ROBOT BATTER

Pratiksha Sharma (ps179)

1. Summary (10)

The robot's goal is to shoot as many balls as possible into the goal post, while the obstacles try to block the ball's path.

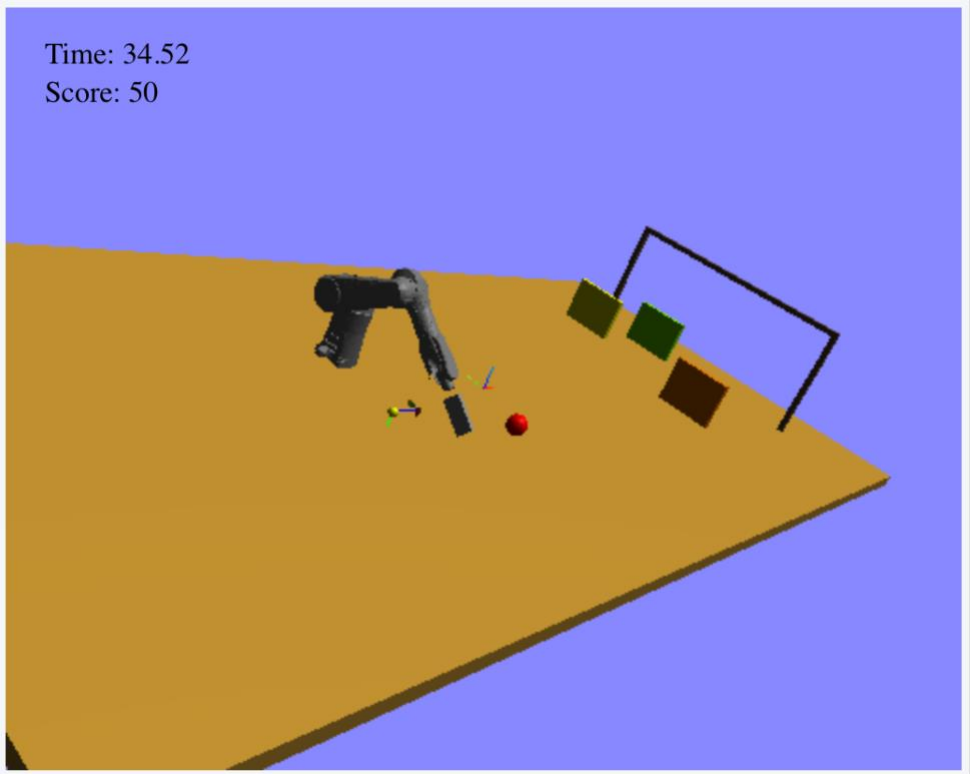

Fig 1: Figure showing the goal of the robot to shoot into the goal post past the 3 blocks

Initially, the robot arm moves to the batting position while the x-position data from the camera/blob detector is being stored for first 5 seconds. The x- position data for each block is then fitted into a sine curve using least-square optimization function in scipy. The robot has two hardcoded targets to hit, one on the left and the other on the right side of the goal post. Using the fitted sine curves, estimation of current state and prediction of future states of the blocks are computed. Based on the prediction of the state of blocks in the future, the robot decides when to shoot and which direction (i.e. left or right) to pick. The components in the robot design are shown in Fig 2.

