UNIVERSITY OF FLORIDA

Final Report

Bob

Patrick Neal

12/9/2015

Intelligent Machine Design Lab
EEL5666C
Email: neap@ufl.edu
TAs: Andy Gray
Jake Easterling
Instructors: Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz

Table of Contents

Abstract

This report is about Bob; a robot that picks up blocks and move them to specific locations based on their color codes. Bob is able to specifically place the blocks by localizing from color code beacons. Bob uses one Pixy CMUcam5 to detect blocks on the ground and one Pixy on a pan/tilt mechanism to localize, .Bob localizes using the generalized geometric triangulation algorithm. At this time Bob is pretty much just remote controlled.

Introduction

My objective for this robot, Bob, is for it to be able to identify and collect objects and sort them by their identifier. The robot will work within a ~25 sq.-ft. arena with corners marked with beacons. A CAD model rendering of a beacon can be seen in Figure 1. Within this arena it will seek these objects and grab them with a claw like gripper. After it has picked up the object it will localize itself with respect to the beacons and deliver the object to the appropriate area. Figure 2 shows the beacons relative positions.

This paper will discuss how the components will interface with each other and what role they play in the system.

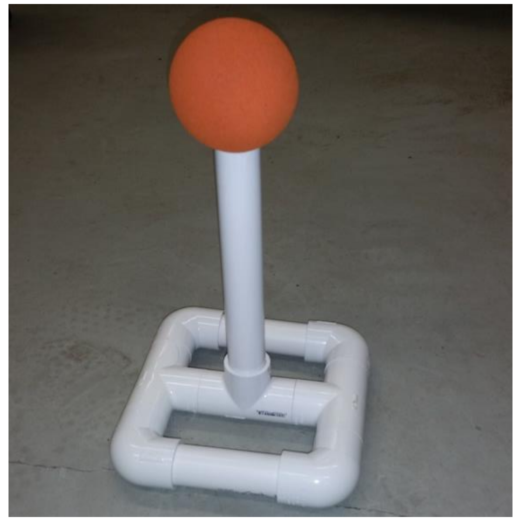


Figure 1 on the left shows the proposed beacon design. The right image shows a finished beacon. These are old pictures of before I decided to use color codes to track instead of balls. The balls that were to be used are ~3.25 inches in diameter which makes the total height of the beacon ~13 inches tall.

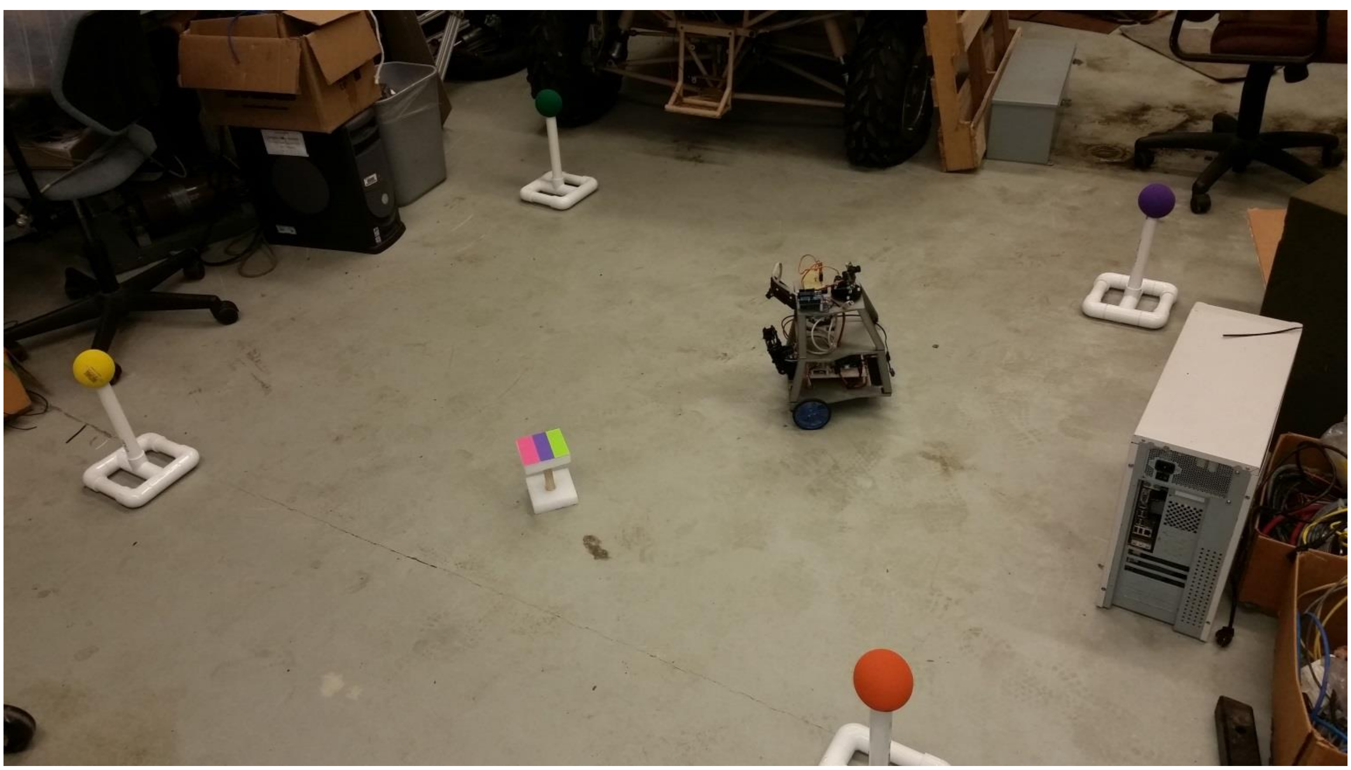


Figure 2 shows the space that Bob will complete its task. The multi-colored object is what Bob will collect and drop off.

Integrated Systems

(Incomplete) The Odroid C1 will receive the visual data from the pan/tilt Pixy camera, the forward facing Pixy camera through the Arduino Uno, and the Arduino Mega. The data is then used to control the behaviors of the robot. The Arduino Mega will interface with all other sensors and provide PWM signals for all motors. Figure 3 shows a diagram of the connections between the processors and sensors. The Arduino Mega will read the encoder data and keep track of Bob's estimated position since last update using dead reckoning. When the robot localizes from the beacons the Odroid C1 will reset the state changes on the Arduino Mega.

The obstacle avoidance behavior will be located on the Arduino Mega to allow the robot to do basic functions without the need for the Odroid C1. To obstacle avoidance behavior will always override any movement commands that are sent by the Odroid C1. This will ensure that Odroid C1 does not drive the robot into an object it does not know about.

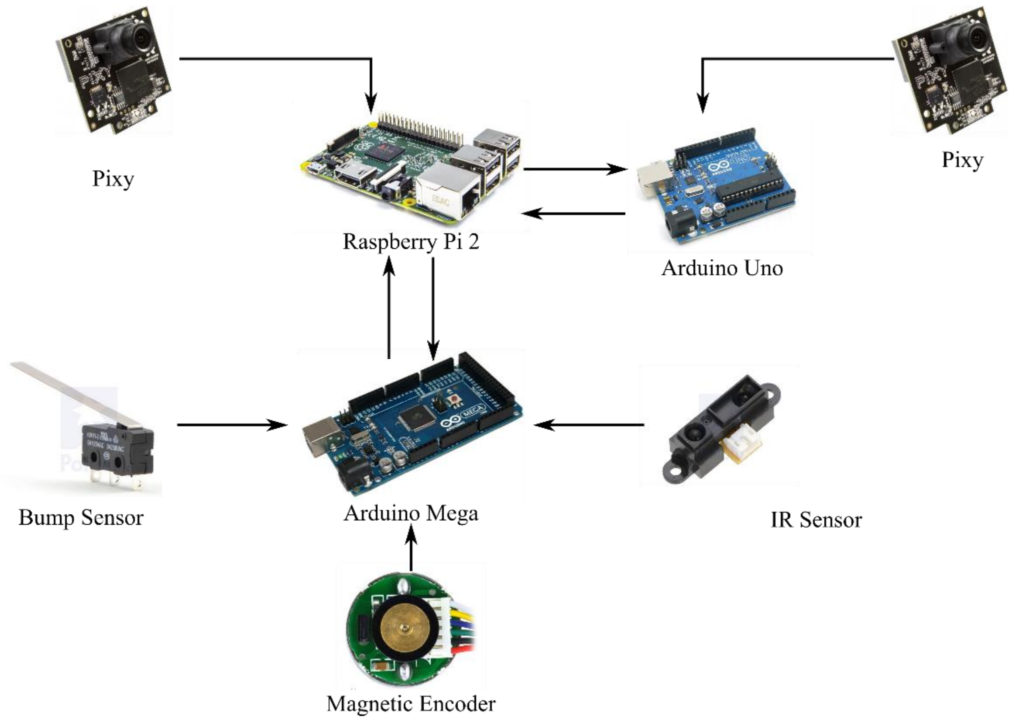


Figure 3 shows the sensors used and an overview of how they interface with the other hardware. The Raspberry Pi 2 was replaced with an Odroid.

The Pixy CMUcam5 is the camera I choose to originally perform stereo vision with. But I chose not to do stereo vision because of problems with the Pixy library. The built in ability to identify colored objects and report their centroids eliminated made them ideal because of time constraints. Teaching the Pixy a color only requires pressing the button on the top or the use of their program Pixymon. After it has learned a color they perform color clustering techniques to determine objects. The Pixy outputs the calculated centroid coordinates for all objects that meet the desired color sets. The forward facing Pixy connected to the Arduino is used to track specific color codes corresponding to the objects to pick up and the pan/tilt Pixy is used to find the beacons.

Mobile System

The mobile system was designed with three levels: the first level is the basic mobile platform and it will hold the Arduino, battery, and related electronics, the second level will hold the Odroid C1 and related power electronics, the third level is where the vision sensors are mounted along with an Arduino Uno to read one of them. Figure 4 shows a picture of the completed Mobile System. In assembling the mobile system I made liberal use of 3M dual lock to attach most of the components to the chassis. This allowed to easy dissembling if I need to change anything.

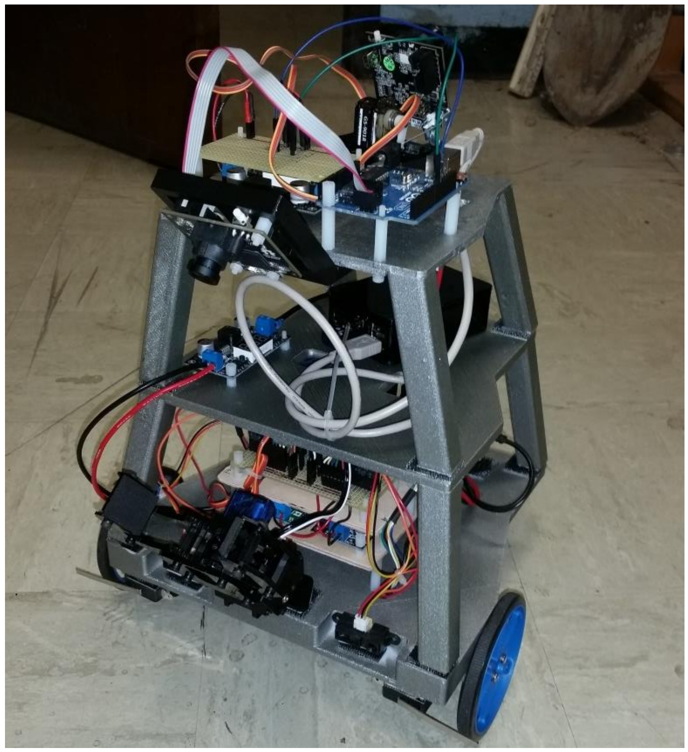


Figure 4 shows the finished mobile system. All gray structure is 3D printed with ABS plastic except the top level. The top level is two sheets of wood stacked on each over. The bump switches can be seen mounted under the front of Bob.

