project I had limited experience in Linux operating systems, writing and using drivers, also using a microcontroller that actually can double as a computer and programming in C++. Although the problem was certainly solvable using an embedded Linux distribution on the Beagleboard to control the Kinect, without experience in this regard added time to successfully hacking the Kinect. The information provided by the Kinect comes from two different streams over USB serially, the raw depth information, which comes in 11 bits 1 channel matrix, and the RGB data which comes in 8 bit 3 channel matrix. The obstacle avoidance code was written in C, which did not prove too difficult, and although improvements were planned it was never done, yet the avoidance system was very reliable.

Once the data became usable for obstacle avoidance the RGB camera along side OpenCV provided the detection of green colored balls. Luckily, Angstrom had prebuilt packages using OpenCV 2.2 and most coding occurred on the host computer and compile within the Beagleboard. Differences in file systems and using porting C to C++ required a little bit of debugging in order to have a fully functioning code base.

The MigaOne Nanoscale linear actuator provides the kicking actuation of the robot. This actuator uses a special metal alloy that goes a to a specific shape when heated up. At least 29V were used to provide a kicking force that can push a ball a few feet. Unfortunately, although mounted it did not actually make it into the final demo of the project however further work will continue.

3.0 Introduction

My project for my robot is based on the concept on using the Kinect Sensor for Microsoft's XBOX to help provide the majority of its external information in order to autonomously interact with the environment. The main challenges in this project are how to interface with the Kinect and use the data provided, process this information and provide it in a meaningful way to control the robot. Other challenges include the use of the Beagleboard, an open-source board that utilizes the Texas Instrument OMAP3530 with an Arm processor. This processor is designed for mobile applications. It has been proven that the Kinect is easily tinkered with the use of Linux/Mac OSX drivers and the use of OpenCV (Open Source Computer Vision) to allow users to create hand recognition algorithms to produce results such as sculpting or puppet software. The puppetting software was in fact a day project using OpenCV and OfxKinect (Linux). This is exciting as it implies that one can use the Kinect to interact with the digital world and this project aims to prove that it can apply to robotics. My main goal is to provide a framework that allows a robot to interact with the outside world more easily and accurately. For the purposes of this class I am interested in kicking a circular object (like a ball) using a servo with the Kinect Sensor as its main eyes. Other sensors such as bump sensors will help the robot know weather an object is too close so as to take an alternate route.

In this report one will find information regarding the approach taken, the challenges the problem posed and finally the results of the work done in the semester.

4.0 Integrated System

Due to the complexity of the Beagleboard the system begins after boot up and the program is ran. An on/off switch is provided to power the servos when needed while the program is already running.

In the program, the camera, motor, sonar, actuator routines will initialize and activate the respective devices. Then the program will enter an infinite loop where the image data from the Kinect will continuously run through image processing code to provide outputs of objects within the view of the robot. The robot will then control its motors based on these outputs. If a ball is identified, the robot will begin its approach while trying to maneuver around possible obstacles. A flag will identify whether if the robot is within range to provide a kick to the ball. If true, the actuator will be called. The program also utilizes threads to poll the Bump Switches, which will act as a interrupt for the program in order to keep objects not in view from interfering with the robots movement. See the following diagram regarding the system layout.

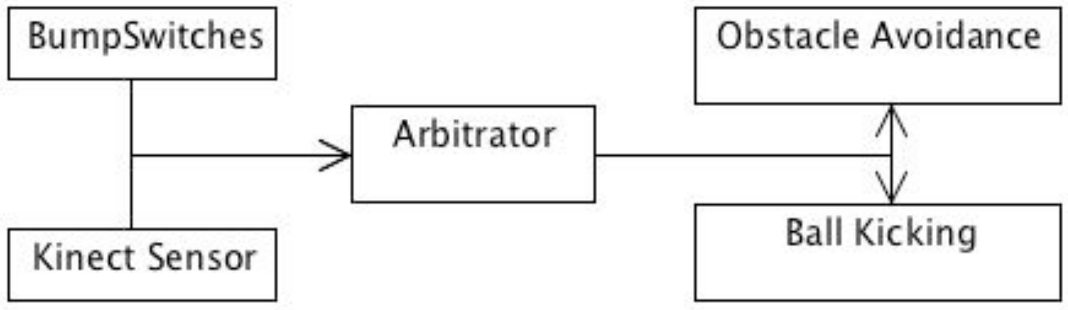


Figure 1: Visualizes the programming layout on the Beagleboard. The BumpSwitches and Kinect Sensor runs on separate threads while the Arbitrator polls data and then runs routines for image processing. Finally an obstacle avoidance and ball kicking routine is decided upon based on the data collected.

5.0 Mobile Platform

The mobile platform will comprise of a flat mounting surface where the wheels, motor assembly, Kinect sensor, and bump switches will be placed. Two servos will be used to actuate the robot where two bump switches mounted on the front left and front right, to help the robot from bumping and getting stuck onto sides of corners or doors. A kicking board not pictured will be used to provide the kicking force needed to kick a small ball.