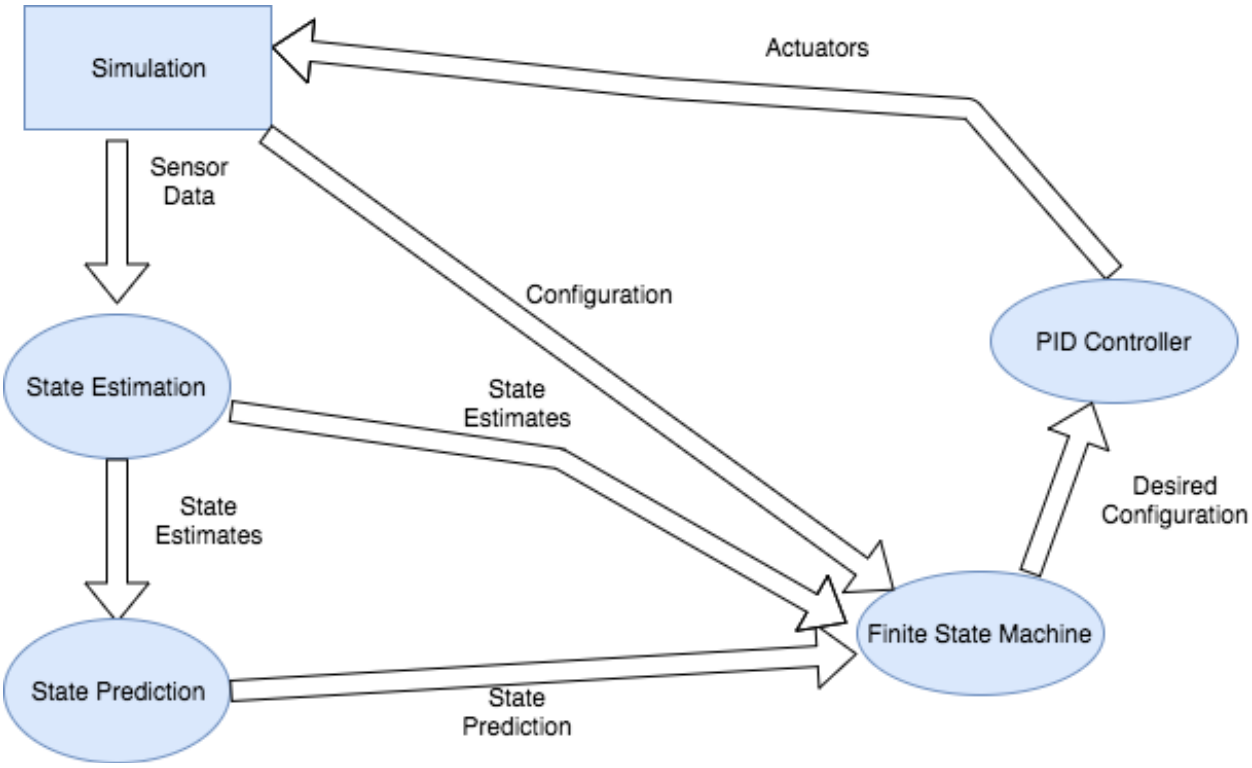

Fig 2: Components of the Robot Design

2. Components (15)

a. Simulation:

This is the world simulation that was already done. This simulates the robot environment along with the blocks, goal post and the ball. CameraBlobDetectorOutput is the output of this stage, which contains CameraBlob objects. CameraBlob is an image of the objects in the robot's environment, i.e. the ball and the 3 blocks.

b. State Estimation

The purpose of this module is to estimate the current x-position of the blocks in the world.

Inputs:

CameraBlob: an object that has width, height, color, x coordinates and y-coordinates

width, height, x-coordinates, y-coordinates: double

color: tuple of integers: r,g,b and alpha values

Initial Configuration: a list with 7 floats (6 of which are the configuration angles of the joints

Output:

Current x-position of 3 blocks; datatype: floats

A sinusoidal curve for the x-positions of each block is calculated using sinefit function of scipy. This curve is used to estimate current state.
The output from this state is then fed into state prediction.
This state is invoked every time step i.e. dt in the program. The state estimation is calculated by feeding in the time at the point of execution to the sine lambda function.
The estimates are all in camera's coordinates.

c. State Prediction

Inputs:

Sine curve for x-position of each block, datatype: lambda function
Widths of the blocks and the ball, datatype: floats
Current State estimation: x-positions of the blocks, datatype: floats

Output:

Future x-positions of all 3 blocks, datatype: floats

The purpose of this module is to estimate the x-positions of the blocks in the given time in the future. The time it takes for the ball to travel from its initial position to the x-position of the blocks is stored in list. Based on this travel time for the ball, the future position of the blocks is calculated. The future position is calculated based on historical x-position data read in the waiting period. The module assumes that the position data of the blocks shows a sinusoidal behavior and thus bases its calculation on the input lambda sine function.
The module is invoked every time step, i.e. dt. The state prediction values are all in camera's frame.

d. Finite State Machine

The FSM is the high level controller of the robot design.
The purpose of this stage is to define various states and implement functions to control behavior of the robot arm when it is in various states. It is described in detail in part 3.
Inputs:

state estimate: x-positions of the blocks, datatype: list of 3 floats
state prediction: predicted x-positions of the blocks, datatype: list of 3 floats
sensed configuration: list of 7 floats

Output:

Desired Configuration: configuration for the robot arm, if the hitBall condition is true, datatype: list of 7 flaots

The module is invoked every time step, i.e. dt.

e. PID Controller

The purpose of this is to move the robot arm to the desired configuration with the desired joint velocities.
Input:

Desired Configuration: list of size 7, angles of joints, datatype: floats
Desired velocity: list of size 7, velocities of the joints, datatype: floats (0)

Output:

PID Controls

This module invokes API provided by the simRobotController class in Klampt. It is invoked when the FSM dictates that the robot arm be moved to a computed configuration.