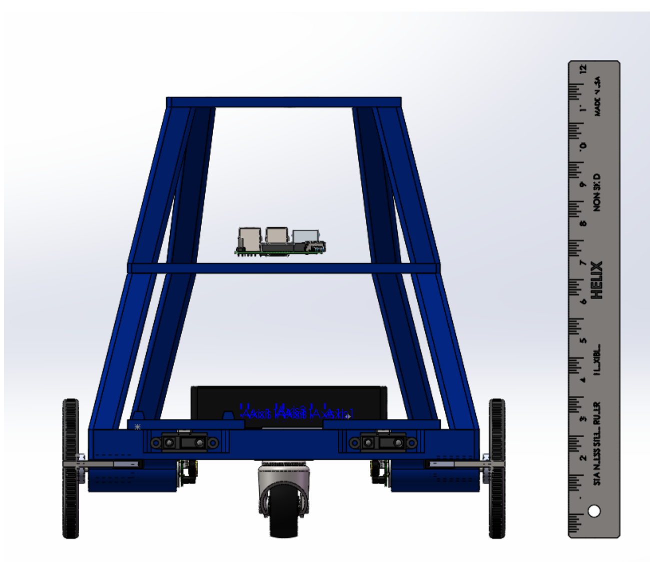


Figure 5 shows the front of the robot and a 12 inch ruler for scale.

Bottom Level

The mobile platform of the robot utilizes differential steering. There are two driven wheels near the front of the platform and a single caster wheel in the rear. There is also a Robot Geek gripper attached between the IR sensors. The gripper will be used to manipulate the colored objects. Most of the power electronics is located on this level. This includes: a 3S 5000maH Venom LiPo battery, a 30 amp circuit breaker, a 30 amp toggle switch, a 5 circuit terminal block, and a DROK buck converter.

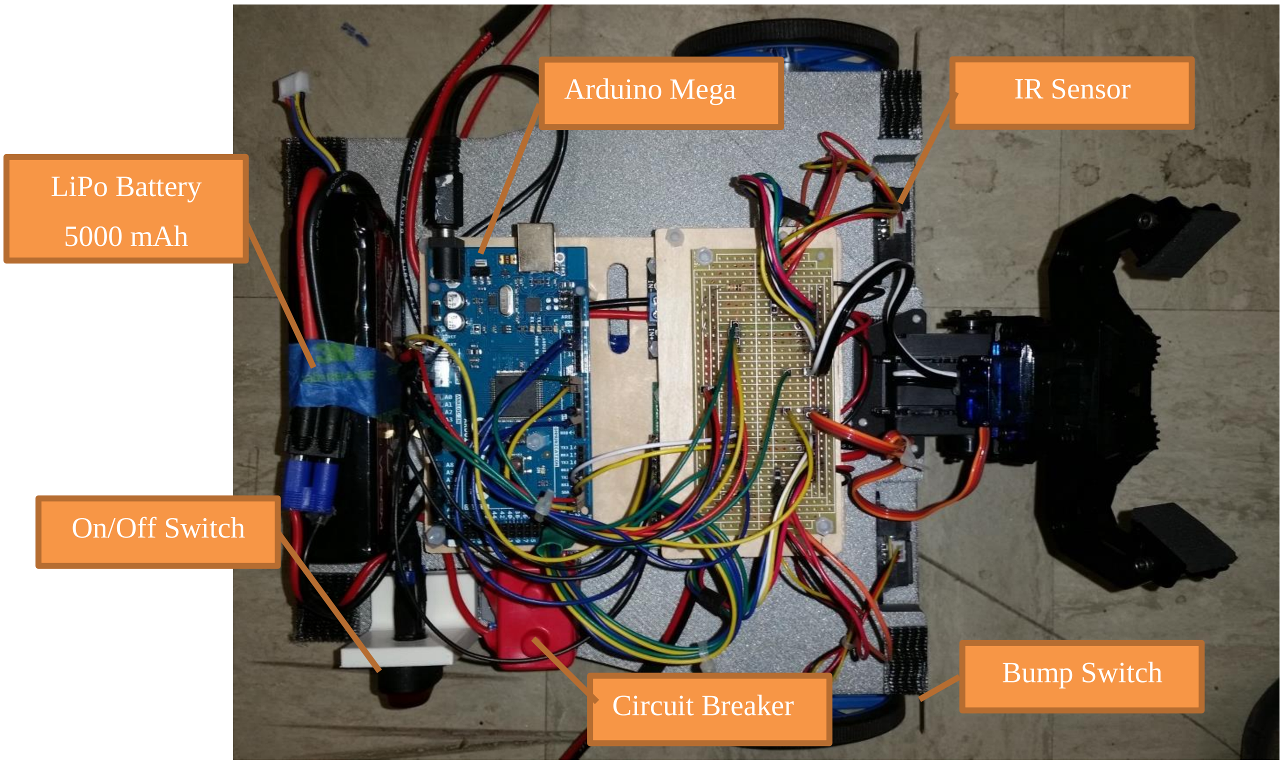


Figure 6 shows the bottom level of Bob. There is a Pololu VNH5019 motor driver and buck converter under the proto board on the left side of image.

Middle Level

Middle level is just to hold the Odroid C1 and be close to the top level. Also holds the buck converter for power the Odroid C1. I left a decent amount of space for the excess wires from the USB connections. The shorted USB A to USB B I could find were three feet.

Top Level

The top level will hold both Pixy cameras used. One is attached to a pan and tilt mechanism to see the beacons while the other is rigidly fixed facing down in front of Bob to see the objects. For some reason the orientation of the servo motors in the pan/tilt mechanism causes them to interfere with one another when both are being used. I decided to only use the pan because of this. The forward facing Pixy is connected to the Odroid C1 through the Arduino Uno because of problems with serial communications between multiple Pixys using the Pixy library.

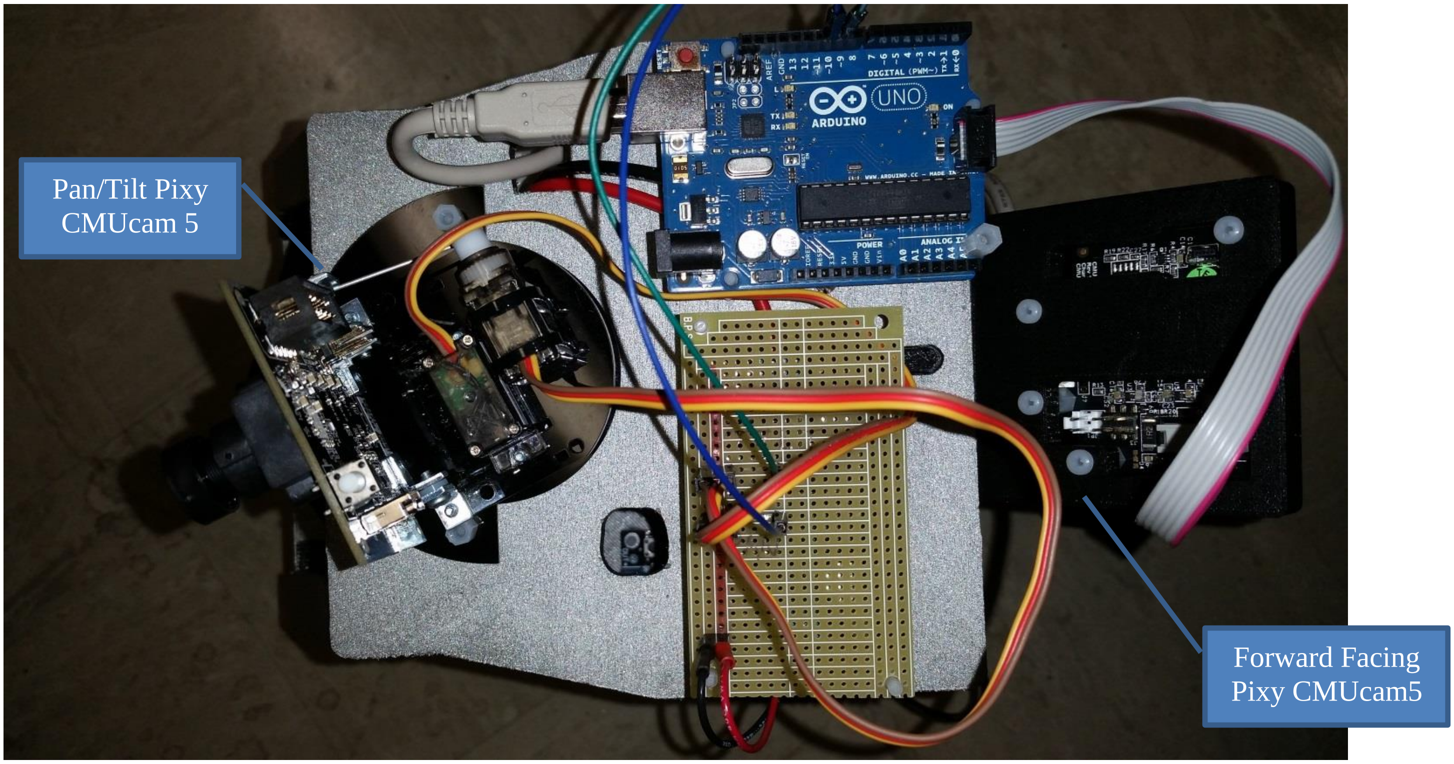


Figure 7 shows the components on the top level of Bob. Under the proto board there is a buck converter that supplies 6V to the pan/tilt servo motors.

Actuation

The actuation for my robot is fairly simple. It consists of two 6 V DC gearmotors; two of them are for movement. The motors should be able to provide enough torque at slow speeds. These motors will be controlled by Pololu motor driver board connected directly to the battery. I will also use three hobby servo motors. One of them will be responsible for panning the camera system while the other two will be used for the mechanism to grip the object.

Name: 98.78:1 Metal Gearmotor 25Dx54L mm HP

Purpose: Main drive motors for robot locomotion.

Specs:

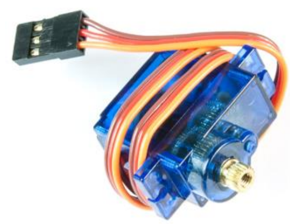
- 6V
- No load RPM of 100 with 450mA draw
- 160 oz.-in drawing 6.5A at stall

Supplier: Pololu Robotics & Electronics

Controller: A PD controller is used to control the velocity of each wheel.

This allows Bob to move in relatively straight lines and pivot about the point centered between the wheels.

Name: FS90MG Servo Motor



Purpose: Used to open and close the gripper.

Specs:

- **No-load Speed:** 0.10sec/60^o
- **Stall Torque:** 25.04 oz.-in
- **Operating angle:**180^o
- **Weight:** 14g
- **Size:** 0.913x0.492x0.866 in

Supplier: Trossen Robotics

Name: Robot Geek RG-SRV180 Servo Motor

Purpose: Used to raise and lower the gripper

Specs:

- **No-load Speed:** 0.23sec/60^o, 43 RPM
- **Stall Torque:** 118.2 oz.-in
- **Operating angle:** 180^o
- **Weight:** 60g
- **Size:** 1.18 x 1.77 x 2.01 in

Supplier: Trossen Robotics

Sensors

The task was to avoid objects while searching for certain colored Lego brick stacks and to localize from colored beacons. To accomplish the obstacle avoidance tasks I choose to use IR sensors and bump switches. The two IR sensors are forward facing recessed about an inch into the chassis. The IR sensors will detect most objects in front of the robot and the bump sensors will detect if Bob has hit something that is located outside of the IRs. I also use current measurements to determine if Bob has hit something it did not catch with IR and bump sensors.

To localize I will used angles measurements obtained finding beacons with the Pixy and recording the servo position. Doing this measurement for three beacons Bob can triangulate its position.

