

Robotics Challenge

Final Report

MEEN 408/612

Team 2

CAD and Manufacturing:

Jon Markcity

Connor Barnett

Eric Dehnert

Modeling and Control:

Austin Helmreich

Lingjie Xu

Meng Jin

Programming:

Austin Burch

Tyler Marr

December 15, 2016

Introduction

The goal of this project is construct a two-link system that is able to accurately throw a small table tennis ball into a hoop that is located 4 feet from the fully extended arm. In order to best accomplish this task it was determined that the team of 8 should be divided into 3 sub-task teams consisting of CAD and manufacturing, modeling and controls, and programming and system implementation. Each team's members can be seen in the title page. This report consists of the subteam design approaches, as well as a detailed description of the ROS bbot_pkg implemented in this system.

This robot utilized two DC brushed motors at the revolute joints, and an electromagnet at the end effector. The control system approach was to plan a consistent throwing path, with an optimal release position and end effector velocity as to make the ball through the hoop. The bbot_pkg system operated with a master node running Ubuntu on an laptop, and two Beaglebone Black nodes, one each for reading and driving a their respective joint. The results of this team's demonstration resulted in mechanical failure, and future work and predictive troubleshooting is discussed throughout this report to note the shortcomings and improvements.

Groups

CAD and Manufacturing

A critical aspect for the robot challenge is to physically construct a robot that can reliably launch a ball into a trash can. After analyzing and making adjustments to an initial design, the final system was designed in SolidWorks and 3D printed (see appendix D) This rather *festive* model can be seen in Figure 1. In order to perform the necessary operations, the system will be clamped to the surface of a table by utilizing the wood. The wooden piece will also be used as a workbench for necessary programming instruments (BBB) to be mounted to.

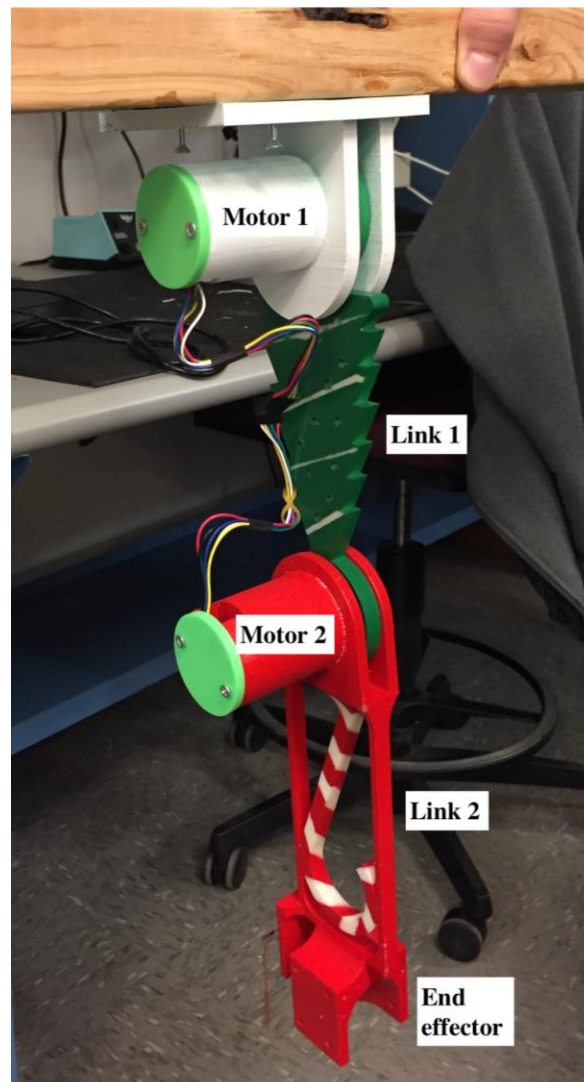





Figure 1. Design of robot system.

One of the most important aspects for the robot arm is how the ball will be contained until the desired launching point. By utilizing electromagnetic technology, the ball would be contained in the end effector until the desired position and velocity is reached. The end effector was designed in a way to hold the ping pong ball in place, but also allow motion in the direction

that the ball is released. By inserting a metal screw into the ball, the ping pong ball becomes ‘magnetic’, and can be held by the electromagnet secured below the end effector.

The next part of research was performed to identify the type of actuator that was best suited for the robot. Table 1 below demonstrates the results from the performed research.

Table 1. Hardware list.

Item	Details	Products
Brushed DC motor with encoder	Weight: 104 g Shaft diameter: 4 mm Size: 25D x 66L mm Gear ratio: 74.83:1 48 CPR encoder	
Motor Driver	.NET Gadgeteer L298 Module Weight: 12 g Dimensions: 47 x 37 x 9.75 mm 5V tolerant	
Electromagnet	Operating supply voltage: 6V	

Initially, servo motors were selected to be used in the design. After more research, it was found that these motors are usually used for low speed, precise applications, which is why the team decided to choose a brushed DC motor with high speed and high torque capabilities. The motor selected includes a shaft with a D-shaped cross section. This shape was mimicked in the design of the links, allowing the motors shaft to fit directly inside the links. This fit allows the motor to manipulate the link in sync with the motor rotation.

The motors were inputted into the two cylinder pieces of the links, and secured to the 3D printed material using screws. End caps were then used to close the cylinder (light green pieces) as shown in Figure 1.

This mechanical system had shortcomings in strength and weight. To combat the weight of the 3D-printed arms, holes were drilled throughout in places that did not compromise the arm’s structural integrity. Furthermore two links sheared at the motor shaft and rendered the arm

incapable of a throwing motion. In future designs, a flanged coupler should be used at both motor joints, to distribute the stresses placed on the weaker 3D printed parts.

Another short coming in the mechanical system was the assembly. Although the pieces of the robot fit together, it was difficult to connect the links together because of the small tolerances from the 3D print. Again, this problem would be fixed by adding a type of coupler to the D shaft and making the tolerances larger, and possibly making the robot a ‘side by side’ joint system, to allow easier access to the working parts. Having the links connected adjacent to one another instead of fitting in line with one another would also allow the same dynamics.

Modeling and Control

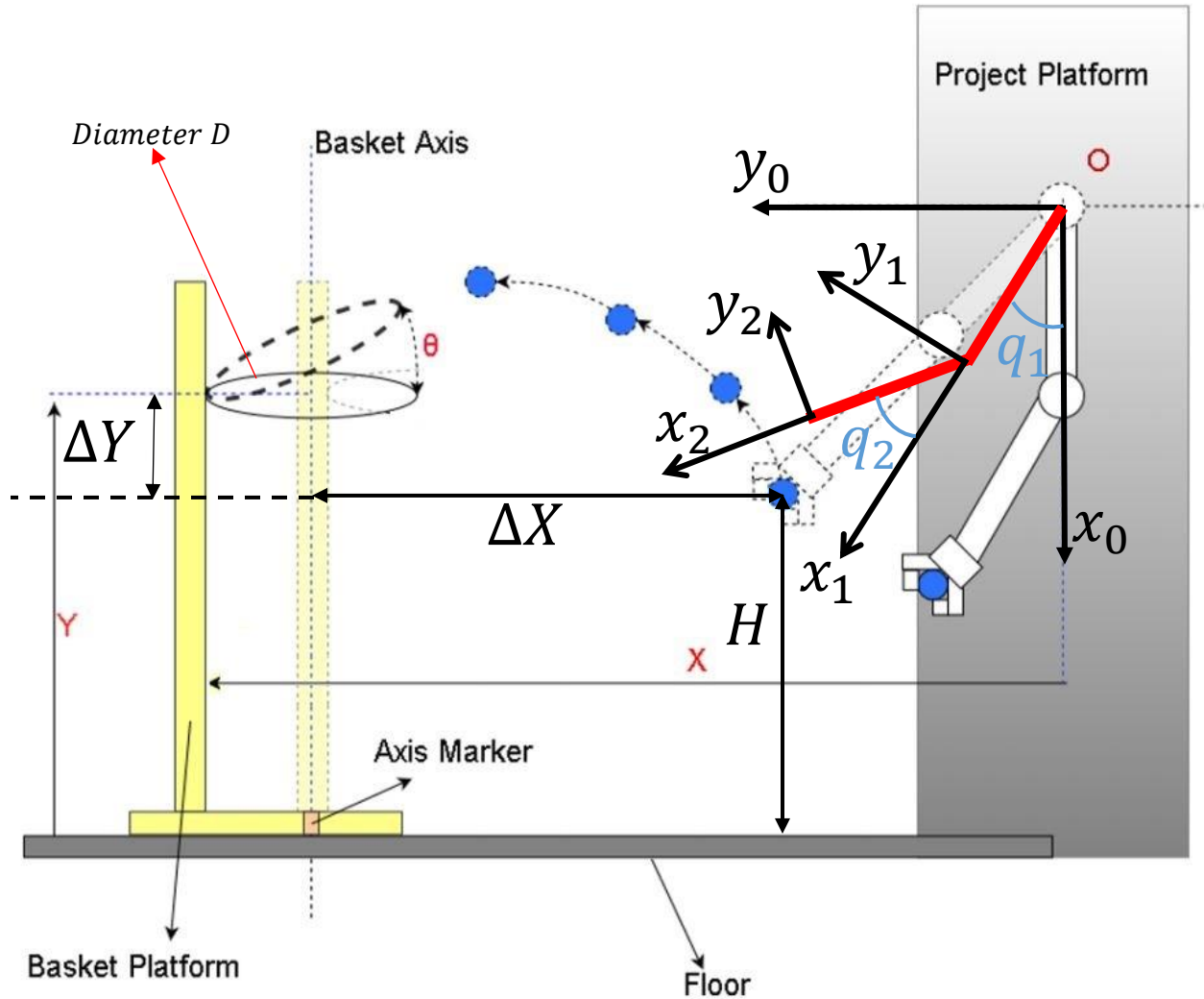


Figure 2. Overview of System and Definitions

The purpose of the model is to allow, when attempting to go through the basket, the ball to hit the middle of the distance between the front end of the hoop and the back stand. In order to achieve the goal, the releasing velocity can be calculated using the end condition and the predefined release angles of each link.

The angle of release is set to be 45 degrees with respect to the ground. And the initial position for the arm is vertically The releasing location in the global frame is (X_0, Y_0) , and the

final location where the ball goes through the basket is (X_t, Y_t) . The time travel after the ball is released is t . And the releasing velocity is v_0 .