3. Planning and Control Strategy. (15)

The general strategy used was to predict the positon of the blocks, hardcode the configurations that would send the ball both left and right (depending on the condition met), and bring the robot arm back in ready to hit position once 1 seconds have passed. The choice of 1 seconds is based after running the simulation multiple times. There are 2 target positions, left and right corner of the goal post. The robot chooses to either hit left or right, based on whether or not the blocks are further away from the target positons. One major failure that could occur is when the sineFit() function cannot fit a sinusoidal curve in the x-position data for the blocks. This is mitigated by providing the robot enough time (5 seconds) for data collection in the beginning of the simulation.

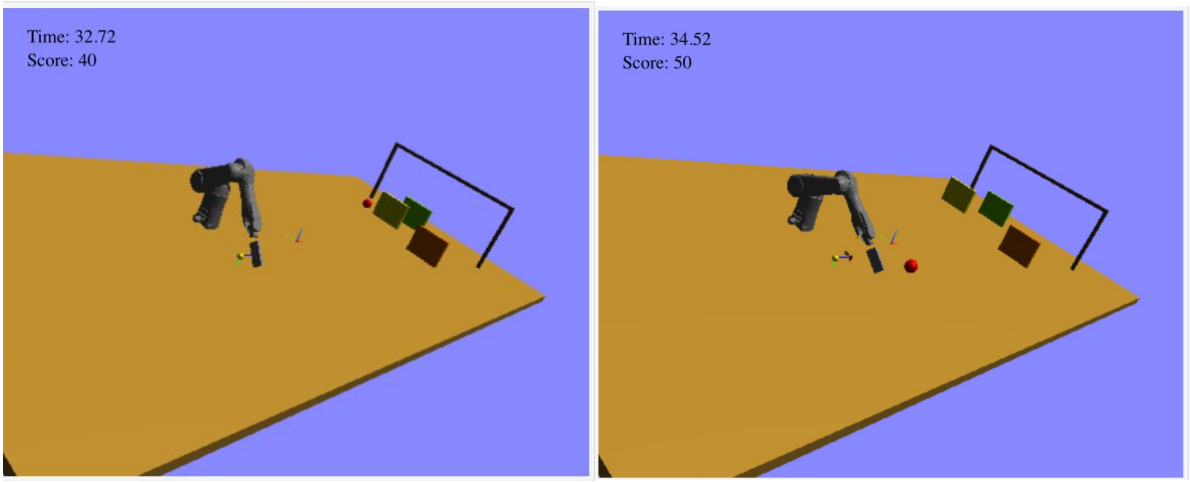

Fig 3: Figure showing that the robot arm is hitting the ball both in left (a) and right (b) corner of the goalpost.

3.1 FSM:

The Finite state machine represents the bulk of high level controller. There are major 7 states in the FSM, each a string type. For each state, the sensed configuration of the robot must be close to the desired configuration, and the joint velocities must not exceed the pre-determined values.

In order to control the joint velocities, setMilestone() function is used. His function uses a dynamic interpolant to get from the current state to the desired milestone (with optional ending velocity). This interpolant is time-optimal with respect to the velocity and acceleration bounds. At another instance, while hitting balls to the right, setLinear() function is used. The value of dt was changed manually until the joint velocities were not exceeded.