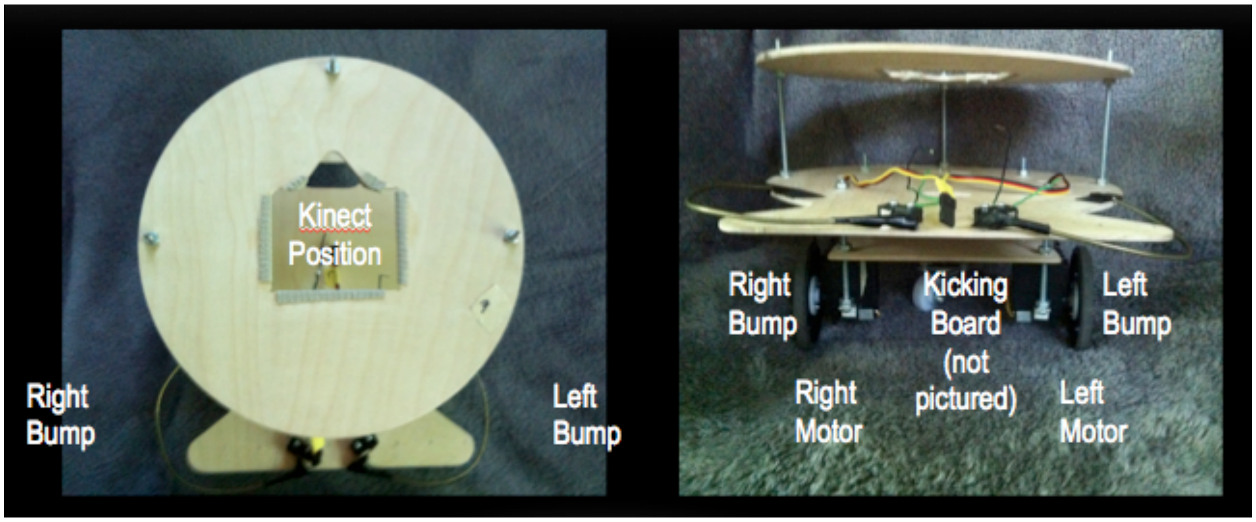


Figure 2: This details the current design of the Kinect Kicker.

6.0 Actuation

The robot will consist of two different type of actuator performing different types of functions. One function is for movement, which is done with the use of two servos that use PWM signals to actuate a specific direction. Hacking of the servos allows one to gain complete control of how the motors move.

Another actuator, known as shape memory alloy linear actuator will be used to provide the kicking for the robot. This actuator will use fishing line wire and metal rod, which serve as a pivot point to provide 2.5 pounds of force. The torque expected based on the distance between the pivot point is capable of moving the ball many feet from the source location.

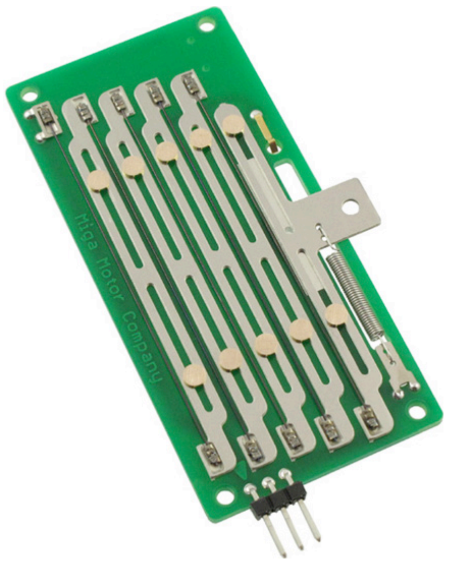


Figure 3: Pictures one of the SMA actuators produced by MigaOne Motor Company mounted on the robot.

7.0 Sensors

Kinect Sensor

The Kinect Sensor consists of two individual sensors. It has a RGB camera which provides 8-bit VGA resolution (640 x 480) with a Bayer color filter. The other sensor provides 11 bit depth at 2048 levels of sensitivity with the use of continuously projected infrared structured light and a CMOS sensor. This provides a powerful stream of data that potentially allows users to interact with 3D data.

This is the main sensor of the system and most of the behavior of the robot will be based on what information is provided by this sensor. The time and effort it may take to have the sensor working viably on the robot may take a long time, therefore other ideas are not being used yet, since this will place unnecessary load on the designer. I would like to be able to have it complete by the end of February so that more interesting ideas can be incorporated with the Kinect Sensing robot.

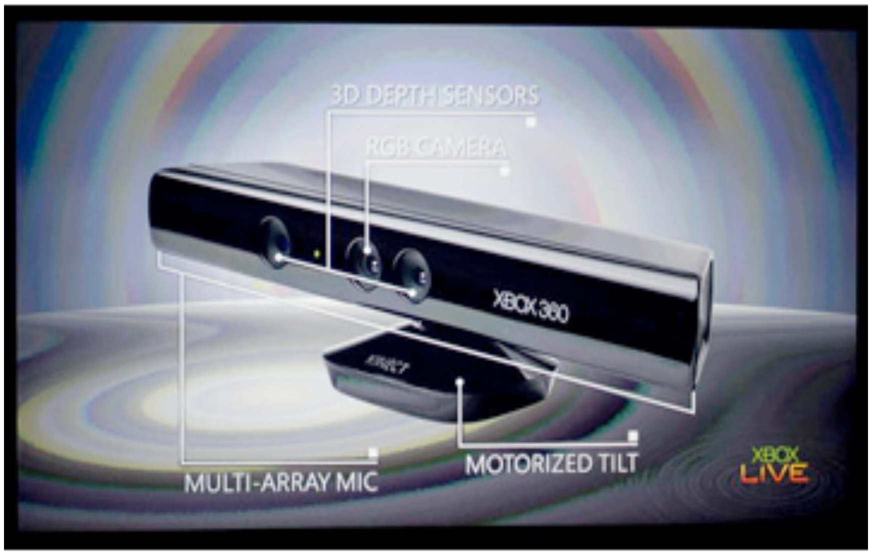


Figure 4: Pictures a Kinect Sensor that annotates locations of special sensors installed in the Kinect.

Bump Sensors

Two bump sensors are used to help provide the KinectKicker visibility in areas on the left and right of the Kinect's cone of visibility. Since the Kinect cannot see too far to the left and the right it can at times get caught in doorways and corners, therefore these sensors remedy this problem by providing an active low input into the Beagleboard.

The following diagram details the circuit schematic of the entire system including the sensors.