1. *The conditions that we can get from the subject:*

First, we choose to set (X_t, Y_t) at the center of the basket. So in the global environment

$$\begin{aligned}(X_t, Y_t) &= [X - \frac{1}{2}D\cos\theta, H - Y] \\ (\Delta X, \Delta Y) &= (X_0, Y_0) - (X_t, Y_t)\end{aligned}$$

According to the laws of physics, there is a kinematic motion equation between the (X_t, Y_t) and (X_0, Y_0) :

$$\begin{aligned}v_x t &= \Delta X \\ v_y t - \frac{1}{2}gt^2 &= \Delta Y\end{aligned}$$

Moreover, since the ball is released by an arm, the velocity has to satisfy the equations derived from the inverse kinematics and the Jacobian:

$$v = \begin{bmatrix} -l_1 \sin q_1 - l_2 \sin(q_1 + q_2) & -l_2 \sin(q_1 + q_2) \\ l_1 \cos q_1 + l_2 \cos(q_1 + q_2) & l_2 \cos(q_1 + q_2) \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}$$

Also, (X_0, Y_0) has to satisfy the relationship:

$$\begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} = \begin{bmatrix} l_1 \cos q_1 + l_2 \cos(q_1 + q_2) \\ l_1 \sin q_1 + l_2 \sin(q_1 + q_2) \end{bmatrix}$$

In addition, there is the equation for the angular position of the link arms:

$$q = \int \dot{q} dt$$

2. *The conditions that are set by us for simplifying the question:*

The desired release trajectory is a 45 degree angle to the horizontal which is the best way to get into the basket. And according to the attributes of vectors, the angle of v will be between q_1 and $q_1 + q_2$. So we set the angle of $q_1 + q_2$ to 45 degrees. We set q_1 is equal to 15 degrees and q_2 is equal to 30 degrees. Next, we integrate the parameters of the links from the modeling CAD files (appendix C).

Then, we can get the certain value of q_1 , q_2 , \dot{q}_1 and \dot{q}_2 . Assuming \ddot{q} is invariable to time:

$$\begin{aligned}q &= \iint \ddot{q} dt^2 = \frac{1}{2}\ddot{q}t^2 \\ \dot{q} &= \int \ddot{q} dt = \ddot{q}t\end{aligned}$$

We can get \ddot{q} from:

$$\ddot{q} = \frac{(\dot{q})^2}{2q}$$

Then

$$t = \frac{\dot{q}}{\ddot{q}}$$

3. The path of the links

A MATLAB code was implemented (shown in Appendix B) which allows us to solve for \ddot{q} . Then the path of the links is:

$$\begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \ddot{q}_1 t^2 \\ \frac{1}{2} \ddot{q}_2 t^2 \end{bmatrix}$$

Based on the solutions for $q_1, q_2, \dot{q}_1, \dot{q}_2, \ddot{q}_1$ and \ddot{q}_2 solved for above and the initial conditions of our links, it is possible to derive a fifth order polynomial that describes the path of the angular position, velocity, and acceleration with time. These equations take the form of a fifth order polynomial. In general, we require a characteristic equation such as:

$$q_0 = a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^4 + a_5 t_0^5$$

This equation describes the path of the angular position with respect to time. In order to describe the path of the angular velocity and acceleration, we take the first and second derivatives of the above:

$$\begin{aligned} q_0 &= a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^4 + a_5 t_0^5 \\ v_0 &= a_1 + 2a_2 t_0 + 3a_3 t_0^2 + 4a_4 t_0^3 + 5a_5 t_0^4 \\ \alpha_0 &= 2a_2 + 6a_3 t_0 + 12a_4 t_0^2 + 20a_5 t_0^5 \end{aligned}$$

Now in order to solve for the constant in these formulas, we must also describe the final position, velocity, and acceleration of the links:

$$\begin{aligned} q_f &= a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^4 + a_5 t_f^5 \\ v_f &= a_1 + 2a_2 t_f + 3a_3 t_f^2 + 4a_4 t_f^3 + 5a_5 t_f^4 \\ \alpha_f &= 2a_2 + 6a_3 t_f + 12a_4 t_f^2 + 20a_5 t_f^5 \end{aligned}$$

Since we know our initial and final desired position, velocity, and acceleration, we can solve for the necessary constant to derive the above fifth order polynomials.

Re-ordering these equations in matrix form yields the following:

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ \alpha_0 \\ q_f \\ v_f \\ \alpha_f \end{bmatrix}$$

Utilizing MATLAB code that appears in *Robot Modeling and Control* by Spong, it is possible to solve for the constants in these polynomials and derive the fifth order polynomial that describe the desired path of our system with respect to time. With that information, it is now possible to come up with a desired trajectory upon which the system can be controlled. The control of this system is mentioned and outlined below.

Based on the coordinate system defined in the schematic. The Euler-Lagrange method can be used to derive the EOM (equation of motion). The general form of the equation is

$$M(q)\ddot{q} + C(q, \dot{q}) + G(q) = \tau$$

where M, C, and G are the matrices derived by running the predefined model configuration through Mathematica. The resultant moments of inertia for both links and motors combined come from the CAD model.

Different combinations of q_1 and q_2 will be tested to secure the feasibility and rationality of the angular velocity of each joint and the corresponding angular accelerations. And the time needed to travel before reaching the release point is based on the selected q_1 and q_2 pair. Once the time needed before the release is calculated, specific locations along the path can be picked to help with the symbolic derivation for the desired joint angle, angular velocity and acceleration. After the derivation, the three kinematic properties will be able to plotted against time when there is a set time vector available based on the pre-release time.

For tracking purpose, apply the equation below:

$$\begin{aligned} \tau &= -K_0\tilde{q} - K_1\dot{\tilde{q}} \\ \tilde{q} &= q - q_d \\ \dot{\tilde{q}} &= \dot{q} - \dot{q}_d \end{aligned}$$

where q is the actual joint angles at each point of time, q_d is the desired joint angles at the same point. \dot{q} is the actual angular velocity, while \dot{q}_d is the desired one. This helps finding out whether K_0 and K_1 on trial generate minimal error. If the errors are too big, K_0 and K_1 need to be readjusted for multiple time. Such iteration goes on until the errors between the actual and desired value come down to an acceptable range.

Due to the fact that the tracking method doesn't take gravity into account, it is necessary to apply inverse dynamics to find more precise values to control the input torque. Define input:

$$u = Da_q + C\dot{q} + G$$

$$a_q = -K_0\tilde{q} - K_1\dot{\tilde{q}} + r$$

$$\text{Let } a_q = \ddot{q}$$

$$\text{Define } q - q_d = e$$

Then the error dynamic can be shown as following:

$$e(\ddot{t}) + K_1 e(\dot{t}) + K_0 e(t) = 0$$

$$K_1 = 2\xi\omega, K_0 = \omega^2$$

Pick the critically damped system, $\xi = 1$:

$$K_0 = \begin{bmatrix} \omega_1^2 & 0 \\ 0 & \omega_2^2 \end{bmatrix}$$

$$K_1 = \begin{bmatrix} 2\omega_1 & 0 \\ 0 & 2\omega_2 \end{bmatrix}$$

In order to find the optimal values for the coefficients, iteration is a necessary step to go through by substituting different pairs of values and demonstrate the response to see what further modifications are needed to shorten the time needed to reach steady state. General rules of tuning the controller is to adjust K_1 only 1st by setting another one to 0 until the system starts to demonstrate self-oscillation. Then adjust K_0 until the curve becomes smooth.

Programming and System Implementation (All codes for bbot_pkg/src are listed in Appendix E)

In order to accomplish the desired goal of launching a ping pong ball into a hoop, the first task would be for the desired path to be given to the BBB's, this was to be accomplished on the command center through the use of the MATLAB program, as previously mentioned in the "Modeling and Control" section, converted into a C++ file that could be used by ROS. The desired path would then be given to BBB 2 that controlled the motors. When each test began, first BBB 2 would turn the electromagnet on by using a GPIO input, and then wait 5 seconds to allow time for the ping pong ball to be placed into position. After this five second period BBB 1 would began collecting data from the encoders on the motors by using the available eQEP capabilities and performed simple calculations based on the conversion from the encoder value to angle in degrees in order to have a real time angle and angular velocity. This data was then sent to BBB 2 that used PWM inputs of varying duty cycles to control the motors. Using both the desired path and the real values of position and velocity the BBB could implement a simple PD controller in order to output the required torque. The system would continue to follow the desired path until the correct final values were reached and then the ball would be released by turning off the electromagnet. Throughout this process, ROSqt would be used to produce a graph of the real time measurements versus the desired values. Throughout the course of attempting to implement this system, many aspects did not go entirely as planned.

First, a library was developed so that all of the functions required of the system could be kept together and referenced in one common place. However, upon completing and compiling this library, the library was not able to be used due difficulties initialized the library and dependencies in the Cmakelists.txt file.. After several attempts at different methods to implement this library, it was decided that all future C++ files would simply include all of their required

classes at the bottom of the file. These classes could simply be copy and pasted as needed from the created library, and would only add a little bit of time to the code writing process and lower efficiency of the code. Next, the basic framework for the sensing and driving nodes were written in C++ on a computer and then transferred onto a BBB. The team was able to achieve communication between these two nodes and send the, at the time nonexistent, angle measurements from the sensing node to the driving node that were on the same BBB. This code could be further tested once all of the equipment was obtained.

Once the electromagnet and motors were obtained, tests of the PWM, eQEP, and GPIO functions could be performed. Through the use of a breadboard and a simple circuit, the GPIO function was tested on the electromagnet. After minor adjustments, this previously written code was found to be successful in turning the electromagnet on and off. Next, the eQEP function was tested by using the encoder on one of the motors. This code was found to be fully functional, and the BBB was able to send this information from the sensing node to the driving node. Finally, the PWM code was also found to work, and driving of the motors was successful. Shortly after this point, the rest of the purchased equipment came in. This testing setup can be seen in Figure 5.

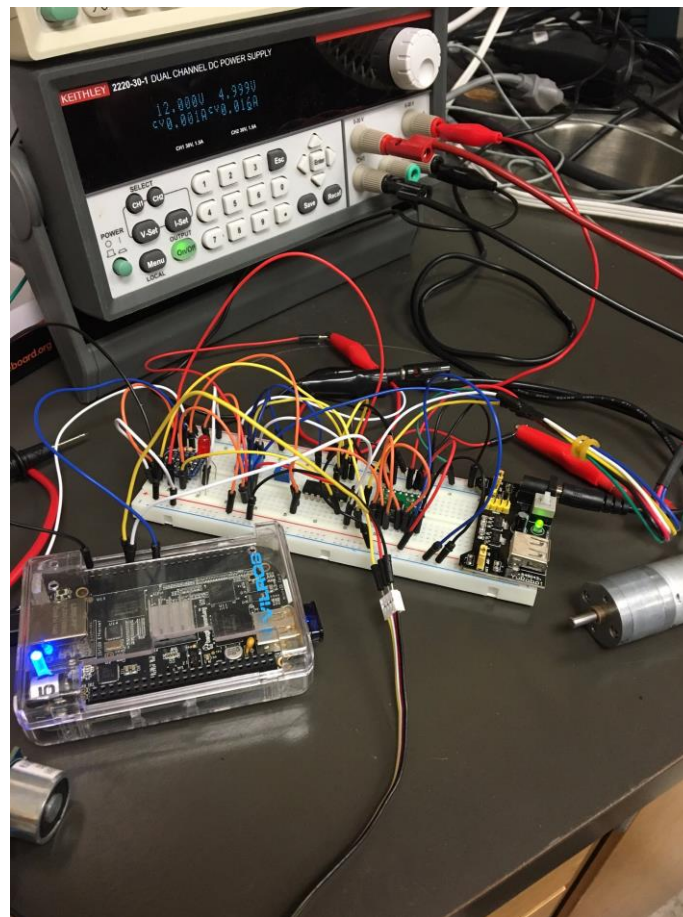


Figure 3: Breadboard Testing setup

After all of the equipment came in, a circuit was designed to allow the system to work as desired. A prototyping board was used to accept the battery and 5V regulator inputs. From here, power distribution to all devices and encoders were mapped. Finally signal lines were routed,

and male headers such that each device could be simply plugged into the board. A second revision of this board would be much more organized. This board can be seen below in Figure 6.