Name: Sharp Analog Distance Sensor

Purpose: Used to avoidance obstacles located in front of Bob.

Specs:

- 4-30 cm range
- Operating voltage of 4.5-5.5 V
- Output voltage differential over range: 2.3V]
- Update period of 16.5 ±4 ms

Model Number: GP2Y0A41SK0F

Supplier: Pololu Robotics & Electronics

Name: KW8-Series Micro Switch (Old Picture)

Purpose: Last ditch attempt at obstacle avoidance. When the switch is pressed it means that Bob has hit something on the left or right front.

Specs:

Supplier: Amazon

Name: Hall effect magnetic encoder

Purpose: Used to measures shaft rotations to estimate the robots position and also the direction the cameras are looking

Specs:

- 48 CPR
- 5 V operating voltage

Supplier: Pololu Robotics & Electronics

Dead Reckoning Equation:

These equations assume only point and shoot motion no complex paths. Variables denoted by an s are the distance traveled by each wheel.

$$s = \frac{s_R + s_L}{2}$$

Turns are performed first. b is the distance between wheels.

$$\theta_{k+1} = \theta_k + \frac{s_R - s_L}{b}$$

Then the straight distance is calculated.

$$x_{k+1} = x_k + \bar{s} * \cos(\theta_k)$$

$$y_{k+1} = y_k + \bar{s} * \sin(\theta_k)$$

Name: Pixy CMUcam5

Purpose: Used in the proposed stereo vision system

Specs:

- Video resolution of 640x400
- Colored object detection at 50 fps
- Lots more technical specs at: http://www.cmucam.org/projects/cmucam5/wiki/Introduction_and_Background

Supplier: Amazon - Charmed Labs

Behaviors

- Ready
  - Bob is ready to start
- Search/Wander
  - Bob is moving around looking for objects
- Align with object/Pickup
  - Bob lowers gripper and aligns with the object in order to pick up. Then approaches the object and grabs it
- Localize
  - Bob searches for the beacons to determine its position within the arena
- Seek & Deposit
  - Bob maneuvers to desired position and drops the object.

There is nothing particularly special about most of these behaviors. I will go into more detail on the method used to localize though. The method of localizing uses the Generalized Geometric Triangulation Algorithm. This algorithm was developed by [1] and it uses three beacons with known absolute locations to triangulate the position of the thing taking measurements. The measurements consist of the angles from the orientation of the object to each of the beacons. They discuss more in the paper the benefits of using their algorithm versus other geometric triangulation algorithms.

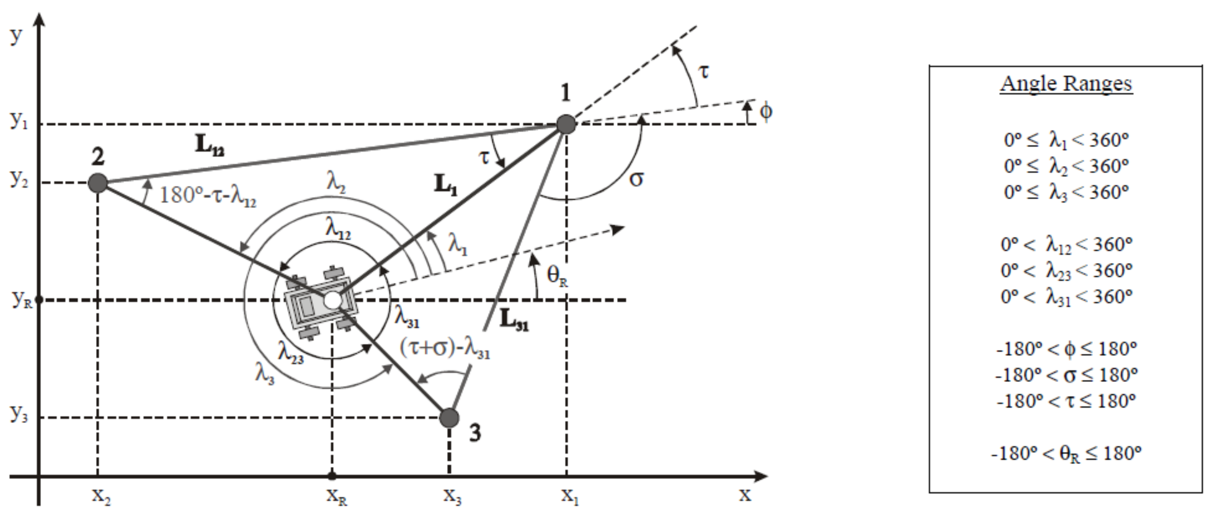


Figure 8 is figure from [1] that shows the different angles that are measured/calculated in order to complete the algorithm. One benefit the mention is the paper is that the beacons do not need to be in a specified order.

Experimental Layout and Results

IR output versus distance test.

This test was performed in order to obtain a mapping between the ADC output on the Arduino and the real distance. The distance was measured using a measuring tape and a piece of white cardstock to block the IR sensors. The lighting was indoors under florescent lights. Data closer than 3 inches was not collected because distance measurement become indeterminate (they become two to one mappings). The IR sensors are also recessed into the chassis to minimize objects being detected within this region. Tables Table I and Table II show the numbers used to determine the mapping and Figure 9 and Figure 10 are just graphs of this data along with the fitted equation.

Table I shows the numerical values recorded during the test for the left IR sensor.

Analog Input Mapping	304	195	130	100	80	90
Distance (inches)	3	5	7	9	11	13

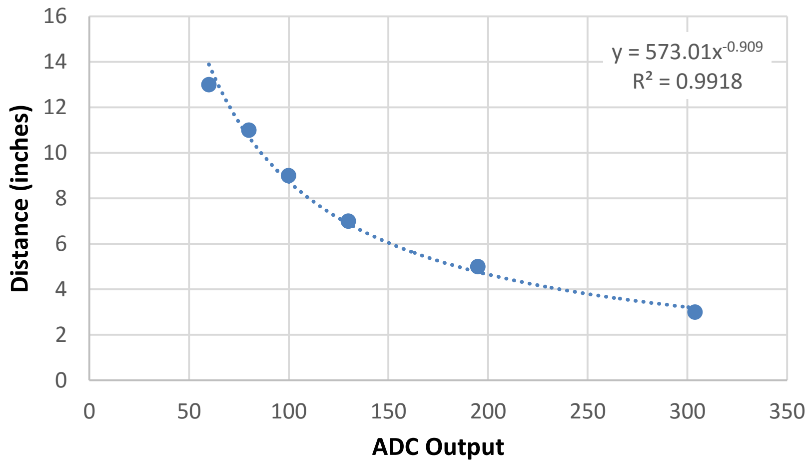


Figure 9 shows the mapping between the ADC output and distance. Data on the other side of the "Hill" for IR sensor was not recorded. The equation resulting from a power fit is shown in the upper right.

Table II shows the numerical values recorded during the test for the right IR sensor.

Analog Input Mapping	307	195	140	110	105	80
Distance (inches)	3	5	7	9	11	13

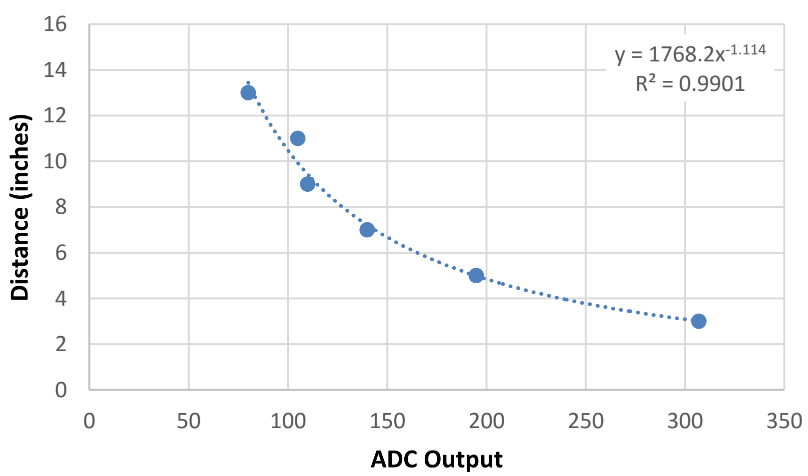


Figure 10 shows the mapping between the ADC output and distance. Data on the other side of the "Hill" for IR sensor was not recorded. The equation resulting from a power fit is shown in the upper right.

Preliminary illuminated orb test:

For this experiment I wanted to see investigate using illuminated objects for color detection. I wanted to see if this was a way to get good object detection while minimizing the ambient noise. I performed this test by placing a ping pong ball on top of a LED flashlight. The lighting conditions were standard for florescent bulbs. Then I took a screenshot at 7, 13 and 30 inches. Figure 11 shows the experimental setup.

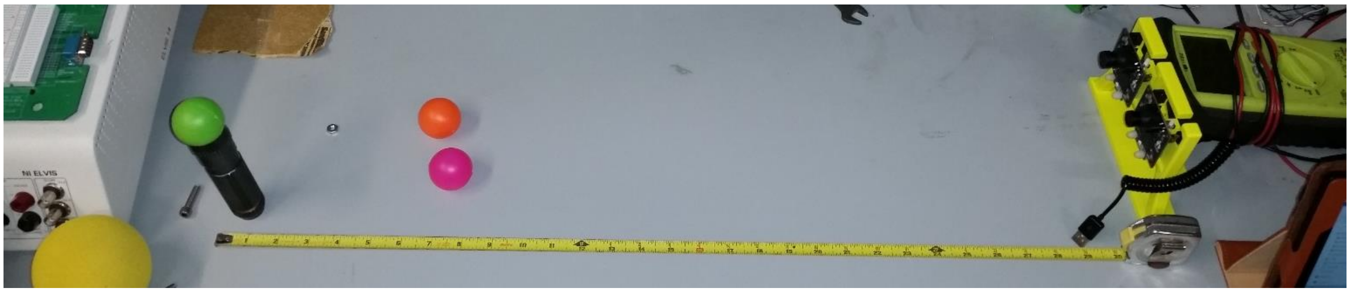


Figure 11 shows my experimental setup for this test. On the left you can see the flashlight with a green ping pong resting on top of it. To the right in the yellow stand is the Pixy camera used for this test. I used a brightness of 25.

Table 3 shows the Pixy tuning parameters I used to achieve the results from the images below. I did not do much tuning with these.

	Orange	Green	Pink
Pixy Tuning Parameter	3.78	7.4	3.96

Figure 12, Figure 13, and Figure 14 below show the tracking achieved using a ping pong ball with a diameter of 1.5”. The tuning parameters are shown in Table 3.

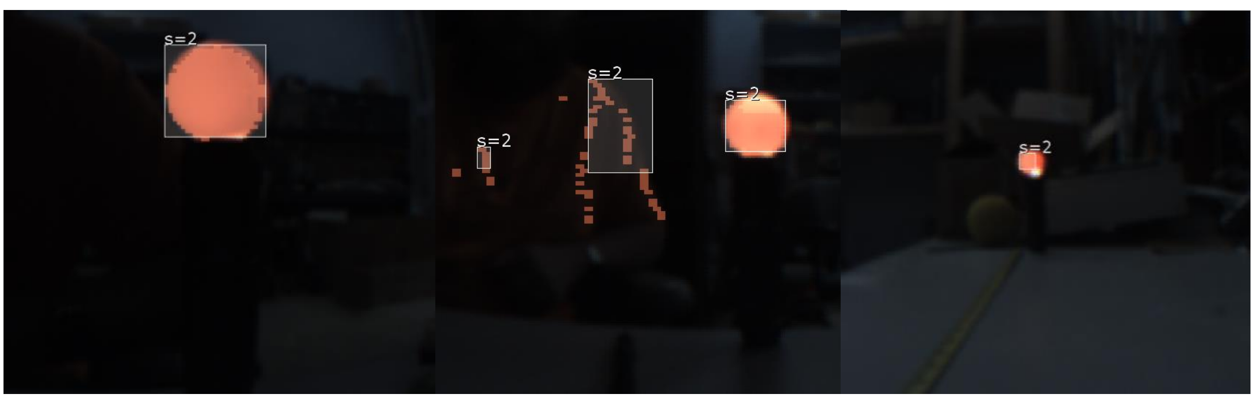


Figure 12 shows the tracking of the orange ping pong at distances 7, 13, 30 inches respectively. You can see the camera picking up spots from my neon orange shirt in the middle image.