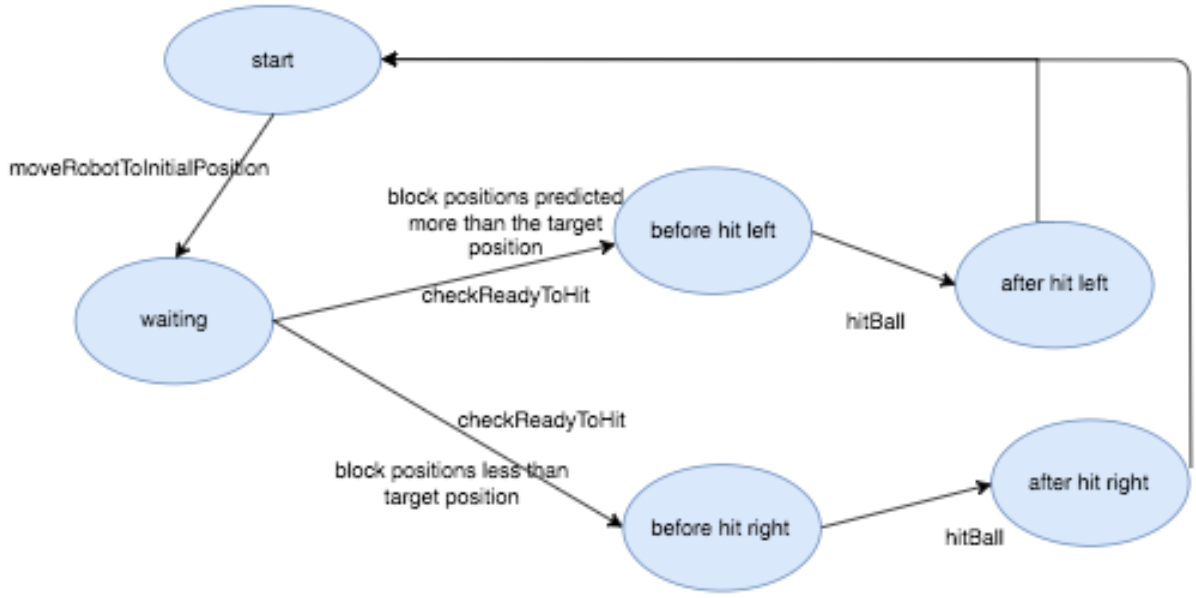

Figure 4: FSM of the robot batter design

a. Start
   The start state of the FSM is when the robot arm is moved to its initial position, i.e. closer to the ball in the configuration used for hitting the ball in the right direction. This state lasts for 5 seconds, once the simulation is run. The x-position data for the blocks are collected in this time. Once the self.time > WAIT_TIME (5 seconds), it transitions to Waiting stage.

b. Waiting
   On this stage, the historical x-position data for the blocks is fit into a sine curve and state estimation and prediction is done. Because the blocks move only along the x-axis, we only care about x-positons. During each step time, state prediction is done to figure out whether or not the conditions for hitting a ball are met. There are 2 hardcoded x-target positions, on the left and on the right. When the sine curve tells that the blocks will be further away from the target positons on the given interval time, then the condition to hit the ball is satisfied. checkReadyTohit() function takes care of this.

c. Before hit right
   Move the robot arm to the configuration for hitting the ball right
d. Before hit left
   Move the robot arm to the configuration for hitting the ball on left direction
e. After hit right
   A linear interpolation is done to move robot arm from its ‘ready to hit right’ configuration to ‘after hit right’ configuration
   State when the robot arm’s sensed configuration is equal to the hardcoded post right configuration

f. After hit left
State when the robot arm's sensed configuration is equal to the hardcoded post left configuration

The state changes to 'start' after the robot arm hits either left or right. The start stage has a condition to record data only if the time is less than WAIT_TIME.

3.2 Hardcoded path and configurations

In order to avoid conditions when IK solver does not find a possible configuration for a target, I tuned the configuration for the robot arms manually. The configuration with second joint up allowed the robot arm to avoid contact with the terrain, which is what I implemented. Also, the configuration transition when the arm has to hit left and hit right was smoother when the second link of the robot arm was up.
For each path, i.e. left or right, there is hardcoded before and after configuration. When the robot arm moves from the before configuration to after using linear interpolation, it hits the balls at an angle that makes it move either left or right. The before and after configurations had to be tuned in manually to specifically hit either left or right.

3.3 Decision to Shoot

As mentioned earlier, there are 2 hardcoded targets, LEFT_TARGET and RIGHT_TARGET in the code. These targets are the x-positions close to the left and right pole of the goal post. When the state prediction of all 3 blocks in their corresponding interval times is further away from these targets, then the robot decides to shoot the ball. When the robot arm hits the ball while moving from pre hit to post hit configuration.

4. Perception Strategy (15)

Describe your perception components in more detail. Do you need to estimate all components of 3D object positions and velocities to perform your task? To what accuracy do you expect to measure them? What strategy or algorithm do you use to implement these components? If there is a mathematical model you use, state this model. For example, if you are using a Bayesian filter, describe the transition model, observation model, and belief initialization.

The perception problem in the robot is one dimensional, on the x-axis. The y positions of the blocks are fixed. The controller does not use z-coordinates; hence the perception problem becomes much simpler. Though the camera is noisy, the perception model does not take into any variances or covariance. Also, the robot does not need instantaneous velocities in any direction, hence they are not computed. Only x-positions of the blocks need to be estimated and predicted.

4.1 Block Position Prediction

The start state of the FSM collected x-positions data for 5 seconds in the beginning of the simulation. Then sineFit function from scipy library is used to fit the data into a sinusoidal curve $f(t)=A\sin(Bt+C)+D$. The output data was tested in MATLAB to make sure that the sinusoidal curve correctly represented the changes in x-position of the blocks.

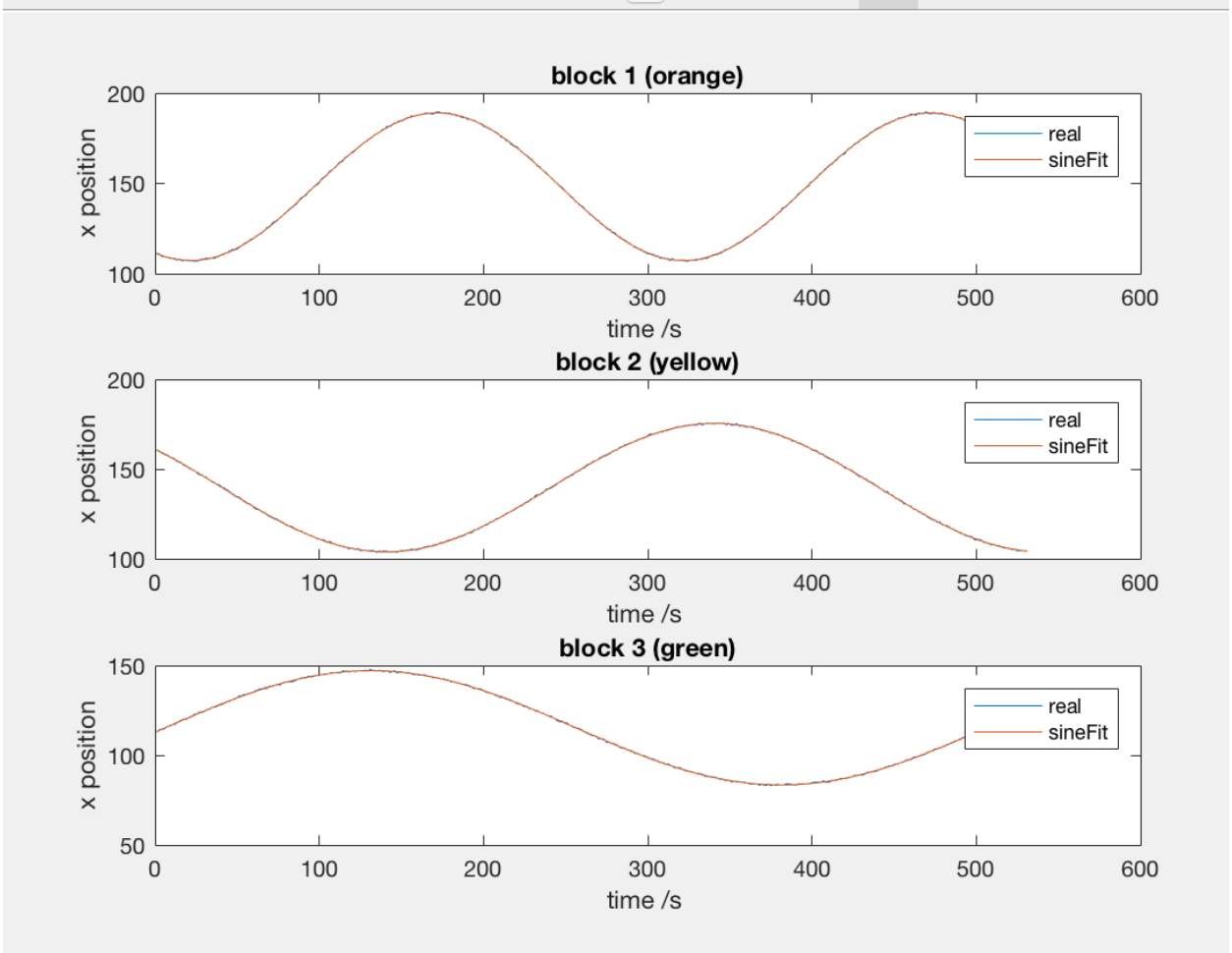

Fig 5: Plot showing the original position data and the fitted sine curve of the x-positions of the blocks