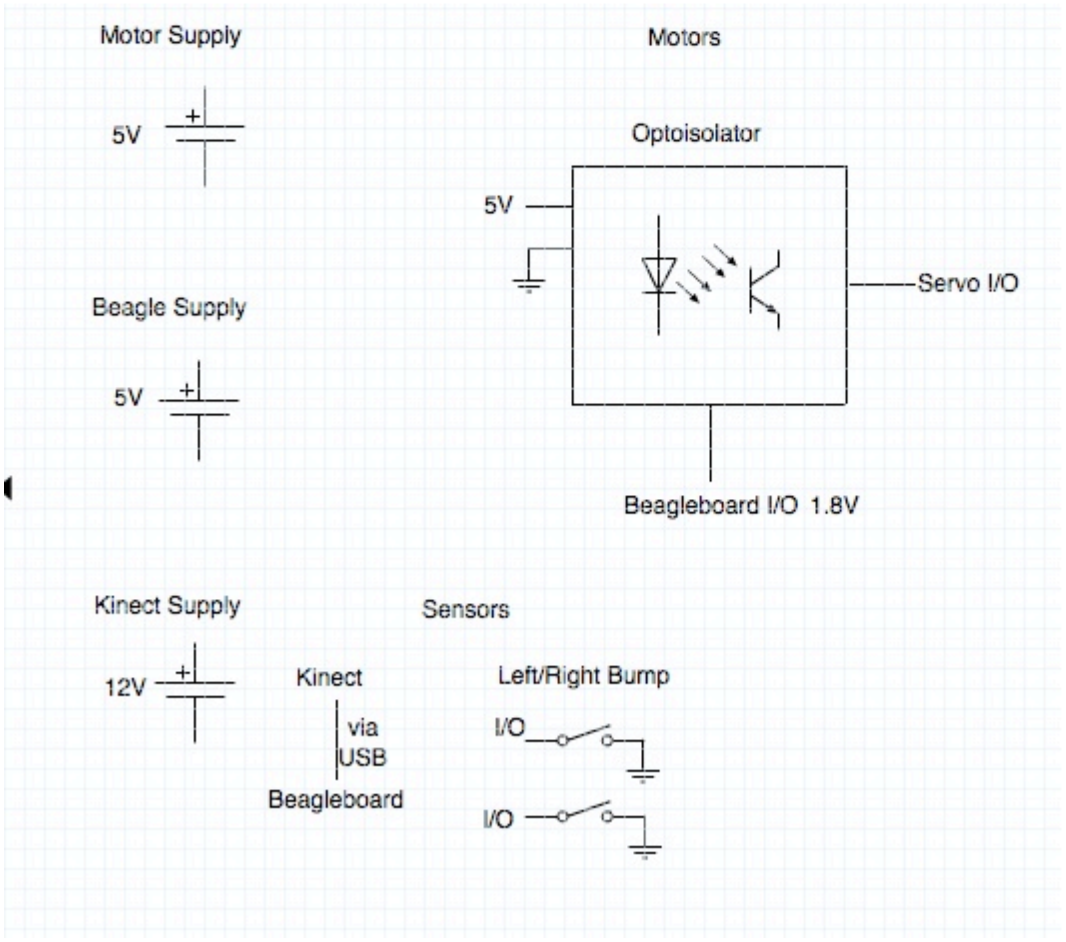


FIGURE 1: This figure details the general set up used to interface the Beagleboard with the rest of the world. The Beagleboard interacted with the outside world via GPIO pins, PWM signals and USB peripheral.

One of the main challenges with the Beagleboard was the 1.8V input/output logic level which is not normally found in sensors used. The use of the 1.8V signals is to allow the board to need as little power as necessary to function. With the use of special optoisolating circuits, this problem was solved.

8.0 Behaviors

The robot will have very simple behavior to demonstrate the different things the Kinect Kicker can do. Here are some of the behaviors along side with the code that is used to.

Behavior #1

Detect green balls, turn left and then go slow for a little bit. This behavior is intended to demonstrate that a special colored ball can be used to provide the robot direction of where a special item might be.

```
if (GREEN_BALL) {
           //FOUND BLUE BALL TURN RIGHT
           desiredLeftMotorVal = LeftMotorForwardF;
           desiredRightMotorVal = RightMotorReverse;
           leftMotorUpdated = 1;
           rightMotorUpdated = 1;
           leftMotorUpdate();
           rightMotorUpdate();
           sleep(1);
           desiredLeftMotorVal = LeftMotorForwardS;
           desiredRightMotorVal = RightMotorForwardS;
           printf("Found Blue Ball, Soft Right\n");
                GREEN_BALL = 0;
}
```