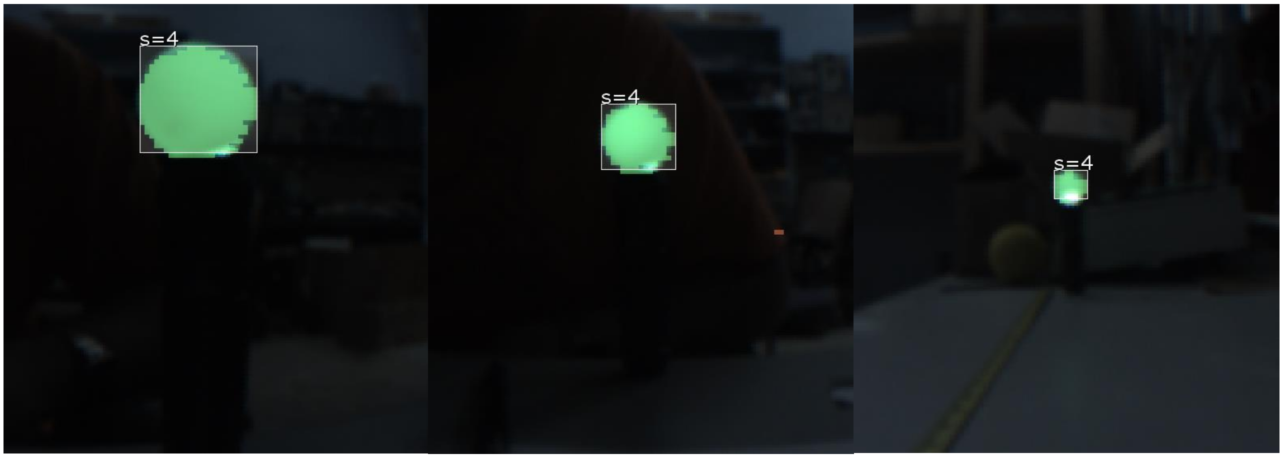


Figure 13 shows the tracking of the lime green ping pong at distances 7, 13, 30 inches respectively.

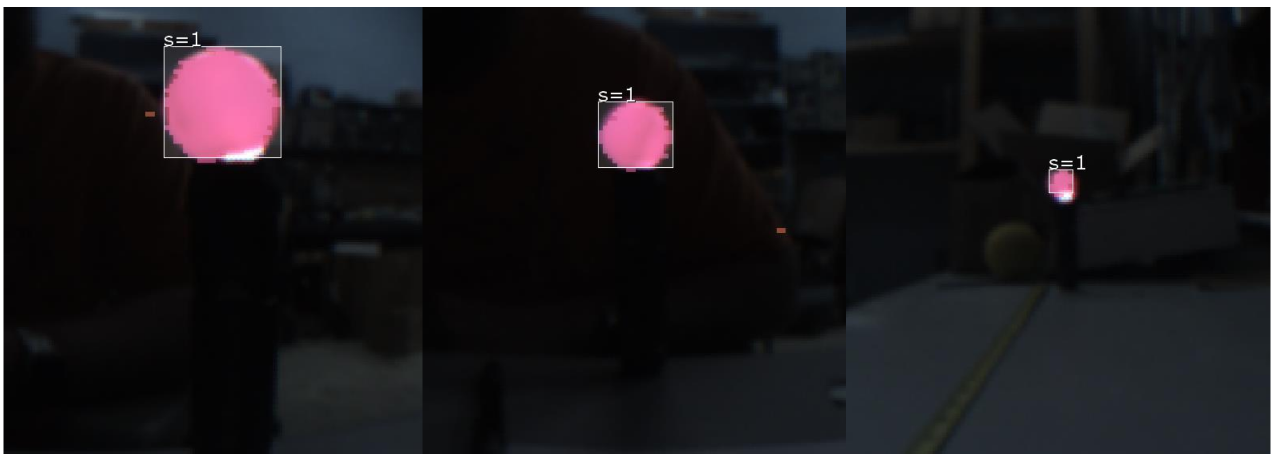


Figure 14 shows the tracking of the lime green ping pong at distances 7, 13, 30 inches respectively.

This experiment yielded some positive results. The Pixy mostly did not pick up background objects even when the tracking thresholds were turned up. The bad part of this experiment was that the light was coming from beneath the object. This caused a non-uniform color over the ball with the bottom being closer to white.

From this experiment I decided to purchase 3.25” Diameter illuminated orb, shown in Figure 15, to use on the beacons. I chose 30 inches to test the ping pong because the ball’s diameter is about half of these larger orbs. So I should be able to get about a 5 feet range from these new orbs.

Figure 15 shows examples of the orbs I plan on using for the beacons. They are self-contained with internal light and batteries.

Update: Turns out these orbs were too dim to be useful in normal lighting conditions.

Conclusions

At the time of this report Bob is basically only remote controlled with an incomplete obstacle avoidance and incomplete dead reckoning. I severely under estimated the time to code all the little things for Bob (serial communication, logic required to use dead reckoning, etc.). I spent too much time on the mechanical/electrical design of the robot. I should have spent more time early on working with serial communications. Also I had two Raspberry Pi 2 fail because of power problems. I do not think they should be used for these battery powered applications. They are also a lot less efficient computationally than the stated processor speeds. I would choose a processor more meant for robots/battery powered operation. I was looking into the Intel Edison which destroys both the Odroid C1 and R pi 2 in most benchmarks done by sparkfun besides graphics. It loses the graphics because it does not video output capabilities. It also can be integrated into an Arduino breakout board and can be used as an Arduino along with a full Linux OS.

Documentation

[1] João Sena Esteves, Adriano Carvalho, Carlos Couto, "Generalized Geometric Triangulation Algorithm for Mobile Robot Absolute Self-Localization", not sure where they published it

Appendices

Components:

Name: Odroid C1

Purpose: Used to do the high level control and decision making of Bob. Also used to perform localization algorithm.

Specs:

- 1.5 GHz quad-core ARM Cortex-A7 CPU
- 1GB DDR3 SDRAM
- 4 USB ports
- 40 GPIO pins

More detailed specifications on AmeriDroid page.

Supplier: AmeriDroid

Name: Arduino Mega 2560

Purpose: Used to do the high level control and decision making of Bob

Specs:

- Microcontroller: ATmega2560 `16 MHz Clock
- Operating Voltage: 5V
- Input Voltage: 7-12V
- 54 Digital I/O Pins(15 of which are PWM)
- 16 Analog Inputs

More detailed specifications on Arduino page.

Supplier: Arduino

Name: Arduino Uno Rev 3

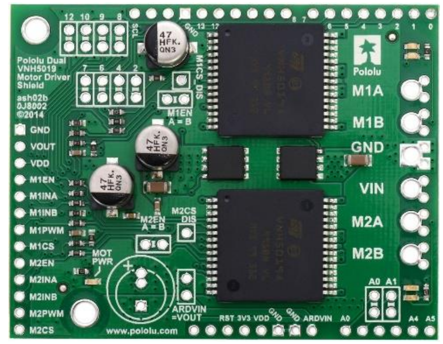
Purpose: Possible to be used to control the Pan and tilt camera and read inputs from a Pixy to determine bricks

Specs:

- Microcontroller: ATmega328P `16 MHz Clock
- Operating Voltage: 5V
- Input Voltage: 7-12V
- 14 Digital I/O Pins(6 of which are PWM)
- 6 Analog Inputs

More detailed specifications on Arduino page.

Supplier: Arduino



Name: Dual Motor Driver Shield for Arduino

Purpose: Used to power the gear motors and measure current draw from the motors

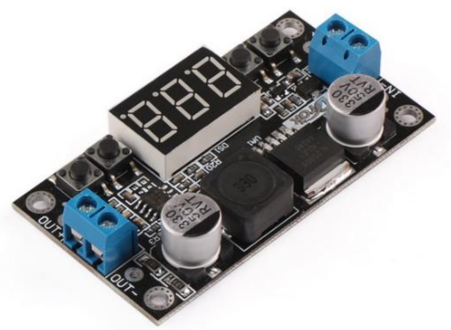
Specs:

- Operates from 5.5-24V
- continuous 12 A (30 A peak) per motor, or 24 A (60 A peak) to a single motor
- Inputs compatible with both 5V and 3.3V systems (logic high threshold is 2.1 V)
- PWM operation up to 20 kHz, which is ultrasonic and allows for quieter motor operation
- Current sense voltage output proportional to motor current (approx. 140 mV/A)

More detailed specifications on Pololu site.

Model Number: VNH5019 Revision

Supplier: Pololu



Name: DC Buck Converter

Purpose: Used to supply 5V to servo motors and to power the Odroid C1

Specs:

- Input Voltage: DC 5-35V
- Output Voltage: 0-33V (continuously adjustable, the input voltage must be 1V higher than the output voltage) Change voltage output by push buttons
- Output Power: 30W max.
- Output Current: 1.5A, Max. 3A, above 1.5a please adding heatsink
- Output Voltage Setting Resolution: 0.1V

More detailed specifications on the Amazon page.

Model Number: LM2596 **NC**

Supplier: Amazon - DROK

Name: Venom LiPo Battery

Purpose: Used to power all electronics on Bob.

Specs:

- 3 Cells 11.1 V
- 5000mAh capacity/55.5 Watt hours
- 20C discharge rating, 1C charge rating
- Universal Plug System

Model Number: 1582

Supplier: Amazon – Venom RC

Name: SPST Oval Rocker Switch

Purpose: Used to turn power on and off to all electronics.

Specs:

- Rated for 30 amps at 12 V DC
- Button lights up when On

Supplier: RadioShack

Name: Sea-Dog 420853-1 Resettable Circuit Breaker with Cover

Purpose: Used to hopefully protect electronics if large current draw from the battery happens

Specs:

- Rated for 30 Amps
- Can be used in 12 or 24 V systems
- Resettable if breaker is thrown.
- Has easy to use screw terminals

Supplier: Amazon – Sea Dog Line

Name: Generic 5 Circuit Terminal Block

Purpose: Used to distribute power form the battery to the various electronic components.

Specs:

- Jumpers can be used to create shared terminals

Not the correct picture.

Supplier: Found in the CIMAR lab.