From Fig 5, we can see that the sinusoidal curve correctly models the movement of the blocks in x-direction. The error on the curve fit of each block was calculated by taking the absolute value of the maximum difference between fitted data points and original data points. The maximum errors on block1, block2 and block3 were 0.46,0.06 and 0.89 respectively.

These sine curves were used to estimate the x-positions of the blocks in the state prediction component. The time interval taken by the ball to reach y positions of 3 blocks were gathered by running the simulation frame by frame. Then state prediction of the blocks is done using this the list of these time intervals. There is a different list of time intervals when a ball is hit left and when a ball is hit right. From the simulation, we know that the blocks seek the balls. The state estimation of the blocks is also done by using these sine curves. In order to mitigate this impact, the balls are shot with the highest velocity possible. Also, a margin error is introduced in the state estimation of the blocks. This is a naïve implementation; however it seems to work.

4.2: Ball Position Estimates
Based on my implementation, it is not necessary to estimate the positon of the ball once it has been hit. Hardcoded configurations are used in the robot so that when the robot arm moves from pre hit to post hit configuration, the ball is hit and moved to the desired direction.

5. (5 pts) Reflection.

The system works well with the current implementation. I usually score 80 points while testing the simulation on medium level. All attempts made to score a goal are successful. The most difficult part of this implementation was figuring out the right configuration for the robot's arm. For both left and right direction, you had to choose the configuration parameters carefully so that the ball was hit towards the right direction, without exceeding the velocity limits on the joints. Manual fine tuning of the dt parameter to find the right time step for linear interpolation was required. Another challenge was to consider the fact that the blocks seek the ball when the ball is hit. I introduced a range of error parameter and hit the ball with the highest velocity possible to take that into account. Though this approach seems to work in testing, it is still a naïve implementation.

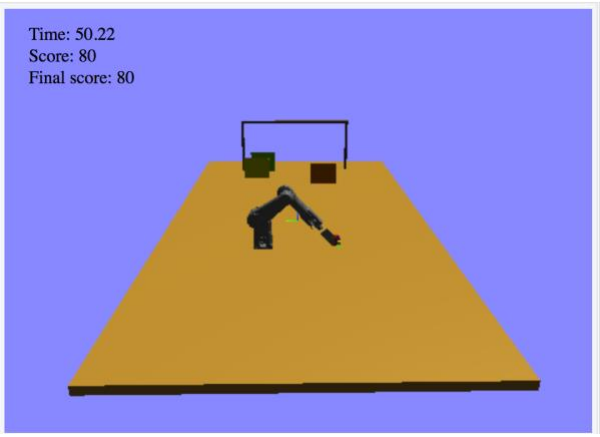

Fig 6: Plot showing the testing of robot system in medium level

I am only choosing to shoot the balls left or right, hence I am missing opportunities to shoot the ball in the middle when there is enough space between the blocks. Therefore, I don't get a score of 100 while running the simulation in medium level. An improvement on my system would be to add controls that would let the robot arm shoot on the center as well. This would increase my chances of scoring. A further improvement would be to include even more number of hardcoded paths so that the possibilities of scoring a goal improves.

Significant challenges would be incurred if this robot was to be implemented in real life. First, the noise from the camera readings in real life can throw the curve fitting off. The robot joints might not be perfectly tuned in, causing discrepancies in the simulated configuration and implemented configurations. The terrain on which the ball rolls has an impact on the ball's

trajectory. The interval times taken for the ball to reach from the initial position to the goal post would change. If the ground is uneven, there would be challenging to hit the ball in the hardcoded paths implemented in the system. As more testing is done in real life, the hardware might start wearing off, which is not a concern at all in simulation.

Here is the link to the YouTube video that shows testing of the robot.
https://www.youtube.com/watch?v=TX15ZfmWqGQ&feature=youtu.be