Behavior #2

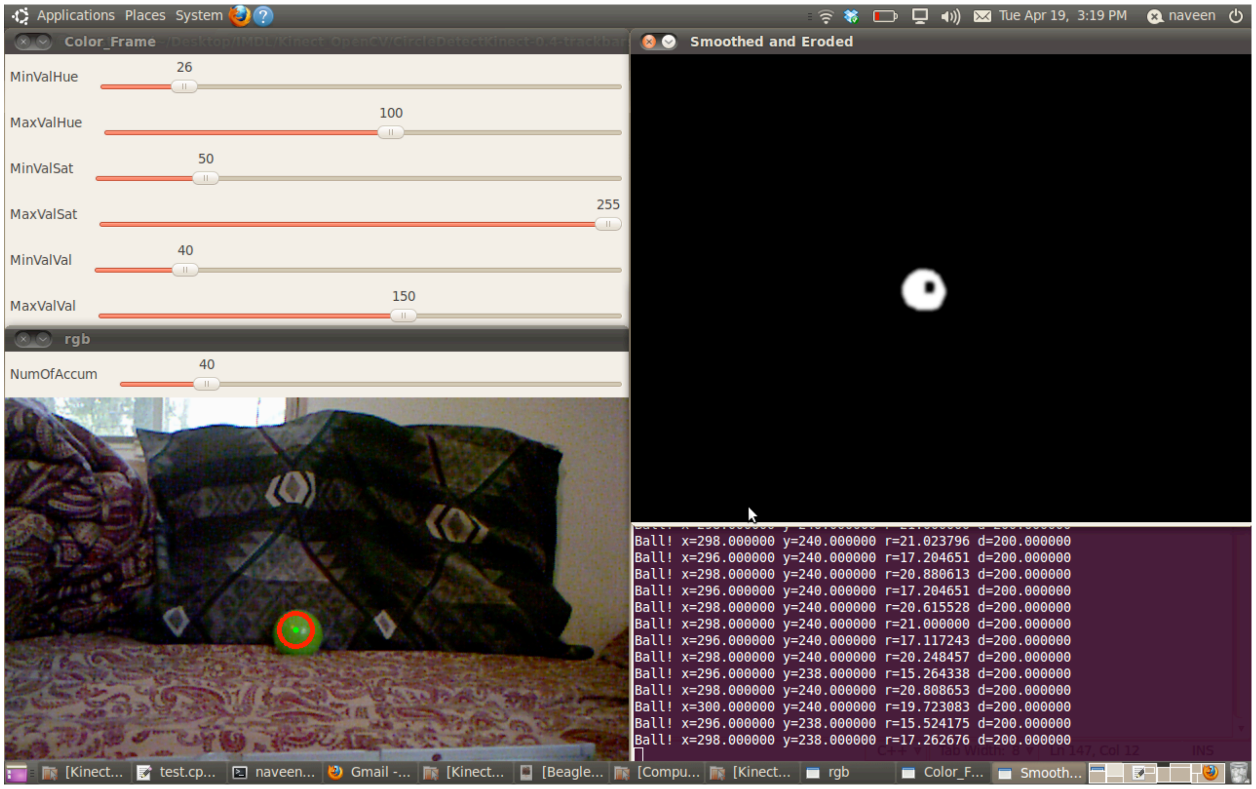
Kicking a blue ball. This behavior is intended to demonstrate the KinectKicker can use both depth, color, and accelerometer data from the Kinect Sensor in order to precisely approach and kick a ball.

```
if (BLUE_BALL) {
      sleep(1);
      desiredLeftMotorVal = LeftMotorStall;
      desiredRightMotorVal = RightMotorStall;
      leftMotorUpdated = 1;
      rightMotorUpdated = 1;
      leftMotorUpdate();
      rightMotorUpdate();
      //Kick
      kick();  //Simple low, high, low output on GPIO139
      //Turn Right
      desiredLeftMotorVal = LeftMotorForwardF;
      desiredRightMotorVal = RightMotorReverseF;
      leftMotorUpdated = 1;
      rightMotorUpdated = 1;
      leftMotorUpdate();
      rightMotorUpdate();
      }

int kick() {
    system("echo 139 > /sys/class/gpio/export");
    system("echo output > /sys/class/gpio139/direction");
    printf("Echoed 139\n");
    system("echo low > /sys/class/gpio/gpio139/direction");
    system("echo high > /sys/class/gpio/gpio139/direction");
    system("echo low > /sys/class/gpio/gpio139/direction");
}
```

9.0 Experimental Layout and Results

OpenCV and the Kinect Sensor were used to help determine where a specially colored ball can be found. Here in this image one can see how well the Image Processing algorithm was able to detect a green colored ball as well as show exactly where the center point of the ball was from the image.


Applications Places System
Tue Apr 19, 3:19 PM
naveen
Color_Frame
rgb
Smoothed and Eroded
[Kinect... test.cp... naveen... Gmail -... [Kinect... [Beagle... [Compu... [Kinect... rgb Color_F... Smooth...

MinValHue 26
MaxValHue 100
MinValSat 50
MaxValSat 255
MinValVal 40
MaxValVal 150
NumOfAccum 40

```
Ball! x=298.000000 y=240.000000 r=21.023796 d=200.000000
Ball! x=296.000000 y=240.000000 r=17.204651 d=200.000000
Ball! x=298.000000 y=240.000000 r=20.880613 d=200.000000
Ball! x=296.000000 y=240.000000 r=17.204651 d=200.000000
Ball! x=298.000000 y=240.000000 r=20.615528 d=200.000000
Ball! x=298.000000 y=240.000000 r=21.000000 d=200.000000
Ball! x=296.000000 y=240.000000 r=17.117243 d=200.000000
Ball! x=298.000000 y=240.000000 r=20.248457 d=200.000000
Ball! x=296.000000 y=238.000000 r=15.264338 d=200.000000
Ball! x=298.000000 y=240.000000 r=20.808653 d=200.000000
Ball! x=300.000000 y=240.000000 r=19.723083 d=200.000000
Ball! x=296.000000 y=238.000000 r=15.524175 d=200.000000
Ball! x=298.000000 y=238.000000 r=17.262676 d=200.000000
```

10.0 Conclusion

The course of this project has been very exciting. Not only was able to communicate with the Kinect through the Beagleboard I learned a lot on how to make things work in an embedded system. Although the robot was unable to complete its primary objective which was to kick a ball, it definitely demonstrated the range of possibilities one can do using the Kinect and the Beagleboard. Overall I would consider this a good experience and the lessons learned will be very valuable to me in the future.

I am also happy to have been able to work with the Kinect on my project and I intend to keep learning how to use it and try to do the more interesting things one can do with it such as gaining a colored point cloud of the data provided and possibly using hand movement to interface with systems.

Some lessons learned were that learning curves definitely exist and the more you know how something works the quicker and more efficient you can move especially when it comes to programming. As this was the first time for me to use Linux, OpenCV, the Beagleboard it was very long and difficult time to figure out how to make the pieces work. The second learned lesson is to make good wires because due to this I spent more time fixing my wires then getting to use my awesome robot to do more things. Finally, the third lesson learned is to not get carried away although I don't think I have fully understood this one yet. At times I would focus more on learning the image processing behind the Kinect rather than actually see what I can do to get my robot up and running. But its all in the interest of learning more about things that you are interested. In the end, I would recommend this class to anyone as it allows you to explore your interests and be able to build on top of the knowledge you already have a gained, which is a truly a valuable experience if done right.