Low Level Controller:

```
//-------Include Libraries-----------------------------------
#include <Servo.h>
#include <DualVNH5019MotorDriver.h> // For use with the Dual motor drivers from Pololu
#include <Encoder.h>
#include <PID_v1.h>
#include <Messenger.h>

//-------Define variables here-------------------------------
// Behavior Variable
```

```
int activeBehavior = 0;

// ---PWM Pin Setting---

// PWM pins for the two drive motors
const int PWMDrivePin_Left = 11;    // Timer 1 16-bit
const int PWMDrivePin_Right = 12;   // Timer 1 16-bit

// Additional PWM Pins for Manipulator. May not be used
const int PWMWristPin = 7;
const int PWMGraspPin = 6;

// ---Pololu Motor Pins---
const int Drive_INA1=22;     // Left Drive Motor
const int Drive_INB1=23;     // Left Drive Motor
const int Drive_EN1DIAG1=24; // Left Drive Motor

const int Drive_INA2=26;     // Right Drive Motor
const int Drive_INB2=27;     // Right Drive Motor
const int Drive_EN2DIAG2=25; // Right Drive Motor

// ---Sensor Pins---

// Bump Sensors
const int leftBumpPin = 2;            // Digital Pin 2, 0 interrupt pin for Mega 2560
const int rightBumpPin = 3;           // Digital Pin 3, 1 interrupt pin for MEga 2560

// Encoder Input
const int leftEncoderAPin = 21;   // Digital Pin 21, 2 interrupt pin for Mega 2560
const int leftEncoderBPin = 20;   // Digital Pin 20, 3 interrupt pin for Mega 2560

const int rightEncoderAPin = 19;  // Digital Pin 19, 4 interrupt pin for Mega 2560
const int rightEncoderBPin = 18;  // Digital Pin 18, 5 interrupt pin for Mega 2560

// IR sensors
const int leftIRPin = 0;          // A0
```

```
const int rightIRPin = 1;        // A1

// Current Sensors
const int leftCurrentPin = 2;    // A8, CS1
const int rightCurrentPin = 3;   // A9, CS2

// ---Sensor sampling periods---
const unsigned long irPeriod = 100;      // Sampling period for IR Sensors (ms)
const unsigned long currentPeriod = 100; // Sampling period for current sensor (ms)
const unsigned long encoderPeriod = 50;  // Sampling period for encoder sensor (ms)
const unsigned long serialPeriod = 100;  // Sampling period for serial read (ms)
const unsigned long stoppedPeriod = 150; // Sampling period for stopped measurement (ms)



// Used for sensor sampling times
unsigned long currentMillis;            // Stores the current time reading in milliseconds
unsigned long previousMillis_IR=0;      // Stores the previous time the IR sensors were read
unsigned long previousMillis_Current=0; // Stores the previous time the Current sensors were read
unsigned long previousMillis_Encoder=0; // Stores the previous time the encoder were read
unsigned long previousMillis_Serial=0;  // Stores the previous time the serial weas read
unsigned long previousMillis_Stopped=0; // Stores the previous time if the robot was stopped

// ---Constants---
const int eps = 0.75;

// Encoder Conversion Constant
const double C = (3.54*3.14159)/4741.41;

// ---Define non-constant variables---
int inByte = 0;     // Variable that will store incoming byte from serial

int irLeft[4] = {0, 0, 0, 0};     // Stores the past 3 Left IR Readings for use in an average
int irRight[4] = {0, 0, 0, 0};    // Stores the past 3 Right IR Reading for use in an average
int irValue;                  // Temporary value to store mesured IR reading
float irLeftAvg;                 // Used to store left IR average reading
float irRightAvg;                 // Used to store right IR average reading
```

```
int currentLeft[4] = {0, 0, 0, 0};   // Stores Left Drive Motor Current Reading
int currentRight[4] = {0, 0, 0, 0};  // Stores Right Drive Motor Current Reading
int currentValue;                    // For current measurement (amps)
float currentLeftAvg;                // Used to store the average current sensor reading for the left motor
float currentRightAvg;               // Used to store the average current sensor reading for the right motor

// Encoder Counting Variables
long leftOldPosition;
long rightOldPosition;
long leftNewPosition = 0;
long rightNewPosition = 0;

// PID input variables
double leftSetpoint, leftInput, leftOutput;
double rightSetpoint, rightInput, rightOutput;

boolean leftDone = false, rightDone = false;
double leftOffset = 0.50, rightOffset = 1.25; // Distance offsets to account stopping time.

// PID Tuning Paramters
double lKp = 1.75, lKi = 0, lKd = 1.25;
double rKp = 1.75, rKi = 0, rKd = 1.25;

double K = 1;

// Serial Communications stuff
double temp = 0;
boolean newBehavior = false;
boolean newDistance = false;   // Signifies if a new command has been recieved
boolean newWristGraspCmd = false;   // Signifies if a new command has been recieved
boolean newRequest = false;
boolean newRobotSpeed = false;
boolean OAoff = false;             // To turn obstacle avoidance on or off
boolean gripOff = false;           // To turn gripper motion off
```

```
Messenger piMessage = Messenger(':');

int wristCmd = 160, graspCmd = 130;

int leftDistance = 0, rightDistance = 0;
int requestState = 0, requestComplete = 0, oaState = 0;
boolean oaOverride = false;
boolean isStopped = true;
double robotSpeed = 5;

// State Update Variables
float dx = 0, dy = 0, dtheta = 0;
int motionDirection = 0, oldMotionDirection = 0;
double leftStartPoint, rightStartPoint;



// Sensor Flags
boolean irFlag =false;              // Will be set true if an IR condition is met
  int irRecomnd = 0;                // Recommendation will be set depending on specific combintaion of IR readings
boolean currentFlag = false;        // Will be set true if Current sense condition is met
  int currentRecomnd = 0;           // Will be set depending of specific combinations
volatile boolean bumpFlag = false;  // Will be set true if a bump sensor is triggered
  volatile int bumpRecomnd = 0;     // Will indicate whether left or right sensor was triggered




// Arbiter Variables
boolean actionLock = false;
boolean actionOverride = false;
unsigned long timerLockout = 750;
unsigned long actionTimer =0;
boolean Reverse = false;
boolean Turn = false;

//----Define Objects-----
```

```
// Define drive motor object
DualVNH5019MotorDriver driveMotors(Drive_INA1,Drive_INB1,PWMDrivePin_Left,\
Drive_EN1DIAG1,leftCurrentPin,Drive_INA2,Drive_INB2,PWMDrivePin_Right,Drive_EN2DIAG2,\
rightCurrentPin, 1);

// Define encoder object
Encoder leftEncoder(leftEncoderAPin, leftEncoderBPin);
Encoder rightEncoder(rightEncoderBPin, rightEncoderBPin);

/* Define PID object
 *  PID will take the velocity as an input. So the derivative will be calculated be calling the PID function
 */
PID leftPID(&leftInput, &leftOutput, &leftSetpoint, lKp, lKi, lKd, DIRECT);
PID rightPID(&rightInput, &rightOutput, &rightSetpoint, rKp, rKi, rKd, DIRECT);

// Define servo objects
Servo Wrist;
Servo Grasp;

// Test Variables
const String leftString = "Left IR Reading: ";
const String rightString = "Right IR Reading: ";

boolean readyBypass = false;

//------- Message parsing function -------------------------

void messageParse(){
  // This will set the variables that need to be changed
  // from the message
  if (piMessage.available()){
    activeBehavior = piMessage.readInt();
    if (activeBehavior != 9){ newBehavior = true;}

    leftDistance = piMessage.readDouble();
```

```
    rightDistance = piMessage.readDouble();
    if (leftDistance != 99 || rightDistance != 99){ newDistance = true; }

    wristCmd = piMessage.readInt();
    graspCmd = piMessage.readInt();
    if (wristCmd != 999 || graspCmd != 999){ newWristGraspCmd = true;}

    temp = piMessage.readDouble();
    if (temp != 99 && (temp >= 0 && temp < 15)){ robotSpeed = temp; }

    requestState = piMessage.readInt();
    if (requestState != 9) { newRequest = true;}

    // Repurpose this for some other information
    oaState  = piMessage.readInt();
    if ( oaState == 0 ) {OAoff = true;}
  }
}


//-------Setup Function----------------------------------

void setup()  // Needs to stay in setup until all necessary communications can be verified
{

  // Attach the servo objects to pins
  Wrist.attach(PWMWristPin);
  Grasp.attach(PWMGraspPin);

  Wrist.write(wristCmd);
  Grasp.write(graspCmd);

  // Initialize drive motor object
  driveMotors.init();

  //---- Bump Switches----
```

```
// Set interrupt pins to input
pinMode(leftBumpPin,INPUT);
pinMode(rightBumpPin,INPUT);

// Turn on pullup resistors
digitalWrite(leftBumpPin, HIGH);
digitalWrite(rightBumpPin, HIGH);

// Attached Interrupt pins
attachInterrupt(0, bumpLeft, RISING);        // Digital Pin 2
attachInterrupt(1, bumpRight, RISING);       // Digital Pin 3

//---- PID Settings ----
leftPID.SetSampleTime(50);
rightPID.SetSampleTime(50);

leftPID.SetOutputLimits(-100,100);
rightPID.SetOutputLimits(-100,100);

leftPID.SetMode(AUTOMATIC);
rightPID.SetMode(AUTOMATIC);

rightPID.SetControllerDirection(REVERSE);


// Initiliaze serial communications
Serial.begin(9600);          // set up Serial library at 9600 bps  boolean readyBypass = true;
piMessage.attach(messageParse);

while(1){
  // Set readyBypass to true to skip waiting for Odroid confirmation and button switch confimation
  if (readyBypass){break;}

  if (Serial.available() > 0){ inByte = Serial.read();}

  if (inByte == 115){
```

```
        Serial.println('g');
        break;
    }

    Serial.println('r');

    delay(100);
  }
}

//--------Main Loop--------------------------------------

void loop()
{

  currentMillis = millis(); // Program run time in milliseconds.

  // Read serial and call parser
  if (currentMillis - previousMillis_Serial > serialPeriod){
    previousMillis_Serial = currentMillis;
    while( Serial.available() ) piMessage.process(Serial.read());
  }

  // Act of behavior here like locking certain commands or something
  if( newBehavior){
    if(activeBehavior == 0){
      // Robot should be inactive
      // Obstacle avoidance should be off and the robot should not move.
      // Gripper should not move also
      OAoff = true;
      gripOff = true;
      }
    else if(activeBehavior == 1){
      // Search/Wander Behavior
      // Obstacle avoidance should be on, should recieve commands from Odroid
      OAoff = false;
```

```
      gripOff = false;
    }
    else if(activeBehavior == 2){
      // Align and Pickup behavior
      // Obstacle avoidance should be off, should recieve commands fro Odroid
      OAoff = true;
      gripOff  = false;
    }
    else if(activeBehavior == 3){
      // Deposit behavior
      // Obstacle avoidance should be on, robot should recieve commands from Odroid
      OAoff = true;
      gripOff = false;
    }
    else if(activeBehavior == 4){
      // Localize bahvior
      // Robot should stop moving, Obstacle avoidance should be turned off
      // Gripper should not be moving also
      OAoff = true;
      gripOff = true;
    }
  }

if(!gripOff){
 if (newWristGraspCmd){
   // Makes sure desired angles are acceptable
   if((wristCmd >= 40 && wristCmd <= 170) && wristCmd != 999){
     Wrist.write(wristCmd);
   }
   if((graspCmd >= 45 && graspCmd <= 135) && graspCmd != 999){
     Grasp.write(graspCmd);
   }
   newWristGraspCmd = false;
 }
}
 currentMillis = millis(); // Program run time in milliseconds. Used for sensor sampling.
```

```
//---- Distance Measurement IR Smart Sensor ----
if (currentMillis - previousMillis_IR >= irPeriod){
  previousMillis_IR = currentMillis;

  // Read in the left IR voltage and put into a buffer
  irValue = analogRead(leftIRPin);
  irLeft[0] = irLeft[1];
  irLeft[1] = irLeft[2];
  irLeft[2] = irLeft[3];
  irLeft[3] = irValue;
  delay(1);

  // Read in the right IR voltage and put into a buffer
  irValue = analogRead(rightIRPin);
  irRight[0] = irRight[1];
  irRight[1] = irRight[2];
  irRight[2] = irRight[3];
  irRight[3] = irValue;

  // Calculate the average input
  irLeftAvg = (float(irLeft[0]) + float(irLeft[1]) + float(irLeft[2]) + float(irLeft[3]))/4;
  irRightAvg = (float(irRight[0]) + float(irRight[1]) + float(irRight[2]) + float(irRight[3]))/4;

  // Convert voltage reading to units of inches.
  irLeftAvg = 573.01 * pow(float(irLeftAvg),-0.909);
  irRightAvg = 1768.2* pow(float(irRightAvg), -1.114);

  // Obstacle Avoidance Logic
  if (irLeftAvg < 3.5 && irRightAvg >= 3.5){
    // reverse and turn right
    bumpRecomnd = 3;
    bumpFlag = true;
  }
  else if (irLeftAvg >= 3.5 && irRightAvg < 3.5){
    // reverse and turn left
```

```
  bumpRecomnd = 4;
  bumpFlag = true;
}
else if (irLeftAvg < 3.5 && irRightAvg < 3.5){
  // reverse and turn left or right
  if(random(0,9)/5 == 1){
    bumpRecomnd = 3;
  }
  else{
    bumpRecomnd = 4;
  }
  bumpFlag = true;
}
else if((irRightAvg-irLeftAvg)< eps && (irLeftAvg >= 3.5 && irLeftAvg < 5.5)\
      && (irRightAvg >= 3.5 && irRightAvg < 5.5)) {
  // Turn left or right
  if(random(0,9)/5 == 1){
    irRecomnd = 1;
  }
  else{
    irRecomnd = 2;
  }
}
else if(irLeftAvg >= 3.5 && irLeftAvg < 5.5 && irRightAvg > 5.5){
  irRecomnd = 1;  // Turn right some random amount
}
else if(irRightAvg >= 3.5 && irRightAvg < 5.5 && irLeftAvg > 5.5){
  irRecomnd = 2;  // Turn left some random amount
}
else{
  irRecomnd = 0;
}

// If the IR recommends something then set the flag to true
if(irRecomnd != 0){
  irFlag = true;
```

```
    }
    else{
      irFlag = false;
    }

    // Debugging outputs
    //  Serial.print(leftString + String(irLeftAvg) + " " );
    //  Serial.println(rightString + String(irRightAvg));

  }
  //-----------------------------------------------------------------------------------------------------------
  //---- Amperage Measurement Smart Sensor ----
  if (currentMillis - previousMillis_Current >= currentPeriod){
    previousMillis_Current = currentMillis;

    currentValue  = driveMotors.getM1CurrentMilliamps();
    currentLeft[0] = currentLeft[1];
    currentLeft[1] = currentLeft[2];
    currentLeft[2] = currentLeft[3];
    currentLeft[3] = currentValue;
    delay(1);

    currentValue = driveMotors.getM2CurrentMilliamps();
    currentRight[0] = currentRight[1];
    currentRight[1] = currentRight[2];
    currentRight[2] = currentRight[3];
    currentRight[3] = currentValue;

    // Calculate average current reading over three samples to try to not in spikes.
    currentLeftAvg = (float(currentLeft[0]) + float(currentLeft[1]) + float(currentLeft[2]) +
float(currentLeft[3]))/4;
    currentRightAvg = (float(currentRight[0]) + float(currentRight[1]) + float(currentRight[2]) +
float(currentRight[3]))/4;

    // Convert to real units (amps)
    currentLeftAvg = 0.034 * currentLeftAvg;
    currentRightAvg = 0.034 * currentRightAvg;
```

```
    // Tune this value
    if(currentLeftAvg >= 4 || currentRightAvg >= 4) {
      currentRecomnd = 1;  // Arbitraty number for now just to trigger the flag.
    }
    else{
      currentRecomnd = 0;
    }

    // If the current sensor recommends something then set the flag to true
    if(currentRecomnd != 0){
      currentFlag = true;
      actionOverride = true;
    }
    else{
      currentFlag = false;
    }

    // Debug variable declaration
    //currentFlag = false;

    /* Debugging Outputs
      Serial.print("Left Current: " + String(currentLeftAvg) + " " );
      Serial.println("Right Current: " + String(currentRightAvg));
    */
  }
//------------------------------------------------------------------------------------------------------------

  if(!OAoff){
    // Any sensor flag will trigger alternative behavior
    if (currentFlag || bumpFlag || irFlag) {
      oaOverride = true;
      if (currentRecomnd == 1) {
      // Stop motion, robot could be stuck.
      Reverse = false;
      Turn = false;
```

```
motionDirection = 0;
currentFlag = false;
}
else if(bumpRecomnd == 3){
  // Reverse and right turn motion
  Reverse = true;
  Turn = true;
  motionDirection = 4;
  bumpFlag = false;
}
else if(bumpRecomnd == 4){
  // Reverse and left turn motion
  Reverse = true;
  Turn = true;
  motionDirection= 3;
  bumpFlag = false;
  /*
  if(!actionLock){
    actionLock = true;
    driveMotors.setM1Speed(-75);
    driveMotors.setM2Speed(75);
    actionTimer =  currentMillis;
    bumpRecomnd = 0;
  }
  */
}
else if(irRecomnd == 1){
  // Turn Right
  Reverse = false;
  Turn = true;
  motionDirection = 4;
}
else if(irRecomnd == 2){
  // Turn left
  Reverse = false;
  Turn = true;
```

```
      motionDirection = 3;
    }
  }
}

//---- Obstacle Avoidance Action ----

if(Reverse && isStopped){
  // perform action
  motionDirection = 2;
}

if(Turn && !Reverse && isStopped){
    // Perform Turn

    // Which way to turn?
    if(motionDirection == 3){
      // Turn Left

        Turn = false;
        motionDirection = 0;
        oaOverride = false;
    }
    else if(motionDirection == 4){
      // Turn Right

      motionDirection = 0;
    }
}

//---- Commanded Robot Move ----
if(newDistance && !oaOverride){
  if( leftDistance < 0 && rightDistance > 0){
    //Forward motion
    motionDirection = 1;
  }
```

```
    else if(leftDistance > 0 && rightDistance < 0){
      // Reverse Motion
      motionDirection = 2;
    }
    else if(leftDistance > 0 && rightDistance > 0){
      // Turning Left
      motionDirection = 3;
    }
    else if(leftDistance < 0 && rightDistance < 0){
      // Turning Right
      motionDirection = 4;
    }
    else{
      //Stop
      motionDirection = 0;
    }
  }

  if((motionDirection != oldMotionDirection || motionDirection == 0) && !isStopped){
      // Now doing a different motion
      oldMotionDirection = motionDirection;
      if(!isStopped){
        leftSetpoint = 0;
        rightSetpoint = 0;

        if(abs(currentMillis - previousMillis_Stopped) > stoppedPeriod){
          if(abs(leftStartPoint - leftEncoder.read()*C) < 0.00001 && abs(rightStartPoint - rightEncoder.read()*C) <
0.00001){
            isStopped = true;
          }
          leftStartPoint = leftInput;
          rightStartPoint = rightInput;
        }
      }
    }

  if(newDistance || oaOverride){
```

```
  if(isStopped && motionDirection == 1){
    // Forward Motion
    leftStartPoint = leftEncoder.read()*C;
    rightStartPoint = rightEncoder.read()*C;
    leftSetpoint = robotSpeed;
    rightSetpoint = robotSpeed;

  }
  else if(isStopped && motionDirection == 2){
    // Reverse Motion
    leftStartPoint = leftEncoder.read()*C;
    rightStartPoint = rightEncoder.read()*C;
    leftSetpoint = -robotSpeed;
    rightSetpoint = -robotSpeed;

  }
  else if(isStopped && motionDirection == 3){
    // Left Turn
    leftStartPoint = leftEncoder.read()*C;
    rightStartPoint = rightEncoder.read()*C;
    leftSetpoint = -robotSpeed;
    rightSetpoint = robotSpeed;

  }
  else if(isStopped && motionDirection == 4){
    // Right Turn
    leftStartPoint = leftEncoder.read()*C;
    rightStartPoint = rightEncoder.read()*C;
    leftSetpoint = robotSpeed;
    rightSetpoint = -robotSpeed;

  }
}
```

```
  /*
  if((leftInput - leftStartPoint)>(leftDistance-leftOffset) && abs(rightInput)>(rightDistance-rightOffset) &&
rightSetpoint != 0){
    leftSetpoint = 0;
    rightSetpoint = 0;
  }
  */

// PD controller
// Need to convert to actual position measurements.
  leftInput = leftEncoder.read()*C;
  leftDone = leftPID.Compute();

  rightInput = rightEncoder.read()*C;
  rightDone = rightPID.Compute();

  if(leftDone && rightDone){
    // Set the speeds together
    driveMotors.setSpeeds(K*leftOutput, K*rightOutput);
    leftDone = false;
    rightDone = false;
  }
}
//----------------------------------------------------------------------------------------------------------------
// ---Extra Error Function---
// This function is used by the Pololu motor drivers to handle errors in operation
void stopIfFault()
{
  if (driveMotors.getM1Fault())
  {
    //Serial.println("M1 fault:");
    while(1);
  }
  if (driveMotors.getM2Fault())
  {
    //Serial.println("M2 fault:");
    while(1);
```

```
  }
}

// ---Left Bump Sensor Interrupt Function---
void bumpLeft()
{
  bumpFlag = true;
  actionOverride = true;
  bumpRecomnd = 3;

}
// ---Right Bump Sensor Interrupt Function---
void bumpRight()
{
  bumpFlag = true;
  actionOverride = true;
  bumpRecomnd = 4;
}
}
```