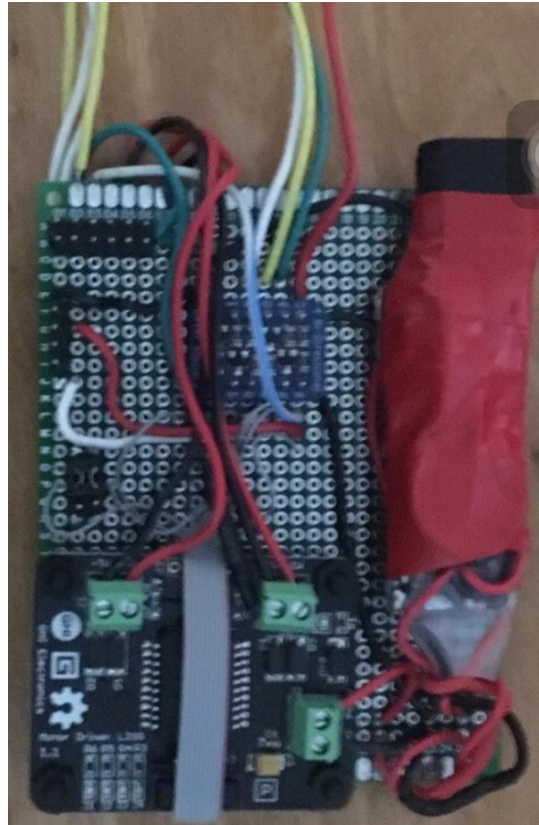


Figure 4: PDB and signal routing on protoboard.

During the time while the protoboard was being completed, the system could not be tested. At this point, the network connecting the two BBBs and the CPU was attempted to be set up. While attempting to set up the network, several problems were encountered. There were several issues with connecting to the different BBBs, when changing the internal settings of the systems. At one point ability to SSH into one of the BBBs was completely lost, and moving forward a replacement BBB had to be used. Unknown at the time, this BBB was not completely set up to perform all of the PWM and eQEP functions. Once the network was completely set up and everything connected the the router, communication between the two BBBs and the CPU was accomplished. All systems were able to successfully send and receive information to each other.

At this point, the MATLAB files were attempted to be transcribed into C++. These programs involved many matrix multiplications and manipulations and C++ is not natively equipped to handle these kind of calculations. It was planned to install Eigen, a C++ template library for linear algebra, however complications were encountered similar to those in the previous attempt to create a library. Due to time constraints, this was seen as a low priority issue, and other tasks were worked on first. Other solutions to this would have been setting up a node in matlab so that the matrix calculations could be performed, or using matlab to generate a text file of desired trajectory values that could be accessed by the ROS command center node. The second option would be much less efficient.

After the network was completely set up and all of the power distribution and signal lines were routed as desired, the system was assembled with the motors and electromagnet in place. The wires were placed into the proper pins of the BBB and initial testing began. The first results of the PWM were very promising. The arm was responsive to the motor input and should signs of very high speeds. After a few trials, as previously mentioned in the “CAD and Manufacturing” section, the arm sheared at where the motor shaft was connected to it, and it became incapable of a throwing motion. It was also at this time that it was discovered that each BBB, as set up, was only capable of one PWM and one eQEP function. Significant effort was put into allowing the BBBs to perform two of either of these functions so that the original sensing and driving nodes could still be used. Upon difficulties to deploy two eQEP channels and 2 PWM channels simultaneously on the BBBs, the setup was adjusted to have a single eQEP and PWM channel per BBB. The setup then became one node reading and driving each joint, while receiving driving commands from the command center node. It was then discovered that one of the BBBs replaced earlier that night when debugging network issues did not have PWM or eQEP functionality set up. eQEP functionality was completed, but due to no internet access, the device tree could not be pulled from online, and therefore was unable to access the PWM chip. At this point, while communication was still accomplished between all devices, there was only one BBB that could be used to drive the motors and it could only perform one PWM, one eQEP, and GPIO functions. It was decided that this would be used most effectively by only driving the motor at joint 1 while simultaneously reading the encoder values using eQEP and converting those values to degrees. The system was programmed so that the electromagnet would be turned on, pause for five seconds, rotate 30°, and release the ball from the electromagnet. While disassembled, the program could be seen to be working correctly at driving the motor and turning the electromagnet on and off. At this point the system was required to demonstrate its functionality, so it was fully assembled with the new link 1.

At demonstration all of the programming aspects behaved as exactly as anticipated. Full connectivity was achieved over all of the devices, and the BBB that was used to control the motor and electromagnet was able to perform all of its tasks. However this did not transfer to the success of the robot due to the same problem with the new link 1 as the previous link 1.

Project Timeline

Since this project was so extensive, it was necessary to make a timeline to ensure that we stayed on track. The following table demonstrates the projected timeline used for the last month of the project. Since the past month was the most critical aspect of the semester, we decided for simplicity to include those pieces of information. As can be seen in table 2, all of the parts were ordered, huge advancements in programming and modeling were made, and the robot was first tested.

Table 2. Project timeline.

Programming		Nov 8 - Nov 13	Nov 13 - Nov 20	Nov 20 - Nov 27	Nov 27 - Dec 4	Dec 4-15
	Finish lib					
	Add msg for second BBB					
	Incorporate main routine into ROS system					
	Implement control system					
	Master computer interface					
	Realtime interface controls					
	Simulation and real time visualization					
	Package and cleanup					
Modeling						
	Physics Theory, Motion, & Dynamics					
	Robotica Modeling					
	Arduino Testing					
	Apparatus tuning					
Manufacturing						
	Finalize and order parts					
	Finalize CAD					
	Print robot arm					
	Assemble arm with motors					
	Test movement					
Testing						
	Control algorithm with microcontroller					
	Simulation					
	Single node testing					
	Full node and master testing					
	Report					

Conclusion

The goal of this project was to create a 2 link robotic system that would throw a ping pong ball into a specified hoop location. This would be done by inputting the given parameters

of the hoop, and letting the system then automatically shoot the ball. The project task was split up into three major groups: modeling, manufacturing, and system control. The system was manufactured using Solidworks design, and 3D printed using the EIC's high fidelity printers. Brushed DC motors were used to manipulate the system, and were assembled into the design along with an electromagnet used to hold the ball. During the presentation of the robot, the motors selected stripped the D-shaped holes in the first link. This failure could next time be prevented by adding a coupler to the motor input, allowing torque to be taken by a metal piece instead of the soft 3D printing material. The team feels confident that the robot would sufficiently work with this manufacturing fix. The network was operational and fully implemented into ROS. Various classes were used throughout the `bbot_pkg/src` files for fast implementation and efficient programming. The use of `eigen` into `ros` to perform all necessary matrix operations was a project oversight, and could have greatly helped with optimizing the control algorithm. Utilizing MATLAB, quintic polynomial that describe the angular position, velocities, and accelerations of both links were created based on the dynamic requirements that were solved for in an additional MATLAB file. While this control system was not fully tested due to the failure of the D-shaped hole in link 1, there are still some critical conclusions that can be drawn regarding the modeling and control of the system. The control algorithm did not take uncertainty term into consideration, which leads to the significant dependency on the power of motor compensating the potential disturbance and noise that in reality could impact the overall system response and the precision of the trajectory. For the optimization purpose, it would be better to consider going with a robust and adaptive control method to help make the system more invariant to different testing environment.

Acknowledgments

The team would like to make a special thank you to Dr. Hur for his effort on our behalf to make this project a successful learning experience. In addition, we would also like to thank Kenny Chour for providing key guidance and helpful feedback during the initial and final stages of our project.

Appendix A: Network System Setup

Figure 6 below, shows the system hardware and communication selection. This figure is described in detail in the Programming and System Implementation section above.

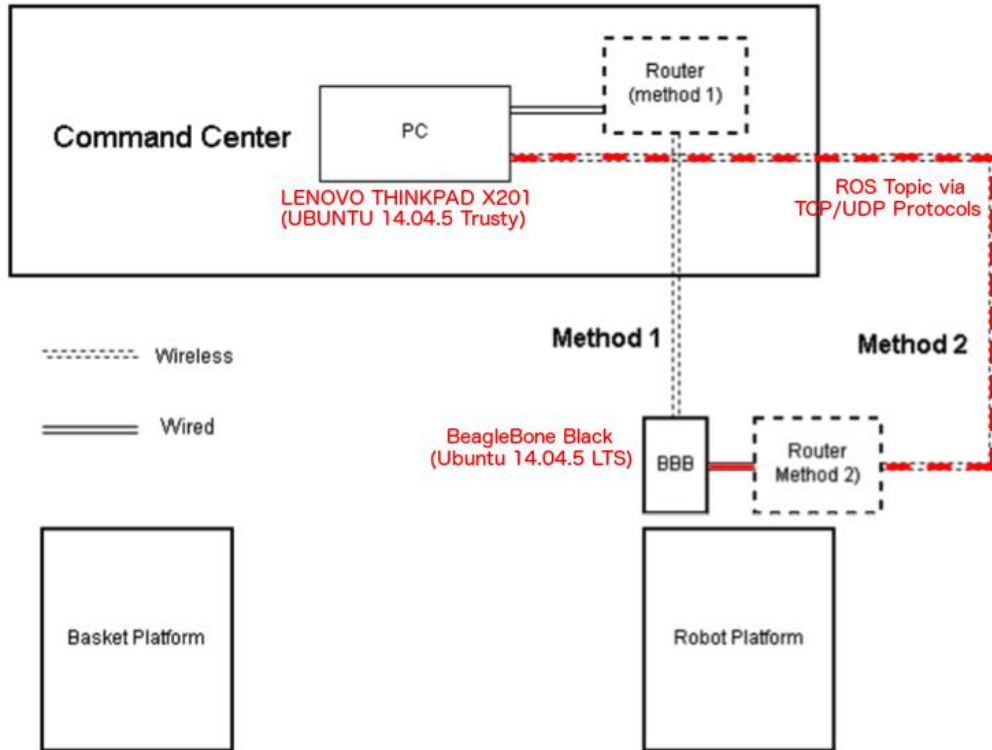


Figure 5. System overview of CPU, OS, and COMM selections.