9.0 Appendix

Code for the Kinect Kicker

```
/*
g++ *.c *.cpp -o KinectKicker -I/usr/local/include/libfreenect -
I/usr/local/include/opencv -I/usr/local/include/opencv2 -
I/usr/include/glib-2.0 -I/usr/lib/glib-2.0/include -lopencv_core -
lopencv_highgui -lpthread -lglib-2.0 -lusb-1.0
/-----------------/
BeagleBoard Compile
/-----------------/
g++ *.c *.cpp -o KinectKicker -I/usr/include/opencv -
I/usr/include/opencv2 -I/usr/include/glib-2.0 -I/usr/lib/glib-
2.0/include -I/home/root/KinectKicker-0.4/ -lopencv_core -
lopencv_highgui -lpthread -lglib-2.0 -lusb-1.0
*/


//OPENCV INCLUDE FILES
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>

//STANDARD INCLUDE FILES
#include <iostream>
#include <vector>
#include <cmath>
#include <pthread.h>
#include <glib.h>
#include <math.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>

//KINECT KICKER INCLUDE FILES
#include "KinectProcessor.h"
#include "KinectKicker.h"
#include "libfreenect.hpp"
#include "libfreenect.h"
#include "libfreenect_sync.h"
#include "OMAP3530_REGISTERS.h"
#include "OMAP3530_PWM.h"
#include "BumpSwitch.h"
```

```
//#define SHOW_VIDEO

//#define SAVE_IMAGES

//#define ON_BEAGLEBOARD

//#define ANNOYING_DEBUG_OUTPUT

#define ORIG_IMAGEPROCESS       0

using namespace cv;
using namespace std;

/*==========================================*/
//Kinect Processor declarations
CvSize size = cvSize(640,480);
IplImage* frame = cvCreateImage(size, IPL_DEPTH_8U, 3);
IplImage* hsv_frame = cvCreateImage(size, IPL_DEPTH_8U, 3);
IplImage* color_threshold    = cvCreateImage(size, IPL_DEPTH_8U, 1);
IplImage* frameDepth = cvCreateImage(cvSize(640,480), IPL_DEPTH_16U,
1);
IplImage* depthf = cvCreateImage(cvSize(640,480), IPL_DEPTH_8U, 1);
CvMemStorage* storage = cvCreateMemStorage(0); //needed for Hough
circle
CvScalar hsv_min, hsv_max;
CvSeq *circleArray[5];
CvSeq* circles;
int newDepthFrame;
int newRGBFrame;
int newFrame;
Mat depthMat(Size(640, 480), CV_16UC1);
int GREEN_BALL;
int BLUE_BALL;
int RED_BALL;
int ACTION;
int counter;

int minHueBlue = 100;
int maxHueBlue = 158;
int minSatBlue = 53;
int maxSatBlue = 255;
int minValBlue = 65;
int maxValBlue = 255;

int minHueGreen = 26;
int maxHueGreen = 100;
int minSatGreen = 50;
int maxSatGreen = 255;
int minValGreen = 40;
int maxValGreen = 150;
```

```
/*int minHueRed = 135;
int maxHueRed = 179;
int minSatRed = 60;
int maxSatRed = 255;
int minValRed = 70;
int maxValRed = 255;*/
/*============================================*/

//--------------------Declarations--
//KinectKicker Declaratinos
int leftMotorUpdated, rightMotorUpdated;
int leftBump = 0;
int rightBump = 0;
guint8 *leftMotor;//Left = gpt10
guint8 *rightMotor;//Right = gpt11
guint32 resolution;
int mem_fd;
double currentLeftMotorVal, currentRightMotorVal;
double desiredLeftMotorVal, desiredRightMotorVal;
struct kinectSensorIRValues s;


//-------------------File-Specific--


class Mutex {
public:
    Mutex() {
     pthread_mutex_init( &m_mutex, NULL );
    }
    void lock() {
     pthread_mutex_lock( &m_mutex );
    }
    void unlock() {
     pthread_mutex_unlock( &m_mutex );
    }
private:
    pthread_mutex_t m_mutex;
};

class MyFreenectDevice : public Freenect::FreenectDevice {
  public:
    MyFreenectDevice(freenect_context *_ctx, int _index)
     : Freenect::FreenectDevice(_ctx, _index),
m_buffer_depth(FREENECT_DEPTH_11BIT_SIZE),m_buffer_rgb(FREENECT_VIDEO_
RGB_SIZE), m_gamma(2048), m_new_rgb_frame(false),
m_new_depth_frame(false),
     depthMat(Size(640,480),CV_16UC1),
rgbMat(Size(640,480),CV_8UC3,Scalar(0)),
ownMat(Size(640,480),CV_8UC3,Scalar(0))
    {
     for( unsigned int i = 0 ; i < 2048 ; i++) {
          float v = i/2048.0;
          v = std::pow(v, 3)* 6;
          m_gamma[i] = v*6*256;
     }
    }
    // Do not call directly even in child
    void VideoCallback(void* _rgb, uint32_t timestamp) {
     if(counter<1) {
          counter++;
          return;
     }
     else {
          //std::cout << "RGB callback" << std::endl;
          m_rgb_mutex.lock();
          uint8_t* rgb = static_cast<uint8_t*>(_rgb);
          rgbMat.data = rgb;
          m_new_rgb_frame = true;
          newRGBFrame = 1;
          m_rgb_mutex.unlock();
          counter = 0;
     }
    };
    // Do not call directly even in child
    void DepthCallback(void* _depth, uint32_t timestamp) {
     //std::cout << "Depth callback" << std::endl;
     m_depth_mutex.lock();
     uint16_t* depth = static_cast<uint16_t*>(_depth);
     //uint32_t* depth = static_cast<uint32_t*>(_depth);
     depthMat.data = (uchar*) depth;
     m_new_depth_frame = true;
     newDepthFrame = 1;
     m_depth_mutex.unlock();
    }

    bool getVideo(Mat& output) {
     m_rgb_mutex.lock();
     if(m_new_rgb_frame) {
          cv::cvtColor(rgbMat, output, CV_RGB2BGR);
          m_new_rgb_frame = false;
          m_rgb_mutex.unlock();
          return true;
     } else {
          m_rgb_mutex.unlock();
          return false;
     }
    }
    bool getDepth(Mat& output) {
            m_depth_mutex.lock();
            if(m_new_depth_frame) {
                 depthMat.copyTo(output);
                 m_new_depth_frame = false;
                 m_depth_mutex.unlock();
                 return true;
            } else {
                 m_depth_mutex.unlock();
                 return false;
            }
     }

  private:
    std::vector<uint8_t> m_buffer_depth;
    std::vector<uint8_t> m_buffer_rgb;
    std::vector<uint16_t> m_gamma;
    Mat depthMat;
    Mat rgbMat;
    Mat ownMat;
    Mutex m_rgb_mutex;
    Mutex m_depth_mutex;
    bool m_new_rgb_frame;
    bool m_new_depth_frame;
};


void computeCircles() {
     //printf("Starting Image Processing!\n\n\n\n\n\n\n\n\n\n\n\n");
     // color detection using HSV
     cvCvtColor(frame, hsv_frame, CV_BGR2HSV);
     //Filter only color designated by trackbars
     cvInRangeS(hsv_frame, hsv_min, hsv_max, color_threshold);
     cvErode(color_threshold, color_threshold, NULL, 3);
     #ifdef SHOW_VIDEO
          cv::imshow("Color_Threshed", color_threshold);
     #endif
     // hough detector works better with some smoothing of the image
    //    cvSmooth(color_threshold, color_threshold, CV_GAUSSIAN, 9,
9 );
     //circles = cvHoughCircles(color_threshold, storage,
CV_HOUGH_GRADIENT, 2,60, 100, 35, 0, 40);
     circles = cvHoughCircles(color_threshold, storage,
CV_HOUGH_GRADIENT, 2, color_threshold->height/2, 200, 40, 10, 200);
     /*for (int i = 0; i < circles->total; i++)
     {
          float* p = (float*)cvGetSeqElem(circles, i);
          printf("Ball! x=%f y=%f r=%f\n\r",p[0],p[1],p[2] );
          //cvCircle( frame, cvPoint(cvRound(p[0]),cvRound(p[1])),3,
CV_RGB(0,255,0), -1, 8, 0 );
```

```
			//cvCircle( frame, 
cvPoint(cvRound(p[0]),cvRound(p[1])),cvRound(p[2]), CV_RGB(255,0,0), 
3, 8, 0 );
		}*/

		//exit(0);
		//return circles;
}
int DetectBallFunc() {
//void *DetectBallFunc(void*) {
	//float sumRadius = 0;
	int circleCount = 0;
	float avgRadius = 0;
	int frameCount = 0;
	int k;
	if (newFrame) {
		ACTION = 0;
		//newFrame = 0;
		computeCircles();
		if (circles->total == 0) {
			frameCount=0;
			//printf("Finished Image Processing(1)!!!!\n\n\n\n\n\n");
			//continue;
			return 0;
		}
		//circleArray[frameCount] = circles;
		//frameCount++;
		//if (frameCount > 4) {
			//Calculate whether to turn
		//	for (k = 0; k < 5; k++)
		//	{
		//		circles = circleArray[k];
				for (int i = 0; i < circles->total; i++) {
					float* p = (float*)cvGetSeqElem(circles, i);
					//printf("Ball! x=%f y=%f 
r=%f\n\r",p[0],p[1],p[2] );
					//cvCircle( frame, 
cvPoint(cvRound(p[0]),cvRound(p[1])),3, CV_RGB(0,255,0), -1, 8, 0 );
					//cvCircle( frame, 
cvPoint(cvRound(p[0]),cvRound(p[1])),cvRound(p[2]), CV_RGB(255,0,0), 
3, 8, 0 );
					//sumRadius += *p+2;
					//circleCount++;
					avgRadius = p[2];
					if (avgRadius>10) {
						break;
					}
				}
			//}
			//avgRadius = sumRadius/circleCount;
```

```
            if(avgRadius > 20) {
                printf("ACTION!!!!\n");
                ACTION = 1;
                //STATE++;
                //STATE_CHANGED++;
                //if (STATE > 3) {
                //    STATE = 0;
                //}
            }
            else {
                printf("No balls?, find Balls!!\n");
                ACTION = 0;
            }
            //frameCount = 0;
      //}
    //printf("Finished Image Processing(2)!!!!\n\n\n\n\n\n");
    //exit(0);
    }
    return 0;
}

//void *collectKinectData(void*) {
//int collectKinectData() {
int main(int argc, char **argv) {
    //Initalization Step
    pthread_t motorThread, bumpThread, kinectThread;
    pthread_t populateSensorData, detectBalls;
    //pthread_create( &kinectThread, NULL, collectKinectData, NULL);
    pthread_create( &bumpThread, NULL, bumpSwitch_init, NULL);
    //pthread_create( &motorThread, NULL, motor_init, NULL);
    //initBUMPERS();
    motor_init();

    bool die(false);
    //Mat depthMat(Size(640,480),CV_16UC1);
    //Mat depthf  (Size(640,480),CV_8UC1);
    Mat rgbMat(Size(640,480),CV_8UC3,Scalar(0));
    //Mat ownMat(Size(640,480),CV_8UC3,Scalar(0));

        Freenect::Freenect freenect;
        MyFreenectDevice& device =
freenect.createDevice<MyFreenectDevice>(0);
    #ifdef SHOW_VIDEO
     namedWindow("rgb",CV_WINDOW_AUTOSIZE);
     namedWindow("Color_Frame",CV_WINDOW_AUTOSIZE);
     namedWindow("Depth_Frame",CV_WINDOW_AUTOSIZE);
    #endif
    //cvCreateTrackbar("MinValHue", "Color_Frame", &minHue, 179,
minHueCallback);
    //cvCreateTrackbar("MaxValHue", "Color_Frame", &maxHue, 179,
maxHueCallback);
    //cvCreateTrackbar("MinValSat", "Color_Frame", &minSat, 255,
    ACTION = 0;  //Do not perform any action

    device.startVideo();
    device.startDepth();
    pthread_create(&populateSensorData, NULL, parseKinectIRValues,
NULL);
    //pthread_create(&detectBalls, NULL, DetectBallFunc, NULL);
    //Begin Arbitrator (combined with the original KinectThread);
    while (!die) {
#ifdef ANNOYING_DEBUG_OUTPUT
    printf("Grabbing frame\n");
#endif
    device.getVideo(rgbMat);
    device.getDepth(depthMat);

    //parseKinectIRValues();
    //readBumpSwitches();
    cvSetData(frame, rgbMat.data, 640*3);
    if(newRGBFrame) {
     newRGBFrame = 0;
     newFrame = 1;
    }
    cvSetData(frameDepth, depthMat.data, 640*2);
motorBehavior();
leftMotorUpdate();
rightMotorUpdate();
usleep(300000);
    //s = parseKinectIRValues((short*)depthMat.data);
    if (frame != NULL) {
     //printf("got frame\n\r");
     //Intelligence
     /*hsv_min = cvScalar(minHueBlue, minSatBlue, minValBlue, 0);
     hsv_max = cvScalar(maxHueBlue, maxSatBlue, maxValBlue, 0);
     DetectBallFunc();
     if (ACTION) {
          BLUE_BALL = 1;
     }*/
     ACTION = 0;
     hsv_min = cvScalar(minHueGreen, minSatGreen, minValGreen, 0);
     hsv_max = cvScalar(maxHueGreen, maxSatGreen, maxValGreen, 0);
     DetectBallFunc();
     if (ACTION) {
          GREEN_BALL = 1;
     }
     ACTION = 0;
motorBehavior();
leftMotorUpdate();
rightMotorUpdate();
```

```
usleep(300000);
//    ACTION = 0;
    #ifdef SAVE_IMAGES
     cvSaveImage("frame.jpg", frame);
    #endif
        } else {
            printf("Null frame\n\r");
        }
    #ifdef SHOW_VIDEO
     cv::imshow("rgb", frame);
     cvConvertScale(frameDepth, depthf, 255/2048.0);
     cv::imshow("depth", depthf);
    #endif
    }
#ifdef ON_BEAGLEBOARD
    pwm_munmap_instance(leftMotor);
    pwm_munmap_instance(rightMotor);
    close_devmem(mem_fd);
#endif
    device.stopVideo();
    device.stopDepth();
    return 0;
}
//int parseKinectIRValues() {
//struct kinectSensorIRValues parseKinectIRValues(short *depth) {
void *parseKinectIRValues(void*) {
    //struct kinectSensorIRValues s;
    static unsigned int indices[480][640];
    static short xyz[480][640][3];
    int i, j;
while(1) {
    if (newDepthFrame) {
    #ifdef ANNOYING_DEBUG_OUTPUT
     printf("Updating IR Sensor Values\n");
    #endif
     short *depth = (short*)depthMat.data;
     newDepthFrame = 0;

     s.rawLeftSensorVal = 2500;
     s.rawMiddleSensorVal = 2500;
     s.rawRightSensorVal = 2500;

     for (i = 0; i < 480; i++) {
              for (j = 0; j < 640; j++) {
               xyz[i][j][0] = j;
               xyz[i][j][1] = i;
               xyz[i][j][2] = depth[i*640+j];
               indices[i][j] = i*640+j;
                    //check if in the leftsensor range
                    if (i < 380 && i > 180 && j < 200 && j > 50) {
                         if (s.rawLeftSensorVal > xyz[i][j][2]) {
                              s.rawLeftSensorVal = xyz[i][j][2];
                              //printf("checking depth values in 
LeftSensorRange");
                         }
                    }
                    if (i < 380 && i > 180 && j < 400 && j > 250) {
                         if (s.rawMiddleSensorVal > xyz[i][j][2]) {
                              s.rawMiddleSensorVal = xyz[i][j][2];
                              //printf("checking depth values in 
MiddleSensorRange");
                         }
                    }
                    if (i < 380 && i > 180 && j < 600 && j > 450) {
                         if (s.rawRightSensorVal > xyz[i][j][2]) {
                              s.rawRightSensorVal = xyz[i][j][2];
                              //printf("checking depth values in 
RightSensorRange");
                         }
                    }
               }
     }
     i =0;
     j=0;
     s.calcLeftSensorVal = convRawToInches(s.rawLeftSensorVal);
     s.calcMiddleSensorVal = convRawToInches(s.rawMiddleSensorVal);
     s.calcRightSensorVal = convRawToInches(s.rawRightSensorVal);
     printf("Left IR: %f, Mid IR: %f, Right IR: %f\n", 
s.calcLeftSensorVal, s.calcMiddleSensorVal, s.calcRightSensorVal);
    }
}
    //return s;
    return 0;
}//end of kinectSensorIRValues();


double convRawToInches(short val) {
     //return 0.1236 * tan(val/2842.5+1.1863) * 39.3700787;
     return 100/(-0.00307 * val + 3.33) * 0.393700787;
}


void no_kinect_quit(void)
{
    printf("Error: Kinect not connected?\n");
    exit(1);
}
```

```
int motor_init() {
//void *motor_init(void*) {

    pthread_t leftMotorThread, rightMotorThread;
#ifdef ON_BEAGLEBOARD
    //system("modprobe servodrive");
    mem_fd = open_devmem();
    if (mem_fd == -1) {
          g_error("Unable to open /dev/mem, are you root?: %s",
g_strerror(errno));
      }
    //Configure PWM10 and PWM11
    pwm_config_mux(mem_fd);
    //pwm_config_mux(mem_fd, 0);

    printf("pwm_config_clock() invoked\n");
    //Set PWM 10 and PWM11 periperal clock to 13MHz (rather than
default 32KHz...ick)
    pwm_config_clock(mem_fd, TRUE, TRUE);

    printf("pwm_mmap_instance() invoked\n");
    //Retrieves the pointers to PWM10 and PWM11
    leftMotor = pwm_mmap_instance(mem_fd, 10);
    rightMotor = pwm_mmap_instance(mem_fd, 11);

    printf("pwm_calc_resolution() invoked\n");
    //Sets the frequency to be 50Hz
    resolution = pwm_calc_resolution(50, PWM_FREQUENCY_13MHZ);

    //Configure the motors at start up and set them at the default
7.09 value
    pwm_config_control(leftMotor, resolution);
    pwm_config_control(rightMotor, resolution);
    pwm_config_timer(leftMotor, resolution, 7.09/100);
    pwm_config_timer(rightMotor, resolution, 7.09/100);
    leftMotorUpdated = 0;
    rightMotorUpdated = 0;
#endif
    printf("Motor initialized!\n");
    //pthread_create( &leftMotorThread, NULL, leftMotor_Thread, NULL);
    //pthread_create( &rightMotorThread, NULL, rightMotor_Thread, NULL
);
    return 0;
}
void leftMotorUpdate() {
//void *leftMotor_Thread(void*) {

    //while (1) {
#ifdef     ANNOYING_DEBUG_OUTPUT
    printf("Updating LeftMotor\n");
```

```
#endif
#ifdef ON_BEAGLEBOARD
     if(leftMotorUpdated) {
          pwm_config_dutyCycle(leftMotor, resolution,
currentLeftMotorVal, desiredLeftMotorVal);
          leftMotorUpdated = 0;
          currentLeftMotorVal = desiredLeftMotorVal;
     }
#endif
    //}

}
void rightMotorUpdate() {
//void *rightMotor_Thread(void*) {

    //while (1) {
#ifdef     ANNOYING_DEBUG_OUTPUT
    printf("Updating RightMotor\n");
#endif
    #ifdef ON_BEAGLEBOARD
     if(rightMotorUpdated) {
          pwm_config_dutyCycle(rightMotor, resolution,
currentRightMotorVal, desiredRightMotorVal);
          rightMotorUpdated = 0;
          currentRightMotorVal = desiredRightMotorVal;
     }
    #endif
    //}

}

void motorBehavior() {
    //double desiredLeftMotorVal, desiredRightMotorVal;
    if (!leftBump && !rightBump) {
     if (s.calcLeftSensorVal < 18 && s.calcRightSensorVal < 18) {
          //system(LeftMotorReverseF);
          //system(RightMotorReverseF);
          desiredLeftMotorVal = LeftMotorReverseF;
          desiredRightMotorVal = RightMotorForwardF;
          //printf("Moving Backwards\n");
     }
     if (s.calcLeftSensorVal < 18 && s.calcMiddleSensorVal < 18 &&
s.calcRightSensorVal > 18) {
          //system(LeftMotorForwardF);
          //system(RightMotorReverseF);
          desiredLeftMotorVal = LeftMotorForwardF;
          desiredRightMotorVal = RightMotorReverseF;
          //printf("Hard Right\n");
     }
     if (s.calcLeftSensorVal < 18 && s.calcMiddleSensorVal > 18 &&
s.calcRightSensorVal > 18) {
          //system(LeftMotorForwardF);
          //system(RightMotorReverse);
          desiredLeftMotorVal = LeftMotorForwardF;
          desiredRightMotorVal = RightMotorReverse;
          //printf("Soft Right\n");
     }
     if (s.calcLeftSensorVal > 18 && s.calcMiddleSensorVal < 18 &&
s.calcRightSensorVal < 18) {
          //system(LeftMotorReverseF);
          //system(RightMotorForwardF);
          desiredLeftMotorVal = LeftMotorReverseF;
          desiredRightMotorVal = RightMotorForwardF;
          //printf("Hard Left\n");
     }
     if (s.calcLeftSensorVal > 18 && s.calcMiddleSensorVal < 18 &&
s.calcRightSensorVal > 18) {
          //system(LeftMotorReverseF);
          //system(RightMotorReverseF);
          desiredLeftMotorVal = LeftMotorForwardF;
          desiredRightMotorVal = RightMotorReverseF;
          //printf("Move Backwards, Random Left or Right\n");
     }
     if (s.calcLeftSensorVal > 18 && s.calcMiddleSensorVal > 18 &&
s.calcRightSensorVal < 18) {
          //system(LeftMotorReverse);
          //system(RightMotorForwardF);
          desiredLeftMotorVal = LeftMotorReverse;
          desiredRightMotorVal = RightMotorForwardF;
          //printf("Soft Left\n");
     }
     if (s.calcLeftSensorVal > 18 && s.calcMiddleSensorVal > 18 &&
s.calcRightSensorVal > 18) {
          //system(LeftMotorForwardF);
          //system(RightMotorForwardF);
          desiredLeftMotorVal = LeftMotorForwardF;
          desiredRightMotorVal = RightMotorForwardF;
          //printf("Moving Forward\n");
     }
     if (BLUE_BALL) {
          //FOUND BLUE BALL TURN RIGHT
          desiredLeftMotorVal = LeftMotorForwardF;
          desiredRightMotorVal = RightMotorReverse;
          leftMotorUpdated = 1;
          rightMotorUpdated = 1;
          leftMotorUpdate();
          rightMotorUpdate();
          sleep(1);
          desiredLeftMotorVal = LeftMotorForwardS;
          desiredRightMotorVal = RightMotorForwardS;
          printf("Found Blue Ball, Soft Right\n");
          BLUE_BALL = 0;
      }

     }
     else {
      if (leftBump) {
          desiredLeftMotorVal = LeftMotorForwardF;
          desiredRightMotorVal = RightMotorReverseF;
      }
      if (rightBump) {
          desiredLeftMotorVal = LeftMotorReverseF;
          desiredRightMotorVal = RightMotorForwardF;
      }
     }
     if (desiredLeftMotorVal != currentLeftMotorVal) {
      printf("LeftMotorVal: %f\n", desiredLeftMotorVal);
      leftMotorUpdated = 1;
     }
     if (desiredRightMotorVal != currentRightMotorVal) {
      printf("RightMotorVal: %f\n", desiredRightMotorVal);
      rightMotorUpdated = 1;
     }
}
void *bumpSwitch_init(void*) {
     initBUMPERS();
     while(1) {
      if (readLEFTBUMPER()) {
          printf("LEFTBUMPER Triggered!\n");
          leftBump = 1;
      }
      else {
          leftBump = 0;
      }
      if (readRIGHTBUMPER()) {
          printf("RIGHTBUMPER Triggered!\n");
          rightBump = 1;
      }
      else {
          rightBump = 0;
      }
     }
}

int kick() {
     system("echo 139 > /sys/class/gpio/export");
     system("echo output > /sys/class/gpio139/direction");
     printf("Echoed 139\n");
     system("echo low > /sys/class/gpio/gpio139/direction");
     system("echo high > /sys/class/gpio/gpio139/direction"); }
```