Top Level Controller:

```
/* Patrick Header Here
 *
 */
// Tilt servo should cyle through 60-140 with 140 facing down and 60 facing up
// Pan servo should be cycled through 0-180 with 0 facing right

#include <Servo.h>
#include <Pixy.h>
#include <SPI.h>
#include <Messenger.h>
//#include <LiquidCrystal.h>

//----- Variable Declarations -----

// Servomotor Pins
```

```
const int panPWM = 6;
const int tiltPWM = 5;

// Button Pin
const int startPin = 7;
int buttonState = HIGH;
int button;
int previousButton = LOW;

//---- Timing Variables ----
unsigned long currentMillis;

unsigned long debouncePeriod = 200;


// Binary true/false array to store if the object has been recovered yet.
int blocksFound[2] = {0, 0}; // Zero is false

// Serial Communications stuff
int inByte;
int activeBehavior = 0;
boolean newBehavior = false;
int panCmd = 90, tiltCmd = 90;
boolean newPanTiltCmd = false;  // Signifies if a new command has been recieved
boolean newRequest = false;
boolean startButton = true;    // Start button needs to be pressed in order for Bob to start moving
boolean readyBypass = false;    //Used to bypass the serial ready check

//----Define Objects----
// Create a message object
Messenger piMessage = Messenger(':');

// Define pan tilt servo objects
Servo Pan;
Servo Tilt;
```

```
// Define pixy object
Pixy ffPixy; // Forward Facing Pixy

//----Serial Communications Parser----
void messageParse(){
  // This will set the variables that need to be changed
  // from the message
  if (piMessage.available()){
    activeBehavior = piMessage.readInt();
    if (activeBehavior != 9){
      newBehavior = true;
    }
    panCmd = piMessage.readInt();
    tiltCmd = piMessage.readInt();
    if (panCmd != 999 || tiltCmd != 999){
      newPanTiltCmd = true;
    }
  }
}

void setup() {

  // Attach servo to specific pins
  Pan.attach(panPWM);
  Tilt.attach(tiltPWM);
  // Align servos to default locatios
  Pan.write(panCmd);
  Tilt.write(tiltCmd);

  // Initialize Pixy object
  ffPixy.init();

  // Start serial and wait for the "Go" command
  Serial.begin(9600);
  piMessage.attach(messageParse);
```

```
  // Stay in a loop until read to move on
  while(1){
    // Set readyBypass to true to skip waiting for Odroid confirmation and button switch confimation
    if (readyBypass){break;}

    // Serial handshake to start the main program.
    if (Serial.available() > 0){inByte = Serial.read();}

    if (inByte == 115 && startButton){
        Serial.println('g');
        break;
    }

    Serial.println('r');

    delay(100);
  }
}

void loop() {
  // put your main code here, to run repeatedly:
  /* This code should be looking for colored blocks that meet certain color codes.
   *  It should be able to store whether or not the colored code was moved already
   *  Once it finds a new color code it should send a command to the Odroid C1 where it will begin the align and pickup behavior.
   *  During this behavior the pan and tilt functions will be disabled. Until the localize behavior is started. This will only take in desired servo angles
   *  and apply them. SHould include some deadband to stop jittering.
   *  This will maybe include the code to display the LCD.
   */

  while( Serial.available() ) piMessage.process(Serial.read());

  if (newPanTiltCmd){
    Serial.println("Repeat Back " + String(activeBehavior)+ " " + String(panCmd) + " " + String(tiltCmd));
    // Make sure to check inout bounds
    if((panCmd >= 5 && panCmd <= 172) && panCmd != 999){
```

```
    Pan.write(panCmd);
    //Serial.println("Repeat Back " + String(panCmd));
  }
  if((tiltCmd >= 80 && tiltCmd <= 130) && tiltCmd != 999){
    Tilt.write(tiltCmd);
    //Serial.println("Repeat Back " + String(State)+ " " + String(panCmd) +" " String(tiltCmd));
  }
  newPanTiltCmd = false;
}

/*
// Pixy Read
static int i = 0;
int j;
uint16_t blocks;
char buf[32];

// grab blocks!
blocks = ffPixy.getBlocks();

// If there are detect blocks, print them!
if (blocks)
{
  i++;

  // do this (print) every 50 frames because printing every
  // frame would bog down the Arduino
  if (i%50==0)
  {
    sprintf(buf, "Detected %d:\n", blocks);
    Serial.print(buf);
    for (j=0; j<blocks; j++)
    {
      sprintf(buf, "  block %d: ", j);
      Serial.print(buf);
      ffPixy.blocks[j].print();
```

```
    }
   }
  }
  */

}
```