Appendix B: Matlab Desired Trajectory Scripts

```
%% quintic.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% M-file to compute a quintic polynomial reference trajectory
%%
%% q0 = initial position
%% v0 = initial velocity
%% ac0 = initial acceleration
%% q1 = final position
%% v1 = final velocity
%% ac1 = final acceleration
%% t0 = initial time
%% tf = final time
%%
%clear;clc
close all
t0 = 0;
tf = 2;
SPS = 100;
d =[0,0,0,q1,q_dot(1),q_ddot(1),t0,tf]; %d=initial data = [q0,v0,ac0,q1,v1,ac1,t0,tf]
q0 = d(1); v0 = d(2); ac0 = d(3);
q1 = d(4); v1 = d(5); ac1 = d(6);
t0 = d(7); tf = d(8);
t = [t0:.01:tf]
c = ones(size(t));
M = [ 1 t0 t0^2 t0^3 t0^4 t0^5;
0 1 2*t0 3*t0^2 4*t0^3 5*t0^4;
0 0 2 6*t0 12*t0^2 20*t0^3;
1 tf tf^2 tf^3 tf^4 tf^5;
0 1 2*tf 3*tf^2 4*tf^3 5*tf^4;
0 0 2 6*tf 12*tf^2 20*tf^3];
%%
b=[q0; v0; ac0; q1; v1; ac1];
a = inv(M)*b;
%%
%% qd = position trajectory
%% vd = velocity trajectory
%% ad = acceleration trajectory
%%

qd = a(1).*c + a(2).*t + a(3).*t.^2 + a(4).*t.^3 + a(5).*t.^4 + a(6).*t.^5;
vd = a(2).*c + 2*a(3).*t + 3*a(4).*t.^2 + 4*a(5).*t.^3 + 5*a(6).*t.^4;
ad = 2*a(3).*c + 6*a(4).*t + 12*a(5).*t.^2 + 20*a(6).*t.^3;
q2=40*pi/180; %final position of link 2 rleative to 1 > q1+q2=45deg
g=9.81;

%%End Effector Position(global)
x2=l1*sin(q1)+l2*sin(q1+q2);
y2=l1*cos(q1)+l2*cos(q1+q2);

%%Distance from End Effector Final Position to hoop
x_2h=x_oh-x2;
y_2h=y_h-(Hm-y2);

%%Calculation of Required Initial Velocity
```



```

%vi=((.5*g*x_2h^2)/((cos(q1+q2)^2)*(tan(q1+q2)-y_2h)))^0.5 % (m/s)
%vx=vi*cos(q1+q2)
%vy=vi*sin(q1+q2)
vy = ((g/((x_2h*tan(50*pi/180)-y_2h)^2))^0.5)*(tan(50*pi/180)*x_2h);
vx = vy/tan(50*pi/180);

%%Calculation of q1_dot & q2_dot
v=[-vy;vx;0;0;0;1]; % v matrix
J=[-l1*sin(q1)-l2*sin(q1+q2),-l2*sin(q1+q2);l1*cos(q1)+l2*cos(q1+q2),l2*cos(q1+q2);... % jacobian
figure
qd = qd * 180/pi;
plot(t,qd)
xlabel('time(s)')
ylabel('deg')

figure
vd = vd * 180 / pi;
plot(t,vd)
xlabel('time(s)')
ylabel('deg/s')

figure
plot(t,ad)
xlabel('time(s)')
ylabel('deg/s^2')

%syms t
%qd=vpa(a(1))+vpa(a(2))*T+vpa(a(3))*t^2+vpa(a(4))*t^3+vpa(a(5))*t^4+vpa(a(6))*t^5
%vd=vpa(a(2))+2*vpa(a(3))*t+3*vpa(a(4))*t^2+4*vpa(a(5))*t^3+5*vpa(a(6))*t^4
%ad=2*vpa(a(3))*t+6*vpa(a(4))*t^2+12*vpa(a(5))*t^3+20*vpa(a(6))*t^4

```

```

%% q.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% M-file to compute a q_desired trajectory calculations
%% Initial Conditions of Hoop Position

theta=0;

x_oh=2; %distance from robot origin to center of hoop (m)
x_oh=x_oh-(((3*2.54)/100)*cos(theta)); % Adjust distance for hoop (m)

y_h=1; %height of center of hoop relative to ground (m)
y_h=y_h+(3*2.54)/100*sin(theta); %adjust distance for hoop (m)

%% Constant of Robot Arm
Hm=1; %height of robot arm origin relative to ground
l1=.2; %(m)
l2=.26; %(m)
q1=20*pi/180; %final position of link 1 > q1+q2=45deg

0,0;0,0;0,0;1,1];

%q=v\J %Calculation of Angular Velocity Matrix
q_dot = inv(J([1 2], [1 2])) * v([1 2], [1])

q_ddot1 = (q_dot(1)^2)/(2*q1);
q_ddot2 = (q_dot(2)^2)/(2*q2);
q_ddotn = [q_ddot1;q_ddot2]

```

Appendix C: MATLAB Calculations of Moment of Inertia and COM

clc; close all; clear all;

%% English Unit from SolidWorks

```
m1 = 0.29;    % mass of link 1
mm1 = 0.22;   % mass of motor 1
m2 = 0.6;     % mass of link 2
mm2 = 0;      % ignore the mass of motor 2
rm1 = 3.65;   % distance between link 1 center to the joint
rmm1 = 7.84;  % distance between motor 1 to the joint
rm2 = 3.65;   % distance between link 2 center to the joint
rmm2 = 0;     % ignore distance between motor 2 to the joint
I1en = 0.14;  % moment of inertia of link 1 in english unit w/o motor  alternate value 78.28
I2en = 1.59;  % moment of inertia of link 2 in english unit w/o motor
```

%% English to Metric Conversion

```
lb2kg = 0.4536;
in2m = 0.0254;
Iconversion = 0.00029264; % lb*square in to kg*square meter
```

```
M1 = m1 * lb2kg;
Mm1 = mm1 * lb2kg;
M2 = m2 * lb2kg;
Mm2 = mm2 * lb2kg;
Rm1 = rm1 * in2m;
Rmm1 = rmm1 * in2m;
Rm2 = rm2 * in2m;
Rmm2 = rmm2 * in2m;
I1 = I1en * Iconversion;
I2 = I2en * Iconversion;
```

%% Calculate the Actual Center of Mass

```
lc1 = (M1*Rm1 + Mm1*Rmm1) / (M1 + Mm1); % actual center of mass of member 1
lc2 = (M2*Rm2 + Mm2*Rmm2) / (M2 + Mm2); % actual center of mass of member 2
lc = [lc1 lc2]; % center of mass matrix
```

```
g=9.81;
```

```
syms q1 q2 q1d q2d
```

```
D=vpa([I1+I2+lc1^2*(M1+Mm1)+(Rmm1^2+lc2^2)*M2+2*Rmm1*lc2*M2*cos(q2),
I2+lc2^2*M2+Rmm1*lc2*M2*cos(q2);...
```

```
I2+lc2^2*M2+Rmm1*lc2*M2*cos(q2), I2+lc2^2*M2],4)
```

```
C=vpa([-Rmm1*lc2*M2*sin(q2)*q2d, -Rmm1*lc2*M2*sin(q2)*(q1d+q2d);...
Rmm1*lc2*M2*sin(q2)*q1d, 0],4)
```

```
G=vpa([g*(lc1*M1+Rmm1*M2)*cos(q1)+lc2*M2*cos(q1+q2);g*lc2*M2*cos(q1+q2)],4)
```

Appendix D: SolidWorks Models for 3D Printed Parts

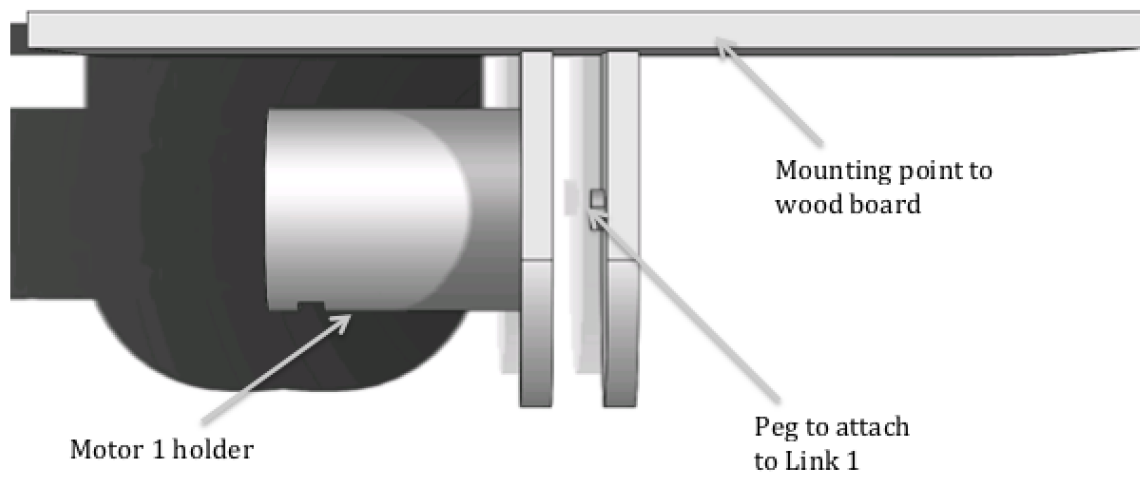


Figure 7 (D.1): Robot Base



Figure 6. (D2). Link 1

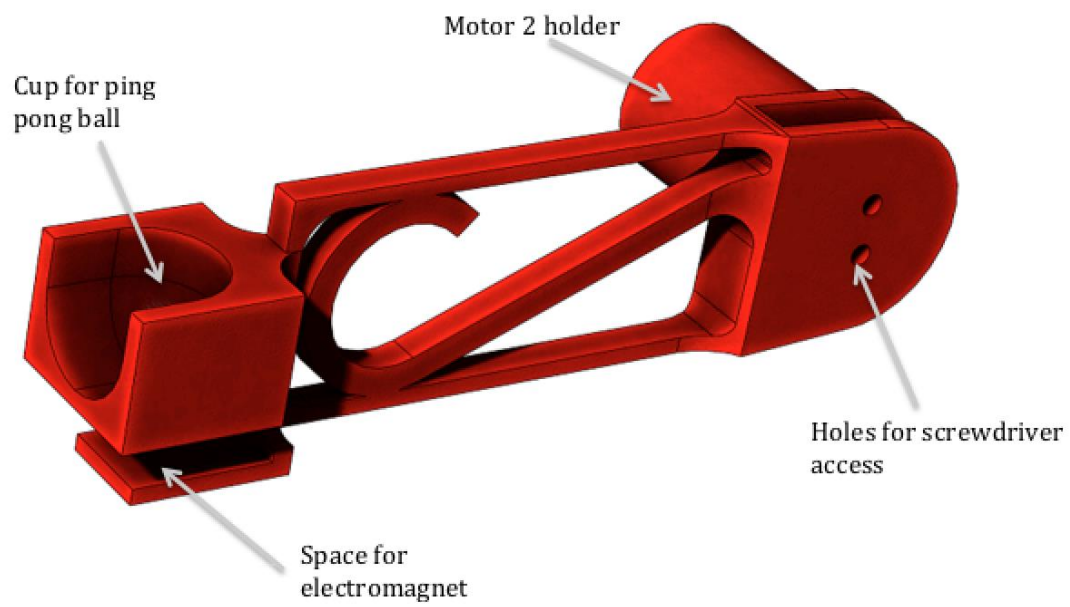


Figure 7. (D3). Link 2

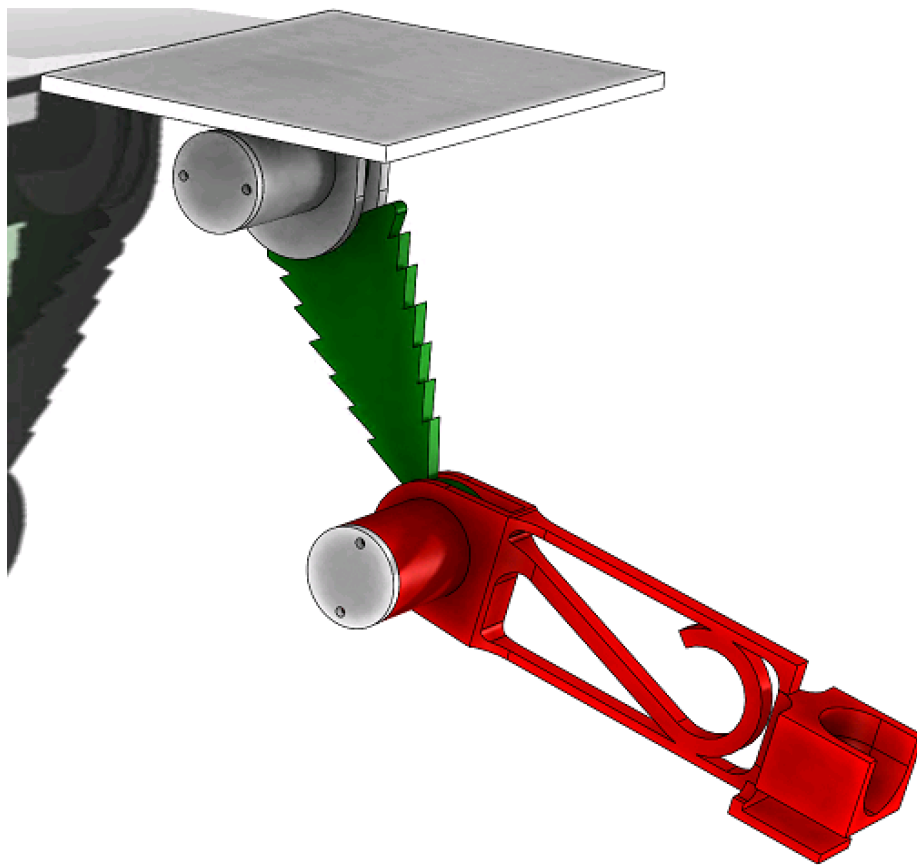


Figure 8. (D.4): SolidWorks model of the robot assembly

Appendix E: ROS bbot_pkg/src files

```
//
// link1_node.cpp
//
// Created by Austin Burch on 11/1/16.
//
//

#include "link1_node.h"
// #include "BBB_lib.h"

using namespace std;

eQEP eqep;
PWM pwm;

double command_center_time = 0;
int link1_count = 0;
// joint vars structure
struct Joint {
    float position = 0, speed = 0, accel = 0;
    float K_p=0, K_i=0, K_d=0;
    float e=0, P_val=0, I_val=0, D_val=0, PID_val=0;
    float prev_e=0, prev_I=0;
    int period=PERIOD_INIT,dutycycle=0,direction;
};
Joint joint1;

struct EndEff {
    float xPos;
    float yPos;
    float zPos;
    float vel;
    float vel_e, prev_vel_e;
};

EndEff gripper;

void command_center_msgCallback(const bbot_pkg::command_center_msg::ConstPtr& command_center_msg)
{
    command_center_time = command_center_msg->command_center_time;
    joint1.dutycycle = command_center_msg->command_center_j1_dutycycle;
    joint1.direction = command_center_msg->command_center_j1_direction;
    release = command_center_msg->command_center_release
}
```

```

int main(int argc, char **argv){
    ros::init(argc, argv, "link1_node");
    ros::NodeHandle link1_nh;
    //int SPS = 100;
    ros::Publisher link1_pub=link1_nh.advertise<bbot_pkg::link1_msg>("link1_msg_node",BUFF_SIZE);
    ros::Subscriber
command_center_sub=command_center_nh.subscribe("command_center_msg_node",BUFF_SIZE,command_cente
r_msgCallback);
    ros::Rate loop_rate(SPS);

    double j1_position_curr=0;
    double j1_position_prev=0;
    double j1_speed_curr=0;
    double j1_speed_prev=0;
    double j1_accel = 0;

    int eqep1_prev = 0;
    int eqep1_curr = 0;

    eqep1_prev = eqep.getValue(0);
    //eqep1_prev = 10;
    //eqep2_prev = 20;

    while(ros::ok())
    {
        i++;
        bbot_pkg::link1_msg link1_msg;
        link1_msg.link1_count=i;

        //sense
        eqep1_curr = eqep.getValue(0);

        //pos
        j1_position_curr = j1_position_prev + (eqep1_curr-eqep1_prev) * GEAR1_RATIO; // map to degrees
        //speed
        j1_speed_curr = (j1_position_curr - j1_position_prev) * SPS;
        //accel
        j1_accel = (j1_speed_curr - j1_speed_prev) * SPS;
        //reset vars
        eqep1_prev = eqep1_curr;

        j1_position_prev = j1_position_curr;

        j1_speed_prev = j1_speed_curr;

```

```

// receive messages
ros::spin();

// pwm output mapping/limits
pwm.reconfig(JOINT1_PWM_CHANNEL, joint1.period, joint1.dutycycle);
//

// release ball
if (release == true) {
    while(joint1.position >= 0) {
        pwm.reconfig(JOINT1_PWM_CHANNEL, joint1.period, 1000000*0.1);

    }
}

//publish
link1_msg.link1_position = j1_position_curr;
link1_msg.link1_speed = j1_speed_curr;
link1_msg.link1_accel = j1_accel;
link1_msg.link1_dutycycle = j1_dutycycle;
link1_msg.link1_period = j1_period;

    ROS_INFO("i: %d j1_pos: %f j1_speed: %f j1_accel: %f j1_dutycycle: %i
",i,j1_position_curr,j1_speed_curr,j1_accel_curr,j1_dutycycle);
    sensing_pub.publish(link1_msg);
    loop_rate.sleep();

}
return 0;

}

////////////////////////////////eQEP CLASS METHODS////////////////////////////////

eQEP::eQEP() {
    cout << "eQEP Class Initiated" << endl;

    eQEP0Export = "/sys/devices/ocp/48304000.epwmss";
    eQEP0Unexport = "/sys/devices/ocp/48304000.epwmss";

    eQEP1Export = "/sys/devices/platform/ocp/48302000.epwmss";
    eQEP1Unexport = "/sys/devices/platform/ocp/48302000.epwmss";

    eQEP0Handle = NULL;
    eQEP0Position = "/sys/devices/platform/ocp/48304000.epwmss/48304180.eqep/position";

```



```

eQEP1Handle = NULL;
eQEP1Position = "/sys/devices/platform/ocp/48302000.epwmss/48302180.eqep/position";

system("echo bone_eqep2b > /sys/devices/platform/bone_capemgr/slots");
//system("echo bone_eqep2 > /sys/devices/platform/bone_capemgr/slots");

}

void eQEP::setup(int channel) {
    cout << "eQEP SETUP" << endl;

    switch (channel) {
        case 0:
        {
            if((eQEP0Handle=fopen(eQEP0Export,"ab"))!=NULL){
                fwrite("0",sizeof(char),1,eQEP0Handle);
                fclose(eQEP0Handle);
            }

            break;
        }
        case 1:
        {
            if((eQEP1Handle=fopen(eQEP1Export,"ab"))!=NULL){
                fwrite("1",sizeof(char),1,eQEP1Handle);
                fclose(eQEP1Handle);
            }
            break;
        }

        default:
            cout << "Channel Select outside of Range (0-1)" << endl;
    }

}

int eQEP::getValue(int channel) {

    cout << "eQEP READ JOINT " << channel+1 << " : ";

    eQEP0Value =0;
    eQEP1Value =0;

```

```

char eQEP0Read[10]={0} ;
char eQEP1Read[10]={0} ;

switch (channel) {
    case 0:
    {
        if ((eQEP0Handle=fopen(eQEP0Position, "r+"))!=NULL){
            //cout <<" READING FILE...";
            fread(eQEP0Read,sizeof(char),10,eQEP0Handle);
            fclose(eQEP0Handle);
        }
        else {
            cout << "error reading file " ;
        }
        for(int i=0 ; i < 10 ; i ++ ){
            cout << eQEP0Read[i];
        }
        // cout << endl;
        stringstream ss(eQEP0Read);
        ss >> eQEP0Value;
        return eQEP0Value;
        break;
    }
    case 1:
    {
        if ((eQEP0Handle=fopen(eQEP1Position, "r+"))!=NULL){
            fread(eQEP1Read,sizeof(char),10,eQEP0Handle);
            fclose(eQEP0Handle);
        }
        for(int i=0 ; i < 10 ; i ++ ){
            cout << eQEP1Read[i];
        }
        // cout << endl;
        stringstream ss(eQEP1Read);
        ss >> eQEP1Value;
        return eQEP1Value;
        break;
    }

    default:

        cout << "Channel Select outside of Range (0-1)" << endl;
        return 1;
}
}

```

```

void eQEP::close(int channel) {
    cout << "eQEP CLOSE" << endl;

    switch (channel) {
        case 0:
        {
            if((eQEP0Handle=fopen(eQEP0Unexport,"ab"))!=NULL){
                fwrite("0",sizeof(char),1,eQEP0Handle);
                fclose(eQEP0Handle);
            }

            break;
        }
        case 1:
        {
            if((eQEP1Handle=fopen(eQEP1Unexport,"ab"))!=NULL){
                fwrite("1",sizeof(char),1,eQEP1Handle);
                fclose(eQEP1Handle);
            }

            break;
        }

        default:

            cout << "Channel Select outside of Range (0-1)" << endl;
    }

}

```

//////////////////////////////////PWM CLASS METHODS//////////////////////////////////

```

PWM::PWM() {
    PWMCape = "/sys/devices/platform/bone_capemgr/slots";

    PWMExport = "/sys/class/pwm/pwmchip0/export";

    PWM0Handle = NULL;
    PWM0Period = "/sys/class/pwm/pwmchip0/pwm0/period";
    PWM0Dutycycle = "/sys/class/pwm/pwmchip0/pwm0/duty_cycle";
    PWM0Enable = "/sys/class/pwm/pwmchip0/pwm0/enable";
}

```

```

PWM1Handle = NULL;
PWM1Period = "/sys/class/pwm/pwmchip0/pwm1/period";
PWM1Dutycycle = "/sys/class/pwm/pwmchip0/pwm1/duty_cycle";
PWM1Enable = "/sys/class/pwm/pwmchip0/pwm1/enable";

PWM2Handle = NULL;
PWM2Period = "/sys/class/pwm/pwmchip0/pwm2/period";
PWM2Dutycycle = "/sys/class/pwm/pwmchip0/pwm2/duty_cycle";
PWM2Enable = "/sys/class/pwm/pwmchip0/pwm2/enable";

PWM3Handle = NULL;
PWM3Period = "/sys/class/pwm/pwmchip0/pwm3/period";
PWM3Dutycycle = "/sys/class/pwm/pwmchip0/pwm3/duty_cycle";
PWM3Enable = "/sys/class/pwm/pwmchip0/pwm3/enable";

PWMUnexport = "/sys/class/pwm/pwmchip0/unexport";

system("echo BB-PWM2 > /sys/devices/platform/bone_capemgr/slots");
printf("PWMCape ON\n");

}

```

```

int PWM::setup(int channel, int period, int dutycycle) {
    cout << "PWM SETUP" << channel << endl;

    switch (channel) {
        case 0:
        {
            if((PWM0Handle=fopen(PWMLExport,"ab"))!=NULL){
                fwrite("0",sizeof(char),1,PWM0Handle);
                fclose(PWM0Handle);
            }
            if ((PWM0Handle=fopen(PWM0Period, "r+"))!=NULL){
                fwrite("1000000",sizeof(char),7,PWM0Handle);
                fclose(PWM0Handle);
            }

            if ((PWM0Handle=fopen(PWM0Dutycycle, "r+"))!=NULL){
                fwrite("0",sizeof(char),1,PWM0Handle);
                fclose(PWM0Handle);
            }
            return 0;
            break;
        }
        case 1:

```

```

{

    if((PWM1Handle=fopen(PWMExport,"ab"))!=NULL){
        fwrite("1",sizeof(char),1,PWM1Handle);
        fclose(PWM1Handle);
    }
    if ((PWM1Handle=fopen(PWM1Period, "r+"))!=NULL){
        fwrite("1000000",sizeof(char),7,PWM1Handle);
        fclose(PWM1Handle);
    }

    if ((PWM1Handle=fopen(PWM1Dutycycle, "r+"))!=NULL){
        fwrite("0",sizeof(char),1,PWM1Handle);
        fclose(PWM1Handle);
    }
    return 0;

    break;
}

default:
    return 1;
    cout << "Channel Select outside of Range (0-1)" << endl;
}
}

```

```

int PWM::reconfig(int channel, int period, int dutycycle) {

    cout << "PWM RECONFIG" << endl;

    char periodStr[20] = {0};
    char dutycycleStr[20] = {0};
    sprintf(periodStr,"%d",period);
    sprintf(dutycycleStr,"%d",dutycycle);

    switch (channel) {
        case 0:
        {
            if ((PWM0Handle=fopen(PWM0Period, "r+"))!=NULL){
                fwrite(periodStr,sizeof(char),strlen(periodStr),PWM0Handle);
                fclose(PWM0Handle);
            }
            if ((PWM0Handle=fopen(PWM0Dutycycle, "r+"))!=NULL){
                fwrite("0",sizeof(char),strlen(dutycycleStr),PWM0Handle);
                fclose(PWM0Handle);
            }
        }
    }
}

```

```

        return 0;
        break;
    }
    case 1:
    {
        if ((PWM1Handle=fopen(PWM1Period, "r+"))!=NULL){
            fwrite(periodStr,sizeof(char),strlen(periodStr),PWM1Handle);
            fclose(PWM1Handle);
        }
        if ((PWM1Handle=fopen(PWM1Dutycycle, "r+"))!=NULL){
            fwrite("0",sizeof(char),strlen(dutycycleStr),PWM1Handle);
            fclose(PWM1Handle);
        }
        return 0;
        break;
    }

    default:
        return 1;
        cout << "Channel Select outside of Range (0-1)" << endl;
    }
}

```

```

int PWM::start(int channel) {
    cout << "PWM START" << endl;

    switch (channel) {
        case 0:
        {
            if ((PWM0Handle=fopen(PWM0Enable, "r+"))!=NULL){
                fwrite("1",sizeof(char),1,PWM0Handle);
                fclose(PWM0Handle);
            }
            return 0;
            break;
        }
        case 1:
        {
            if ((PWM1Handle=fopen(PWM1Enable, "r+"))!=NULL){
                fwrite("1",sizeof(char),1,PWM1Handle);
                fclose(PWM1Handle);
            }
            return 0;

            break;
        }
    }
}

```

```

        default:
            cout << "Channel Select outside of Range (0-1)" << endl;
            return 1;
    }

}

void PWM::stop(int channel) {
    cout << "PWM STOP" << endl;

    switch (channel) {
        case 0:
        {
            if ((PWM0Handle=fopen(PWM0Enable, "r+"))!=NULL){
                fwrite("0",sizeof(char),1,PWM0Handle);
                fclose(PWM0Handle);
            }

            break;
        }
        case 1:
        {
            if ((PWM1Handle=fopen(PWM1Enable, "r+"))!=NULL){
                fwrite("0",sizeof(char),1,PWM1Handle);
                fclose(PWM1Handle);
            }

            break;
        }

        default:

            cout << "Channel Select outside of Range (0-1)" << endl;
    }

}

}

void PWM::close(int channel) {
    cout << "PWM CLOSE" << endl;

    switch (channel) {
        case 0:
        {
            if ((PWM0Handle=fopen(PWM0Enable, "r+"))!=NULL){
                fwrite("0",sizeof(char),1,PWM0Handle);

```

```

        fclose(PWM0Handle);
    }
    if((PWM0Handle=fopen(PWMUnexport,"ab"))!=NULL){
        fwrite("0",sizeof(char),1,PWM0Handle);
        fclose(PWM0Handle);
    }

    break;
}
case 1:
{
    if ((PWM1Handle=fopen(PWM1Enable, "r+"))!=NULL){
        fwrite("0",sizeof(char),1,PWM1Handle);
        fclose(PWM1Handle);
    }
    if((PWM1Handle=fopen(PWMUnexport,"ab"))!=NULL){
        fwrite("1",sizeof(char),1,PWM1Handle);
        fclose(PWM1Handle);
    }

    break;
}

default:

    cout << "Channel Select outside of Range (0-1)" << endl;
}

}

```



```

//
// link1_node.h
//
//
// Created by Austin Burch on 11/1/16.
//
//

#ifndef ____driving_node__
#define ____driving_node__

#include <iostream>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sstream>

#include "ros/ros.h"
#include "bbot_pkg/driving_msg.h"
#include "bbot_pkg/sensing_msg.h"
#include "bbot_pkg/command_center_msg.h"
// #include "/home/ubuntu/BBB_lib/BBB_lib.h"
#define PERIOD_INIT 1000000
#define MAX 48
#define SPS 100
#define BUFF_SIZE 100
#define JOINT1_PWM_CHANNEL 0
#define JOINT2_PWM_CHANNEL 1

class PWM{

    const char *PWMCape

    const char *PWMExport;
    const char *PWMUnexport;

    FILE *PWM0Handle;
    const char *PWM0Period;
    const char *PWM0Dutycycle;
    const char *PWM0Enable;

    FILE *PWM1Handle;
    const char *PWM1Period;
    const char *PWM1Dutycycle;
    const char *PWM1Enable;

    FILE *PWM2Handle;
    const char *PWM2Period;
    const char *PWM2Dutycycle;

```

```

const char *PWM2Enable;

FILE *PWM3Handle;
const char *PWM3Period;
const char *PWM3Dutycycle;
const char *PWM3Enable;

public:
    PWM();
    int setup(int channel, int period, int dutycycle);
    int reconfig(int channel, int period, int dutycycle);
    int start(int channel);
    void stop(int channel);
    void close(int channel);

};

#endif /* defined(____driving_node__) */

```

```

//
// link2_node.cpp
//
//
// Created by Austin Burch on 11/1/16.
//
//

#include "link2_node.h"
// #include "BBB_lib.h"

using namespace std;

eQEP eqep;
PWM pwm;

double command_center_time = 0;
int link2_count = 0;
// joint vars structure
struct Joint {
    float position = 0, speed = 0, accel = 0;
    float K_p=0, K_i=0, K_d=0;
    float e=0, P_val=0, I_val=0, D_val=0, PID_val=0;
    float prev_e=0, prev_I=0;
    int period=PERIOD_INIT,dutycycle=0,direction;
};
Joint joint2;

struct EndEff {
    float xPos;
    float yPos;
    float zPos;
    float vel;
    float vel_e, prev_vel_e;
};

EndEff gripper;

void command_center_msgCallback(const bbot_pkg::command_center_msg::ConstPtr& command_center_msg)
{
    command_center_time = command_center_msg->command_center_time;
    joint2.dutycycle = command_center_msg->command_center_j2_dutycycle;
    joint2.direction = command_center_msg->command_center_j2_direction;
    release = command_center_msg->command_center_release
}

```

```

int main(int argc, char **argv){
    ros::init(argc, argv, "link2_node");
    ros::NodeHandle link2_nh;
    //int SPS = 100;
    ros::Publisher link2_pub=link2_nh.advertise<bbot_pkg::link2_msg>("link2_msg_node", BUFF_SIZE);
    ros::Subscriber
    command_center_sub=command_center_nh.subscribe("command_center_msg_node", BUFF_SIZE, command_center_msgCallback);
    ros::Rate loop_rate(SPS);

    double j2_position_curr=0;
    double j2_position_prev=0;
    double j2_speed_curr=0;
    double j2_speed_prev=0;
    double j2_accel = 0;

    int eqep2_prev = 0;
    int eqep2_curr = 0;

    eqep2_prev = eqep.getValue(0);
    //eqep1_prev = 10;
    //eqep2_prev = 20;

    // GPIO Initialize magnet as on////////
    FILE *FILEHandle=NULL;
    char setValue[4], GPIOString[4], GPIOValue[MAX], GPIODirection[MAX];
    sprintf(GPIOString, "%d", 60);
    sprintf(GPIOValue, "/sys/class/gpio/gpio%d/value", 60);
    sprintf(GPIODirection, "/sys/class/gpio/gpio%d/direction", 60);

    if((FILEHandle=fopen("/sys/class/gpio/export", "ab"))==NULL){
        printf("Cannot export the GPIO pin,\n");
    }
    strcpy(setValue, GPIOString);
    fwrite(&setValue, sizeof(char), 2, FILEHandle);
    fclose(FILEHandle);

    // direction
    if((FILEHandle=fopen(GPIODirection, "rb+"))==NULL){
        printf("Cannot open direction handle.\n");
    }
    strcpy(setValue, "out");
    fwrite(&setValue, sizeof(char), 3, FILEHandle);
    fclose(FILEHandle);
    printf("direction written successfully");
    if((FILEHandle=fopen(GPIOValue, "rb+"))==NULL){
        printf("Cannot open value handle.\n");
    }

```

```

    }
    printf("debug1");
    strcpy(setValue,"1");
    fwrite(&setValue,sizeof(char),1,FILEHandle);
    fclose(FILEHandle);
    printf("debug2");
    //////////////////////////////////////

    while(ros::ok())
    {
    i++;
    bbot_pkg::link2_msg link2_msg;
    link1_msg.link2_count=i;

    //sense
    eqep2_curr = eqep.getValue(0);

    //pos
    j2_position_curr = j2_position_prev + (eqep2_curr-eqep2_prev) * GEAR2_RATIO; // map to degrees
    //speed
    j2_speed_curr = (j2_position_curr - j2_position_prev) * SPS;
    //accel
    j2_accel = (j2_speed_curr - j2_speed_prev) * SPS;
    //reset vars
    eqep2_prev = eqep2_curr;

    j2_position_prev = j2_position_curr;

    j2_speed_prev = j2_speed_curr;

    // receive messages
    ros::spin();

    // pwm output mapping/limits
    pwm.reconfig(JOINT1_PWM_CHANNEL, joint2.period, joint2.dutycycle);
    //

    // release ball
    if (release == true) {

        FILEHandle=fopen(GPIOValue,"rb+");
        strcpy(setValue,"0");
        fwrite(&setValue,sizeof(char),1,FILEHandle);
        fclose(FILEHandle);

        while(joint2.position >= 0) {
            pwm.reconfig(JOINT1_PWM_CHANNEL, joint2.period, 1000000*0.1);
        }
    }

```

```

    }

    //publish
    link2_msg.link2_position = j2_position_curr;
    link2_msg.link2_speed = j2_speed_curr;
    link2_msg.link2_accel = j2_accel;
    link2_msg.link2_dutycycle = j2_dutycycle;
    link2_msg.link2_period = j2_period;

    ROS_INFO("i: %d j2_pos: %f j2_speed: %f j2_accel: %f j2_dutycycle: %i",
    i,j2_position_curr,j2_speed_curr,j2_accel_curr,j2_dutycycle);
    sensing_pub.publish(link2_msg);
    loop_rate.sleep();

    }
    return 0;
}

////////////////////////////////eQEP CLASS METHODS////////////////////////////////

eQEP::eQEP() {
    cout << "eQEP Class Initiated" << endl;

    eQEP0Export = "/sys/devices/ocp/48304000.epwmss";
    eQEP0Unexport = "/sys/devices/ocp/48304000.epwmss";

    eQEP1Export = "/sys/devices/platform/ocp/48302000.epwmss";
    eQEP1Unexport = "/sys/devices/platform/ocp/48302000.epwmss";

    eQEP0Handle = NULL;
    eQEP0Position = "/sys/devices/platform/ocp/48304000.epwmss/48304180.eqep/position";

    eQEP1Handle = NULL;
    eQEP1Position = "/sys/devices/platform/ocp/48302000.epwmss/48302180.eqep/position";

    system("echo bone_eqep2b > /sys/devices/platform/bone_capemgr/slots");
    //system("echo bone_eqep2 > /sys/devices/platform/bone_capemgr/slots");
}

```

```

void eQEP::setup(int channel) {
    cout << "eQEP SETUP" << endl;

    switch (channel) {
        case 0:
        {

            if((eQEP0Handle=fopen(eQEP0Export,"ab"))!=NULL){
                fwrite("0",sizeof(char),1,eQEP0Handle);
                fclose(eQEP0Handle);
            }

            break;
        }
        case 1:
        {

            if((eQEP1Handle=fopen(eQEP1Export,"ab"))!=NULL){
                fwrite("1",sizeof(char),1,eQEP1Handle);
                fclose(eQEP1Handle);
            }
            break;
        }

        default:
            cout << "Channel Select outside of Range (0-1)" << endl;
    }

}

```

```

int eQEP::getValue(int channel) {

    cout << "eQEP READ JOINT " << channel+1 << " : ";

    eQEP0Value =0;
    eQEP1Value =0;

    char eQEP0Read[10]={0} ;
    char eQEP1Read[10]={0} ;

    switch (channel) {
        case 0:
        {
            if ((eQEP0Handle=fopen(eQEP0Position, "r+"))!=NULL){

```

```

        //cout << " READING FILE...";
        fread(eQEP0Read,sizeof(char),10,eQEP0Handle);
        fclose(eQEP0Handle);
    }
    else {
        cout << "error reading file " ;
    }
    for(int i=0 ; i < 10 ; i ++ ){
        cout << eQEP0Read[i];
    }
    // cout << endl;
    stringstream ss(eQEP0Read);
    ss >> eQEP0Value;
    return eQEP0Value;
    break;
}
case 1:
{
    if ((eQEP0Handle=fopen(eQEP1Position, "r+"))!=NULL){
        fread(eQEP1Read,sizeof(char),10,eQEP0Handle);
        fclose(eQEP0Handle);
    }
    for(int i=0 ; i < 10 ; i ++ ){
        cout << eQEP1Read[i];
    }
    // cout << endl;
    stringstream ss(eQEP1Read);
    ss >> eQEP1Value;
    return eQEP1Value;
    break;
}

default:

    cout << "Channel Select outside of Range (0-1)" << endl;
    return 1;
}
}

```

```

void eQEP::close(int channel) {
    cout << "eQEP CLOSE" << endl;

```

```

switch (channel) {
    case 0:

```



```

{
    if((eQEP0Handle=fopen(eQEP0Unexport,"ab"))!=NULL){
        fwrite("0",sizeof(char),1,eQEP0Handle);
        fclose(eQEP0Handle);
    }

    break;
}
case 1:
{
    if((eQEP1Handle=fopen(eQEP1Unexport,"ab"))!=NULL){
        fwrite("1",sizeof(char),1,eQEP1Handle);
        fclose(eQEP1Handle);
    }

    break;
}

default:

    cout << "Channel Select outside of Range (0-1)" << endl;
}

}

```

////////////////////////////////PWM CLASS METHODS////////////////////////////////

```

PWM::PWM() {
    PWMCape = "/sys/devices/platform/bone_capemgr/slots";

    PWMExport = "/sys/class/pwm/pwmchip0/export";

    PWM0Handle = NULL;
    PWM0Period = "/sys/class/pwm/pwmchip0/pwm0/period";
    PWM0Dutycycle = "/sys/class/pwm/pwmchip0/pwm0/duty_cycle";
    PWM0Enable = "/sys/class/pwm/pwmchip0/pwm0/enable";

    PWM1Handle = NULL;
    PWM1Period = "/sys/class/pwm/pwmchip0/pwm1/period";
    PWM1Dutycycle = "/sys/class/pwm/pwmchip0/pwm1/duty_cycle";
    PWM1Enable = "/sys/class/pwm/pwmchip0/pwm1/enable";

    PWM2Handle = NULL;
    PWM2Period = "/sys/class/pwm/pwmchip0/pwm2/period";
}

```

```

PWM2Dutycycle = "/sys/class/pwm/pwmchip0/pwm2/duty_cycle";
PWM2Enable = "/sys/class/pwm/pwmchip0/pwm2/enable";

PWM3Handle = NULL;
PWM3Period = "/sys/class/pwm/pwmchip0/pwm3/period";
PWM3Dutycycle = "/sys/class/pwm/pwmchip0/pwm3/duty_cycle";
PWM3Enable = "/sys/class/pwm/pwmchip0/pwm3/enable";

PWMUnexport = "/sys/class/pwm/pwmchip0/unexport";

system("echo BB-PWM2 > /sys/devices/platform/bone_capemgr/slots");
printf("PWMCape ON\n");

}

```

```

int PWM::setup(int channel, int period, int dutycycle) {
    cout << "PWM SETUP" << channel << endl;

    switch (channel) {
        case 0:
        {
            if((PWM0Handle=fopen(PWMExport,"ab"))!=NULL){
                fwrite("0",sizeof(char),1,PWM0Handle);
                fclose(PWM0Handle);
            }
            if ((PWM0Handle=fopen(PWM0Period, "r+"))!=NULL){
                fwrite("1000000",sizeof(char),7,PWM0Handle);
                fclose(PWM0Handle);
            }

            if ((PWM0Handle=fopen(PWM0Dutycycle, "r+"))!=NULL){
                fwrite("0",sizeof(char),1,PWM0Handle);
                fclose(PWM0Handle);
            }
            return 0;
            break;
        }
        case 1:
        {
            if((PWM1Handle=fopen(PWMExport,"ab"))!=NULL){
                fwrite("1",sizeof(char),1,PWM1Handle);
                fclose(PWM1Handle);
            }
            if ((PWM1Handle=fopen(PWM1Period, "r+"))!=NULL){
                fwrite("1000000",sizeof(char),7,PWM1Handle);
            }

```

```

        fclose(PWM1Handle);
    }

    if ((PWM1Handle=fopen(PWM1Dutycycle, "r+"))!=NULL){
        fwrite("0",sizeof(char),1,PWM1Handle);
        fclose(PWM1Handle);
    }
    return 0;

    break;
}

default:
    return 1;
    cout << "Channel Select outside of Range (0-1)" << endl;
}

}

int PWM::reconfig(int channel, int period, int dutycycle) {

    cout << "PWM RECONFIG" << endl;

    char periodStr[20] = {0};
    char dutycycleStr[20] = {0};
    sprintf(periodStr,"%d",period);
    sprintf(dutycycleStr,"%d",dutycycle);

    switch (channel) {
        case 0:
        {
            if ((PWM0Handle=fopen(PWM0Period, "r+"))!=NULL){
                fwrite(periodStr,sizeof(char),strlen(periodStr),PWM0Handle);
                fclose(PWM0Handle);
            }
            if ((PWM0Handle=fopen(PWM0Dutycycle, "r+"))!=NULL){
                fwrite("0",sizeof(char),strlen(dutycycleStr),PWM0Handle);
                fclose(PWM0Handle);
            }
            return 0;
            break;
        }
        case 1:
        {
            if ((PWM1Handle=fopen(PWM1Period, "r+"))!=NULL){
                fwrite(periodStr,sizeof(char),strlen(periodStr),PWM1Handle);
                fclose(PWM1Handle);
            }

```

```

    }
    if ((PWM1Handle=fopen(PWM1Dutycycle, "r+"))!=NULL){
        fwrite("0",sizeof(char),strlen(dutycycleStr),PWM1Handle);
        fclose(PWM1Handle);
    }
    return 0;
    break;
}

default:
    return 1;
    cout << "Channel Select outside of Range (0-1)" << endl;
}
}

```

```

int PWM::start(int channel) {
    cout << "PWM START" << endl;

    switch (channel) {
        case 0:
        {
            if ((PWM0Handle=fopen(PWM0Enable, "r+"))!=NULL){
                fwrite("1",sizeof(char),1,PWM0Handle);
                fclose(PWM0Handle);
            }
            return 0;
            break;
        }
        case 1:
        {
            if ((PWM1Handle=fopen(PWM1Enable, "r+"))!=NULL){
                fwrite("1",sizeof(char),1,PWM1Handle);
                fclose(PWM1Handle);
            }
            return 0;

            break;
        }

        default:
            cout << "Channel Select outside of Range (0-1)" << endl;
            return 1;
        }
    }
}

```

```

void PWM::stop(int channel) {

```

```

cout << "PWM STOP" << endl;

switch (channel) {
    case 0:
    {
        if ((PWM0Handle=fopen(PWM0Enable, "r+"))!=NULL){
            fwrite("0",sizeof(char),1,PWM0Handle);
            fclose(PWM0Handle);
        }

        break;
    }
    case 1:
    {
        if ((PWM1Handle=fopen(PWM1Enable, "r+"))!=NULL){
            fwrite("0",sizeof(char),1,PWM1Handle);
            fclose(PWM1Handle);
        }

        break;
    }

    default:

        cout << "Channel Select outside of Range (0-1)" << endl;
}

}

```

```

void PWM::close(int channel) {
    cout << "PWM CLOSE" << endl;

    switch (channel) {
        case 0:
        {
            if ((PWM0Handle=fopen(PWM0Enable, "r+"))!=NULL){
                fwrite("0",sizeof(char),1,PWM0Handle);
                fclose(PWM0Handle);
            }
            if((PWM0Handle=fopen(PWMUnexport,"ab"))!=NULL){
                fwrite("0",sizeof(char),1,PWM0Handle);
                fclose(PWM0Handle);
            }

            break;
        }
    }
}

```

```

    }
    case 1:
    {
        if ((PWM1Handle=fopen(PWM1Enable, "r+"))!=NULL){
            fwrite("0",sizeof(char),1,PWM1Handle);
            fclose(PWM1Handle);
        }
        if((PWM1Handle=fopen(PWMUnexport,"ab"))!=NULL){
            fwrite("1",sizeof(char),1,PWM1Handle);
            fclose(PWM1Handle);
        }

        break;
    }

    default:

        cout << "Channel Select outside of Range (0-1)" << endl;
    }

}

```

```

//
// link2_node.h
//
//
// Created by Austin Burch on 11/1/16.
//
//

#ifndef ____link2_node__
#define ____link2_node__

#include <iostream>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sstream>

#include "ros/ros.h"
#include "bbot_pkg/link1_msg.h"
#include "bbot_pkg/link2_msg.h"
#include "bbot_pkg/command_center_msg.h"
// #include "/home/ubuntu/BBB_lib/BBB_lib.h"

#define SPS 100
#define BUFF_SIZE 100
#define GEAR1_RATIO 74.83
#define GEAR2_RATIO 74.83

#define PERIOD_INIT 1000000
#define MAX 48
#define SPS 100
#define BUFF_SIZE 100
#define JOINT1_PWM_CHANNEL 0
#define JOINT2_PWM_CHANNEL 1

class PWM{

    const char *PWMCape;

    const char *PWMExport;
    const char *PWMUnexport;

    FILE *PWM0Handle;
    const char *PWM0Period;
    const char *PWM0Dutycycle;
    const char *PWM0Enable;

    FILE *PWM1Handle;

```

```

const char *PWM1Period;
const char *PWM1Dutycycle;
const char *PWM1Enable;

FILE *PWM2Handle;
const char *PWM2Period;
const char *PWM2Dutycycle;
const char *PWM2Enable;

FILE *PWM3Handle;
const char *PWM3Period;
const char *PWM3Dutycycle;
const char *PWM3Enable;

public:
    PWM();
    int setup(int channel, int period, int dutycycle);
    int reconfig(int channel, int period, int dutycycle);
    int start(int channel);
    void stop(int channel);
    void close(int channel);

};

class eQEP {

    const char *eQEP0Export;
    const char *eQEP0Unexport;
    const char *eQEP1Export;
    const char *eQEP1Unexport;

    FILE *eQEP0Handle;
    const char *eQEP0Enable;
    const char *eQEP0Position;
    // char *eQEP0Read;

    FILE *eQEP1Handle;
    const char *eQEP1Enable;
    const char *eQEP1Position;
    // char *eQEP1Read;

    int eQEP0Value;
    int eQEP1Value;

public:
    eQEP();
    void setup(int channel);
    int getValue(int channel);

```



```
    void close(int channel);  
};  
  
#endif /* defined(____link2_node__) */
```

```

//
// command_center_node.h
//
//
// Created by Austin Burch on 11/1/16.
//
//

#ifndef ____command_center_node__
#define ____command_center_node__

#include <iostream>
#include <math.h>
#include "ros/ros.h"
#include "bbot_pkg/link1_msg.h"
#include "bbot_pkg/link2_msg.h"
#include "bbot_pkg/command_center_msg.h"

#define PERIOD_INIT 1000000
#define SPS 100
#define BUFF_SIZE 100
// #include "/home/ubuntu/BBB_lib/BBB_lib.h"

#endif /* defined(____command_center_node__) */

```

```

//
// command_center_node.cpp
//
//
// Created by Austin Burch on 11/1/16.
//
//

#include "command_center_node.h"

using namespace std;

double command_center_time = 0;
int sensing_count = 0;
int driving_count = 0;

double Hm = 1; //table height
double q1 = 15;
double q2 = 30;
double qf = 45;

// joint vars structure
struct Joint {
    float position = 0, speed = 0, accel = 0;
    float K_p=0, K_i=0, K_d=0;
    float e=0, P_val=0, I_val=0, D_val=0, PID_val=0;
    float prev_e=0, prev_I=0;
    int period=PERIOD_INIT,dutycycle=0,direction;
};
Joint joint1, joint2;

struct EndEff {
    float xPos;
    float yPos;
    float zPos;
    float vel;
    float vel_e, prev_vel_e;

};
EndEff endeff;

struct Hoop {
    double theta, xPos_G, yPos_G, xPos_R, yPos_R;

```

```

};
Hoop hoop;

double ball_vx0 = 0;
double ball_vy0 = 0;

// Callback Function for link1_node
void link1_msgCallback(const bbot_pkg::link1_msg::ConstPtr& link1_msg)
{
    link1_count = link1_msg->link1_count;

    joint1.position = link1_msg->link1_position;
    joint1.speed = link1_msg->link1_speed;
    joint1.accel = link1_msg->link1_accel;

    joint1.period = link1_msg->link1_period;
    joint1.dutycycle = link1_msg->link1_dutycycle;

    ROS_INFO("i: %d j1_pos: %f j1_speed: %f j1_accel: %f j1_dutycycle: %i",
    i,joint1.position,joint1.speed,joint1.accel,joint1.dutycycle);
}

// Callback Function for link2_node
void link2_msgCallback(const bbot_pkg::link2_msg::ConstPtr& link2_msg)
{
    link2_count = link2_msg->link2_count;

    joint2.position = link2_msg->link2_position;
    joint2.speed = link2_msg->link2_speed;
    joint2.accel = link2_msg->link2_accel;

    joint2.period = link2_msg->link2_period;
    joint2.dutycycle = link2_msg->link2_dutycycle;

    ROS_INFO("i: %d j1_pos: %f j1_speed: %f j1_accel: %f j1_dutycycle: %i",
    i,joint2.position,joint2.speed,joint2.accel,joint2.dutycycle);
}

// Main Routine
int main(int argc,char **argv){

    // setup comm
    ros::init(argc,argv,"command_center_node");

```

```

    ros::NodeHandle link1_nh;
    ros::NodeHandle link2_nh;
    ros::NodeHandle command_center_nh;

    ros::Publisher
    command_center_pub=command_center_nh.advertise<bbot_pkg::command_center_msg>("command_center_msg",
    BUFF_SIZE);

    ros::Subscriber link1_sub=link1_nh.subscribe("link1_msg_node",BUFF_SIZE,link1_msgCallback);
    ros::Subscriber link2_sub=link2_nh.subscribe("link2_msg_node",BUFF_SIZE,link2_msgCallback);

    ros::Rate loop_rate(SPS); // maybe dont need this? just check to see it new data has arrived?

    int command_center_count=0;

    // user inputs
    hoop.theta = 0 ; // [deg]
    hoop.xPos = 2; // [m]
    hoop.yPos = .5; // [m];

    // trajectory calculations
    hoop.xPos_G=hoop.xPos-(((3*2.54)/100)*cos(hoop.theta)); //Adjust distance for hoop (m)
    hoop.yPos_G=hoop.yPos+(3*2.54)/100*sin(hoop.theta);

    endeff.xPos_G=l1*sin(q1)+l2*sin(q1+q2);
    endeff.yPos_G=l1*cos(q1)+l2*cos(q1+q2);

    endeff.xPos_R=hoop.xPos_G-endeff.xPos_G;
    endeff.yPos_R=hoop.yPos_G-(Hm-endeff.yPos_G);

    //Calculation of Required Initial Velocity - math ref - > pow (7.0, 3.0)
    ball_vy0 = pow((GRAVITY_CONST/((endeff.xPos_R*tan(qf)-
    endeff.yPos_R)*2)),0.5)*(tan(qf)*endeff.xPos_R);
    ball_vx0 = ball_vy0/tan(qf);

    //Calculation of q1_dot & q2_dot
    //v=[-vy;vx;0;0;0;1]; // v matrix
    //J=[-l1*sin(q1)-l2*sin(q1+q2),-l2*sin(q1+q2);l1*cos(q1)+l2*cos(q1+q2),l2*cos(q1+q2);... // jacobian
    0,0;0,0;0,0;1,1];

    // q=v\J %Calculation of Angular Velocity Matrix
    //q_dot = inv(J([1 2], [1 2])) * v([1 2], [1]);

    //q_ddot1 = (q_dot(1)^2)/(2*q1);
    //q_ddot2 = (q_dot(2)^2)/(2*q2);
    //q_ddotn = [q_ddot1;q_ddot2];

```

```

int release = false ;

    while(ros::ok())
    {
// update driving and sensing vars
bbot_pkg::command_center_msg command_center_msg;
ros::spin();

// release ball
if (joint1.position + joint2.position >= 45) {
    release = true;
}

joint1.dutycycle = joint1.K_p*(q1-joint1.position) + joint1.K_d*(q1-joint1.position)
joint2.dutycycle = joint2.K_p*(q2-joint2.position) + joint2.K_d*(q2-joint2.position)

//update command center topic
    command_center_msg.command_center_time=command_center_time;
    command_center_msg.command_center_j1_dutycycle=joint1.dutycycle;
    command_center_msg.command_center_j1_direction=joint1.direction;
    command_center_msg.command_center_j2_dutycycle=joint2.dutycycle;
    command_center_msg.command_center_j2_direction=joint2.direction;
    command_center_msg.command_center_ = release

    ROS_INFO("time: %d j1_dutycycle: %f j2_dutycycle: %f, j1_direction: %d,j2_direction:
%d",time, joint1.dutycycle, joint2.dutycycle, j, joint2.direction);
    command_center_count++;
}
return 0;
}

```

Appendix F

```
//  
// command_center_node.h  
//  
//  
// Created by Austin Burch on 11/1/16.  
//  
//  
  
#ifndef ____command_center_node__  
#define ____command_center_node__  
  
#include <iostream>  
#include <math.h>  
#include "ros/ros.h"  
#include "bbot_pkg/link1_msg.h"  
#include "bbot_pkg/link2_msg.h"  
#include "bbot_pkg/command_center_msg.h"  
  
#define PERIOD_INIT 1000000  
#define SPS 100  
#define BUFF_SIZE 100  
//#include "/home/ubuntu/BBB_lib/BBB_lib.h"  
  
#endif /* defined(____command_center_node__) */
```

Appendix F:ROS bbot_pkg/msg

// link1_msg.msg

```
int32 link1_count
int32 link1_period
int32 link1_dutycycle
float64 link1_position
float64 link1_speed
float64 link1_accel
```

// link2_msg.msg

```
int32 link2_count
int32 link2_period
int32 link2_dutycycle
float64 link2_position
float64 link2_speed
float64 link2_accel
```

```
// command_center_msg.msg
float64 command_center_time
int32 command_center_j1_dutycycle
int32 command_center_j1_direction
int32 command_center_j2_dutycycle
int32 command_center_j2_direction
int32 command_center_release
```