Main Program:

```
__Author__ = "Patrick Neal"

"""
 Place intro header here
 Name

 Purpose

 Any inputs or variable descriptions
"""

from Robot.Robot import *

# Second import should be beacon locations.
Bob = Robot("Bob", np.array([0, 0, 0]))

# A bunch of logical variables
foundBlock = False
pickedupBlock = False
localizeDone = False
droppedBlock = False
blocksDone = False
setupComplete = False
continuousRun = False
taskComplete = False
arduinoMegaReady = False
arduinoUnoReady = False

print "Waiting for Arduino Setup"

while True:

    # Wait for the Arduino Mega to respond ready
    if not arduinoMegaReady:
        messageMega = Bob.arduinoMega.readline()
        print messageMega
        if messageMega == 'r\r\n':
            while True:
                Bob.arduinoMega.write('s')
                sleep(0.01)
                if Bob.arduinoMega.readline() == 'g\r\n':
                    break
            arduinoMegaReady = True
            print "Mega Done"

    sleep(0.1)

    # Wait for the arduino Uno to respond ready
    if not arduinoUnoReady:
        messageUno = Bob.arduinoUno.readline()
        print messageUno
        if messageUno == 'r\r\n':
```

```
            while True:
                Bob.arduinoUno.write('s')
                sleep(0.01)
                if Bob.arduinoUno.readline() == 'g\r\n':
                    break
            arduinoUnoReady = True
            print "Uno Done"
        sleep(0.1)

    # Leave setup when both arduinos are ready
    if arduinoMegaReady and arduinoUnoReady:
        break

# Clear the serial buffers before main loop
Bob.arduinoMega.flushInput()
Bob.arduinoMega.flushOutput()

Bob.arduinoUno.flushInput()
Bob.arduinoUno.flushOutput()

# Bob should localize first time through
# Bob.localize()

# Robot will run in this loop.
while continuousRun:

    if not foundBlock:       # Searches for the block
        if Bob.behavior != 1:
            Bob.updateBehavior(1)

    elif foundBlock:         # Picks up the block
        if Bob.behavior != 2:
            Bob.updateBehavior(2)

    elif pickedupBlock:      # Should localize and drop the block
        ## Should cross of that block from search list
        # Bob.localize()

        if Bob.behavior != 3:
            Bob.updateBehavior(3)

    elif droppedBlock:
        # if all blocks are found then task is complete else return to

        if blocksDone:
            taskComplete = True
        else:
            foundBlock = False
            pickedupBock = False
            droppedBlock = False

    if Bob.behavior == 1:
        # Do what is necessary to find a block
    elif Bob.behavior == 2:
        # Do what is necessary to pick up the block
    elif Bob.behavior == 3:

    elif Bob.behavior == 4:


    # Just start moving forward (maybe move randomly to increase chance of seeing block
    Bob.move("F", 25)



    # Poll the Uno and Mega for updates
    Bob.stateUpdate()
```

```
    # Update the state of Bob maybe after some specified time.

    # Behavior changes should go here


    if taskComplete:
        # Stop robot functions and then break loop
        break
```

Robot Class:

```
__author__ = 'Patrick'

from math import *               # Standard math library
#from libpixyusb_swig.pixy import *     # Python wrapper for C++ Pixy CMUcam5 library
#from ctypes import *
import serial                   # Library for communicating over serial
import numpy as np               # Matrix math library
from time import clock, sleep            # Some standard library
import random


class Robot:

    def __init__(self, robotName, beacons):
        self.name = robotName                # Name of your robot
        self.behavior = 0                    # Current behavior the robot is in
        self.state = np.array([0, 0, 0])
        # Serial communications for the Arduino Mega
        self.arduinoMega = serial.Serial(port = '/dev/ttyACM0', baudrate = 9600, timeout = 1,
writeTimeout = 2)

        # Serial communications for the Arduino Uno
        self.arduinoUno = serial.Serial(port = '/dev/ttyACM1', baudrate = 9600, timeout = 1,
writeTimeout = 2)

        self.arduinoMega.flushInput()
        self.arduinoMega.flushOutput()

        self.arduinoUno.flushInput()
        self.arduinoUno.flushOutput()

        self.beaconPos = beacons

    def stateUpdate(self):
        self.arduinoMega.write()
        sleep(0.1)
        message = self.arduinoMega.readline()

        statechange = np.array([float(message[0]), float(message[1]), float(message[2])])

        self.state += statechange

    def updateBehavior(self, behavior):
        self.behavior = behavior
        self.arduinoMega.write(str(self.behavior) + ':' + '99:99:999:999:99:9:9:\r')
        sleep(0.1)
        self.arduinoUno.write(str(self.behavior) + ':' + '99:99:999:999:99:9:9:\r')  # Update
with correct Uno message
        sleep(0.1)

    def move(self, Dir, amount):
```

```
        b = 10.375  # inches, distance between wheels
        if Dir == 'L' or Dir == 'l':
            distance = b*float(amount)/2
            self.arduinoMega.write('9:' + "{:.2f}".format(distance) + ':' +
"{:.2f}".format(distance) + ':' +
                                   '999:999:99:9:9:\r')
        elif Dir == 'R' or Dir == 'r':
            distance = b*float(amount)/2
            self.arduinoMega.write('9:' + "{:.2f}".format(-distance) + ':' +
"{:.2f}".format(-distance) + ':' +
                                   '999:999:99:9:9:\r')
        elif Dir == 'F' or Dir == 'f':
            distance = float(amount)
            self.arduinoMega.write('9:' + "{:.2f}".format(-distance) + ':' +
"{:.2f}".format(distance) + ':' +
                                   '999:999:99:9:9:\r')
        elif Dir == 'B' or Dir == 'b':
            distance = float(raw_input())
            self.arduinoMega.write('9:' + "{:.2f}".format(distance) + ':' + "{:.2f}".format(-
distance) + ':' +
                                   '999:999:99:9:9:\r')

    def readMega(self):
        # This method will read the output of the Arduino Mega and parse the information
        # returns a list of the separate components of the message as strings
        megaMessage = self.arduinoMega.readline()
        messageListMega = megaMessage.split(":")
        return messageListMega

    def readUno(self):
        # This method will read the output of the Arduino Uno and parse the information
        # returns a list of the separate components of the message as strings
        unoMessage = self.arduinoUno.readline()
        messageListUno = unoMessage.split(":")
        return messageListUno
    """
    def collectdata(self):
        # This needs to collect data from three out of four beacons.
        # These beacons should have a identifier so their specific absolute position is know.
        # Needs to be able to communicate with the Arduino Uno.(Global serial variables?)
        # Also needs to communicate with a Pixy camera
        # Pick the first one that is in view.(Object should meet specific geometry
conditions)

        # Tell both the Mega and Uno what behavior is now acting (Localize)

        # Arduino Mega should stop movements and passive while waiting for more commands.
        self.arduinoMega.write()
        self.arduinoUno.write()




        foundBeacons = False

        # Decides where to start looking left or right.
        if random.randint(0, 9)/5 == 1:
            # Start turning left
            direction = 'L'
        else:
            direction = 'R'

        lambdaArray = np.array([0, 0, 0])

        # Read the current state of the
        self.arduinoUno.read()
```

```
        while not foundBeacons:
            # Find first beacon ( start randomly looking left or right first find the
closest.)
            # Add the beacon ID to a list along with the angle *maybe numpy array)
            # continue in the same direction to find the next

            # If at boundary then command the robot to rotate some amount( with move
command?)
            # Also rotate the camera back the same amount.

            # Continue measuring angle between next beacon and place in the list


            if beaconCount == 3:
                foundBeacons = True

            if  and direction == 'L' and not foundBeacons:




         = self.localize(r1, r2, r3, lambdaArray[0], lambdaArray[1], lambdaArray[2])

        # Some error checking maybe

        # Should return Data which stores the measured angles

        return Data



    def localize(self):

        # Update robot current behavior
        self.updateBehavior(4)

        # Collect data for localizng
        Angles = self.collectData()

        # Perform the calculations


        # Convert degrees into radians
        lambda1 = radians(lambda1)
        lambda2 = radians(lambda2)
        lambda3 = radians(lambda3)

        # If less than three beacons return error code
        lambda12 = lambda2 - lambda1
        if lambda1 > lambda2: lambda12 += 2*pi

        lambda31 = lambda1 - lambda3
        if lambda3 > lambda1: lambda31 += 2*pi

        # Apply check for singularity here
        # if lambda12= 0 or pi or lambda31 = 0 or pi


        # Compute L12 from r1, r2
        L12 = sqrt(()^2 + ()^2)

        # Compute L31 from r1, r3
        L31 = sqrt(()^2 + ()^2)

        # Need to calculate phi and sigma
```

```
        # Calculate gamma angle
        gamma = sigma - lambda31

        tau = atan((sin(lambda12)*(L12*sin(lambda31)-
L31*sin(gamma)))/(L31*sin(lambda12)*cos(gamma)-L12*cos(lambda12)*sin(lambda31)))

        if lambda12 < pi and tau < 0: tau += pi
        if lambda12 > pi and tau > 0: tau -= pi

        if abs(sin(lambda12))>abs(sin(lambda31)):
            L1 = (L12*sin(tau+lambda12))/sin(lambda12)
        else
            L1 = (L31*sin(tau+sigma-lambda31))/sin(lambda31)

        xR = r1.x - L1*cos(phi+tau)
        yR = r1.y - L1*sin(phi+tau)

        thetaR = phi +tau -lambda1
        if thetaR <= -pi:
            thetaR += 2*pi
        elif thetaR>pi:
            thetaR -= 2*pi

        stateTriang =  np.array([xR, yR, thetaR])
        self.state = stateTriang
"""
```

Remote Control Code:

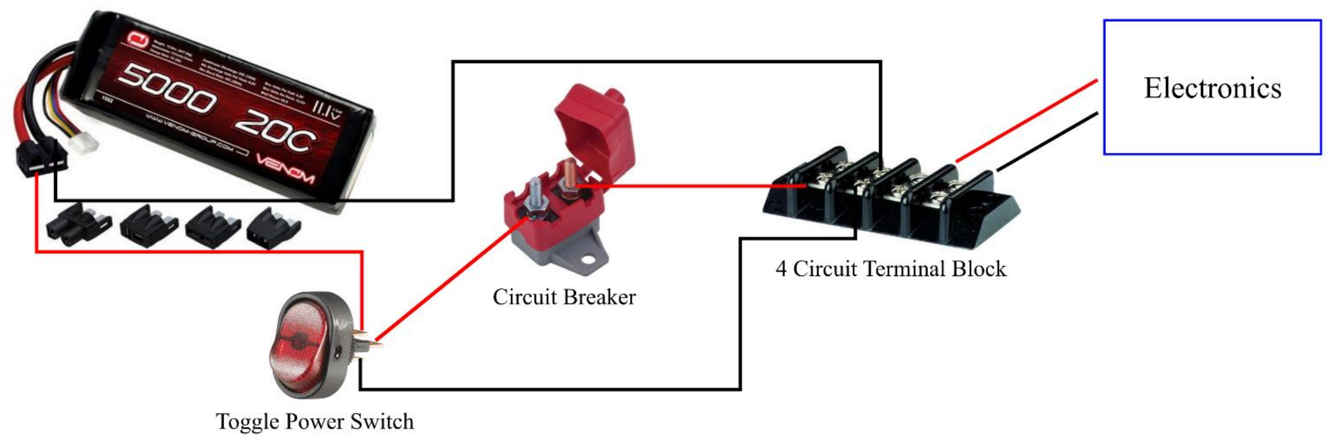
```
# Import libraries to use
from math import *              # Standard math library
# from libpixyusb_swig.pixy import *    # Python wrapper for C++ Pixy CMUcam5 library
# from ctypes import *
import serial                   # Library for communicating over serial
import numpy as np              # Matrix math library
from time import clock, sleep     # Some standard library

# Start serial connection with Arduino (These settings will most likely
# need to be changed).

def convertStr(s):
    """Convert string to either int or float."""
    try:
        ret = int(s)
    except ValueError:
        # Try float.
        ret = float(s)
    return ret


# Serial communcations for the Arduino Mega
arduinoMega = serial.Serial('/dev/ttyACM1', 9600, timeout = 1, writeTimeout = 2)

# Serial communcations for the Arduino Uno
arduinoUno = serial.Serial('/dev/ttyACM0', 9600, timeout = 1, writeTimeout = 2)

arduinoMega.flushInput()
arduinoMega.flushOutput()

arduinoUno.flushInput()
arduinoUno.flushOutput()

arduinoMegaReady = False
```

```
arduinoUnoReady = False

print "Waiting for Arduino Setup"

while True:

    if not arduinoMegaReady:
        messageMega = arduinoMega.readline()
        print messageMega
        if messageMega == 'r\r\n':
            while True:
                arduinoMega.write('s')
                sleep(0.01)
                if arduinoMega.readline() == 'g\r\n':
                    break
            arduinoMegaReady = True
            print "Mega Done"

    sleep(0.1)

    if not arduinoUnoReady:
        messageUno = arduinoUno.readline()
        print messageUno
        if messageUno == 'r\r\n':
            while True:
                arduinoUno.write('s')
                sleep(0.01)
                if arduinoUno.readline() == 'g\r\n':
                    break
            arduinoUnoReady = True
            print "Uno Done"
        sleep(0.1)

    if arduinoMegaReady and arduinoUnoReady:
        break

b = 10.375

Loop = True

while Loop:
    print "This is the Main Menu for the Serial Communications Test"
    print "1. Move a certain amount"
    print "2. Change gripper wrist/claw position"
    print "3. Change pan/tilt angles"
    print "4. Change robot state"
    print "Q. Quit (Stops all motion)"
    choice = raw_input()

    if choice == "1":
        while True:
            print "This is the menu for robot movement"
            print "1. Move Forwards"
            print "2. Move Backwards"
            print "3. Turn Left"
            print "4. Turn Right"
            print "5. Stop"
            print "Q. GO back"
            choice = raw_input()

            if choice == '1':
                print "Input desired distance"
                distance = float(raw_input())
                arduinoMega.write('9:' + "{:.2f}".format(-distance) + ':' +
"{:.2f}".format(distance) + ':' +
                                  '999:999:99:9:9:\r')
                sleep(0.01)
```

```
                print arduinoMega.readline()
            elif choice == '2':
                print "Input desired distance"
                distance = float(raw_input())
                arduinoMega.write('9:' + "{:.2f}".format(distance) + ':' + "{:.2f}".format(-
distance) + ':' +
                                  '999:999:99:9:9:\r')
                sleep(0.01)
                print arduinoMega.readline()
            elif choice == '3':
                print "Input desired angle"
                amount = convertStr(raw_input())
                distance = b*float(amount)/2
                arduinoMega.write('9:' + "{:.2f}".format(distance) + ':' +
"{:.2f}".format(distance) + ':' +
                                  '999:999:99:9:9:\r')
                sleep(0.01)
                print arduinoMega.readline()
            elif choice == '4':
                print "Input desired angle"
                amount = convertStr(raw_input())
                distance = b*float(amount)/2
                arduinoMega.write('9:' + "{:.2f}".format(-distance) + ':' + "{:.2f}".format(-
distance) + ':' +
                                  '999:999:99:9:9:\r')
                sleep(0.01)
                print arduinoMega.readline()
            elif choice == '5':
                print "Stopped"
                distance = 0
                arduinoMega.write('9:' + "{:.2f}".format(distance) + ':' +
"{:.2f}".format(distance) + ':' +
                                  '999:999:99:9:9:\r')
                sleep(0.01)
                print arduinoMega.readline()
            elif choice == 'q' or choice == 'Q':
                break

    elif choice == '2':
        while True:
            print "This is the menu for the gripper"
            print "1. Change wrist angle"
            print "2. Change claw position"
            print "Q. GO back"
            choice = raw_input()

            if choice == '1':
                print "Input wrist angle in degrees (some limits in place"
                wrist = convertStr(raw_input())
                if 30 < wrist < 181:
                    arduinoMega.write('9:99:99:' + str(wrist) + ':999:99:9:9:\r')
                    sleep(0.01)
                    print arduinoMega.readline()
                else:
                    print "Invalid Input"
            elif choice == '2':
                print "Input grasp angle in degrees (some limits in place"
                grasp = convertStr(raw_input())
                if 45 <= grasp < 135:
                    arduinoMega.write('9:99:99:999:' + str(grasp) + ':99:9:9:\r')
                    sleep(0.01)
                    print arduinoMega.readline()
                else:
                    print "Invalid Input"
            elif choice == 'q' or choice == 'Q':
                break
```

```
    elif choice == '3':
        while True:
            print "This is the menu for the pan/tilt"
            print "1. Change pan angle"
            print "2. Change tilt angle"
            print "Q. GO back"
            choice = raw_input()

            if choice == '1':
                print "Input pan angle in degrees (some limits in place)"
                pan = convertStr(raw_input())
                if 0 <=  pan <= 180:
                    arduinoUno.write('9:'+ str(pan) + ':999:\r')
                    sleep(0.1)
                    print arduinoUno.readline()
                else:
                    print "Invalid Input"
            elif choice == '2':
                print "Input tilt angle in degrees (some limits in place)"
                tilt = convertStr(raw_input())
                if 80 < tilt < 130:
                    arduinoUno.write('9:999:' + str(tilt) + ':\r')
                    sleep(0.1)
                    print arduinoUno.readline()
                else:
                    print "Invalid Input"
            elif choice == 'q' or choice == 'Q':
                break

    elif choice == '4':
        while True:
            print "This is the menu for the pan/tilt"
            print "1. Enter Behavior 1-4"
            print "Q. GO back"
            choice = raw_input()

            if choice == '1':
                print "1 = Search,  2 = Pickup,  3 = Drop off,  4 = Localize)"
                behavior = convertStr(raw_input())
                if 1 <= behavior <= 4:
                    arduinoMega.write(str(behavior) + ':99:99:999:999:99:9:9:\r')
                    sleep(0.1)
                    arduinoUno.write(str(behavior) + ':999:999:\r')
                else:
                    print "Invalid Input"
            elif choice == 'q' or choice == 'Q':
                break

    elif choice == 'q' or choice == 'Q':
        # Send messages to turn off actuators
        arduinoMega.write('9:0:0:180:125:99:9:9:\r')
        arduinoMega.close()
        arduinoUno.write('9:90:90:\r')
        arduinoUno.close()
        Loop = False
```

Power Circuitry Diagrams:



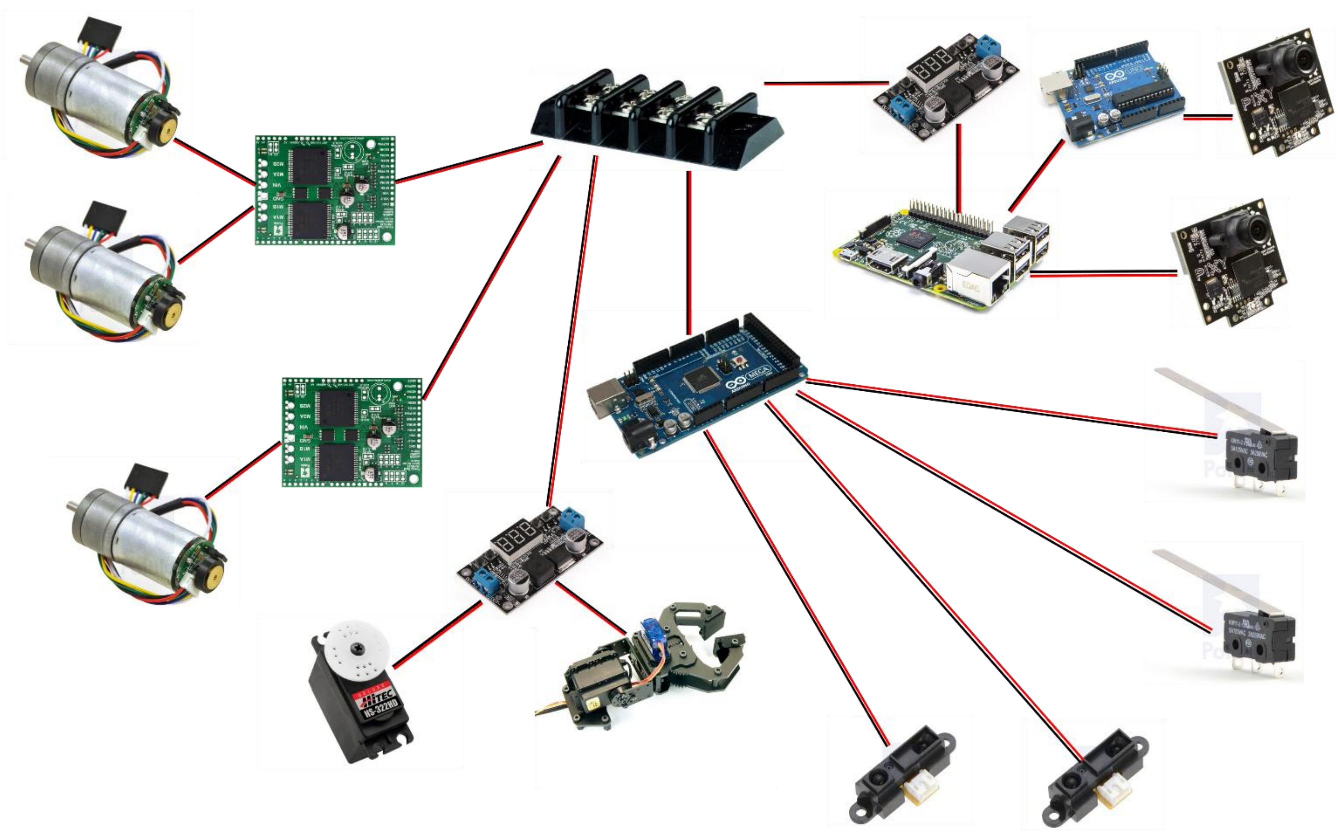


Figure 16 the Odroid C1 2 Model B shown in the image was replaced with a Odroid C1 for the final implementation.

Complete Sensor wiring diagram: