ECE 470 Final Project Report

Team: Eleanor Rig-B-ball

Kevin Palani (kevinrp2) Becca Napoli (rgn2)
Arkin Shah (arkinas2) Aditya Sriram (asriram3)

December 16, 2019

Abstract

Our project was to build an autonomous robot that was capable of picking up a ball and returning it to the user. The main use case was in sports, such as basketball, where a person practicing could have the ball returned to them, instead of wasting valuable practice time by chasing and picking up the ball themselves. To do this, we created a robot in the simulator "V-Rep"[1], and used many of the inbuilt tools to create our final product. Our robot was fairly sucsesfull, managing to return the ball 100% percent of the time, although sometimes slow. On average, the robot took about 3.1297 $\frac{\text{sec}}{\text{m}}$ and 1.304 $\frac{\text{sec}}{\text{rad}}$ In the end we learned a lot about the professional tools used in the real world, and have a basic understand of its capabilities. We were also able to apply many of the concepts learned in class. Future goals may include adding obstacles for the robot to go around, or have the robot track multiple balls. The code base may be found here `https://github.com/asriram3/Eleanor-Rig-B-ball`, and a video demonstration may be found here `https://drive.google.com/a/illinois.edu/file/d/1gdCs7A8ng7CPMR2xnmjBqz2qtvVEuix8/view?usp=drive_web`. Note that the video demonstration has been sent to Professor Katie Driggs, and may not be accessible to others.

1 Introduction

Our goal was to be able to pick up a ball at a location in the world and bring it back to the user. This task can be broken up into the following steps: Locating the ball, driving to the ball, picking up the ball, locating the user, driving to the user, and dropping off the ball. To do this, we looked at a variety of robots provided in the V-rep simulator, and choose the Kuka Youbot [2]. This robot features mecanum wheels, allowing it to drive side to side, in addition to the normal front to back motion. It also features a 5-axis robotic arm, as well as a carrying plate. These features made it the perfect base robot. This robot is pictured in Figure 1. To learn more about the V-rep simulator, we read through a variety of blog posts and articles, such as on how to use sensors[3] and inverse kinematics tutorials[4]. We also read through the Vrep API [5] to see what was possible within the simulator. Although possible to use more modern programming languages, the simulator seemed to be the easiest to use with Lua, so we also read the Lua documentation to understand the syntax of the language[6].

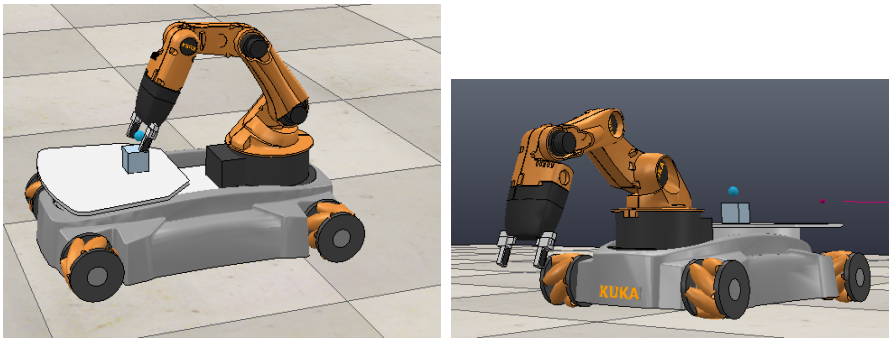


Figure 1: Kuka Youbot

2 Method and Design

2.1 State Machine

As stated in the introduction, our task can be broken up into the following steps: Locating the ball, driving to the ball, picking up the ball, locating the user, driving to the user, and dropping off the ball. The robot will need to keep track of its state, and do different things in each state. To do this, our top most control module is a state machine. A diagram of this state machine is given in the block diagram in Figure 2.

2.2 Ball tracking and Motion Profiles

To help simplify our design, we assumed the robot would have a rough estimate of where the ball actually is. In a more realistic simulation, this data would be obtained through

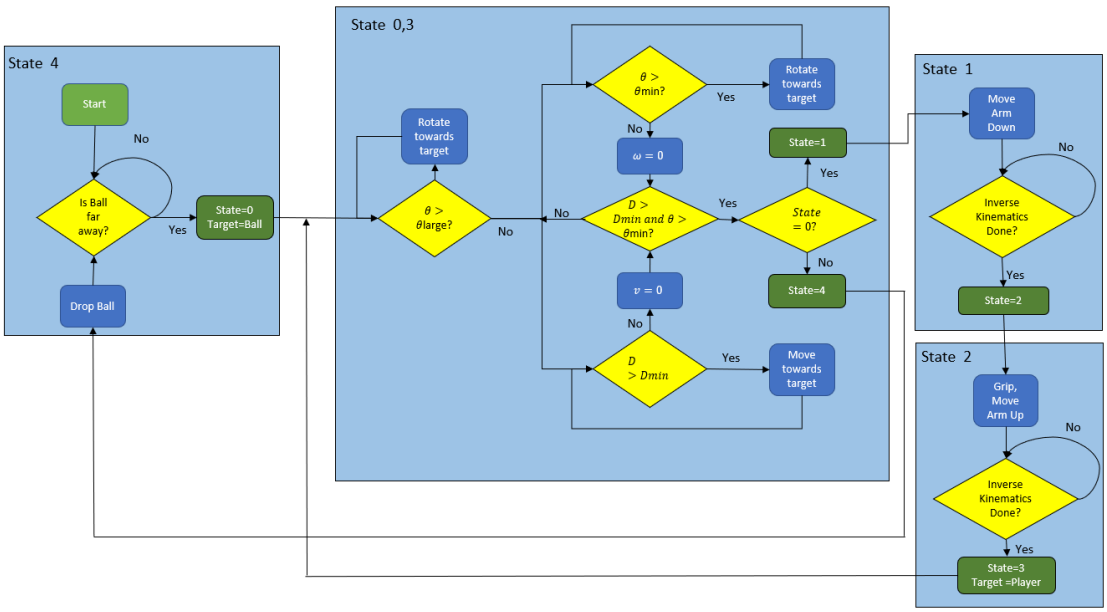


Figure 2: Robot Block Diagram

sensors such as a camera, however we obtained this data directly from the simulation. As the robot approached the ball, it uses an ultrasonic sensor to get the distance to the ball. We wanted to approach the ball quickly, but not crash into it. To do so, we used a 2 motion profiles, one for the robot's rotational velocity, and another for it's forward velocity. These are based on the robots linear and angular distance respectively. The robots final velocity is a combination of its rotational and forward velocity. These motion profiles are shown in below in Equations 2.1 and 2.2, where v and ω represent the linear and angular veolcities, and D and θ represent the linear and angular separation.

$$v = \left\{ \begin{array}{ll} 5D^2, & \text{for } 0.37 \leq D, \theta \leq 1.7 \\ 0, & \text{for } 1.7 < \theta \\ 0, & \text{for } D < 0.37 \end{array} \right\} \tag{2.1}$$

$$\omega = \left\{ \begin{array}{ll} 11.9, & \text{for } 1.7 \geq \theta \\ 7\theta, & \text{for } 0.01 \leq \theta < 1.7 \\ 0, & \text{for } \theta < 0.01 \end{array} \right\} \tag{2.2}$$

These equations show the 3 stages that the robot takes to travel to the ball. Stage one occurs when the robot has an angular separation of more than 1.7 radians, or about 97 degrees. In this stage, the robot rotates at a constant angular velocity, and stays in place. In the next stage, the robot both rotates and moves based on the error in distance and rotation. When either the rotation or the distance are within the error range, the robot enters the third state, where it only rotates or moves, but not both. The 3 stage motion profile for

the rotation was required, as we wanted to approach the ball after the robot finished the majority of its rotation. This profile can be seen below in Figure 3.

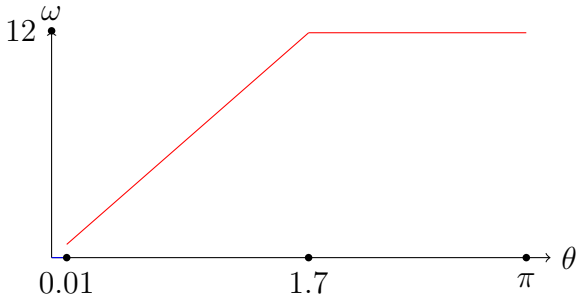


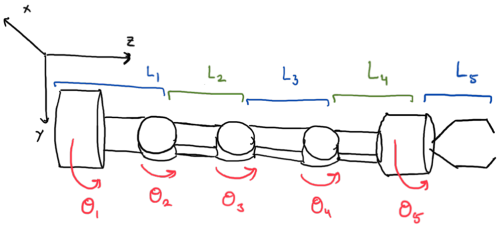
Figure 3: Rotational Motion Profile

These equations were based around the PID concepts learned from class. We originally planned to add I and D terms to the equation, however we were able to get good results by just picking different P constants for different errors, which was significantly simpler. The same method was used to move to the player.

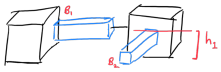
2.3 Forward and Inverse Kinematics

Now that we could drive to the ball, we needed to pick it up. To do this, we needed to control the robotic arm. Although not part of our final design, we calculated the forward kinematics for our robot. Figure 4 shows the simplified model of our robot.

(a) Simulation Zero State



(b) Model Arm Zero State



(c) Model Base Zero State

Figure 4: Zero States

By using the Forward Kinematics equations covered in class, we were able to come up with the transformation matrix shown in Equation 2.3.

$$T = \begin{bmatrix} -\sin(b_1)\sin(\theta_3)\cos(b_2+\theta_1+\theta_2)+\cos(b_1)\cos(\theta_3) & -\sin(b_1)\cos(\theta_3)\cos(b_2+\theta_1+\theta_2)-\sin(\theta_3)\cos(b_1) & \sin(b_1)\sin(b_2+\theta_1+\theta_2) & \sin(d1)+0.155\sin(b_1)\sin(b_2)+0.1348\sin(b_1)\sin(b_2+\theta_1)+0.1936\sin(b_1)\sin(b_2+\theta_1+\theta_2) \\ \sin(b_1)\cos(\theta_3)+\sin(\theta_3)\cos(b_1)\cos(b_2+\theta_1+\theta_2) & -\sin(b_1)\sin(\theta_3)+\cos(b_1)\cos(\theta_3)\cos(b_2+\theta_1+\theta_2) & -\sin(b_2+\theta_1+\theta_2)\cos(b_1) & \sin(d2)-0.155\sin(b_2)\cos(b_1)-0.1348\sin(b_2+\theta_1)\cos(b_1)-0.1936sin(b_2+\theta_1+\theta_2)\cos(b_1) \\ \sin(\theta_3)\sin(b_2+\theta_1+\theta_2) & \sin(b_2+\theta_1+\theta_2)\cos(\theta_3) & \cos(b_2+\theta_1+\theta_2) & 0.155\cos(b_2)+0.1348\cos(b_2+\theta_1)+0.1936\cos(b_2+\theta_1+\theta_2)+0.2454 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.3)$$

The forward kinematics was the first step shown in class to obtain the inverse kinematic solutions. However, we noticed Vrep had its own built in inverse kinematic solver. This solver works by creating Inverse Kinematics groups. This actually follows the method covered in class using the pseudo inverse of the Jacobian. We allowed for 6 iterations of the algorithm. This simplified our work, while still using the concepts covered in class.

3 Experiment

3.1 Setup

To see how our robot performs, we set up a series of tests. We wanted to find out how fast the robot was, and its success rate at picking up the ball. To do this, we placed the robot at a variety of locations in the world, and timed how long it takes to get to the ball, and if it picked up the ball. Since our robot is rotationally symmetric, we kept the starting angle relative to the world the same, so that various X,Y's would create different angles relative to the ball. An example of this setup is shown in Figure 5.

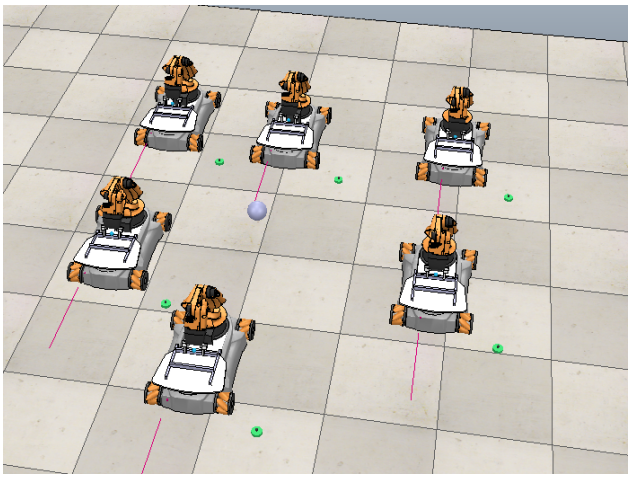


Figure 5: Experimental Seutp

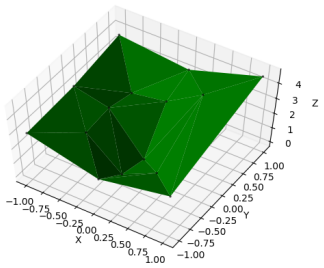
3.2 Results and Analysis

The raw data is shown in Table 1. We recorded the time twice for each location, and calculated the distance and the angle, and if the ball was picked up. We are unsure why there is a variation in the time.

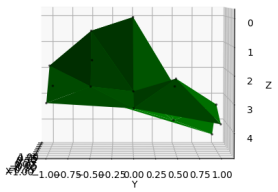
From Table 1, we see that the robot has a 100% success rate at picking up the ball. The rest of the data can be summarized with plots shown in Figure 6.

X	Y	Distance	θ	Time 1	Time 2	Picked Ball?
0.0	1.0	1.000	180.0	3.156250	3.281250	Yes
0.0	0.5	0.500	180.0	2.406250	2.687500	Yes
0.0	−1.0	1.000	0.0	1.656250	1.906250	Yes
0.0	−0.5	0.500	0.0	0.531250	0.468750	Yes
1.0	1.0	1.414	−135.0	4.656250	4.468750	Yes
−1.0	1.0	1.414	135.0	3.468750	3.968750	Yes
−1.0	−1.0	1.414	45.0	2.812500	3.125000	Yes
1.0	−1.0	1.414	−45.0	2.468750	2.906250	Yes
0.5	0.5	0.707	−135.0	3.625000	4.187500	Yes
−0.5	0.5	0.707	135.0	2.125000	2.250000	Yes
0.5	−0.5	0.707	−45.0	1.687500	1.593750	Yes
−0.5	−0.5	0.707	45.0	2.156250	2.781250	Yes
−1.0	0.0	1.000	90.0	3.156250	3.156250	Yes
−0.5	0.0	0.500	90.0	2.562500	2.718750	Yes

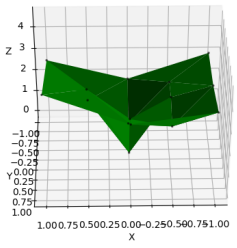
Table 1: Raw Data



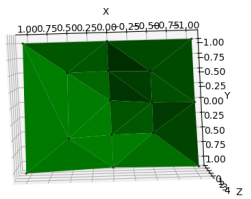
(a) Angled View



(b) Y-Z View (Inverted Z)



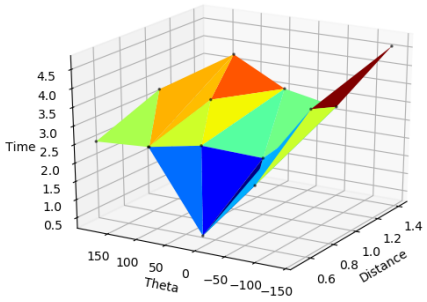
(c) X-Z view



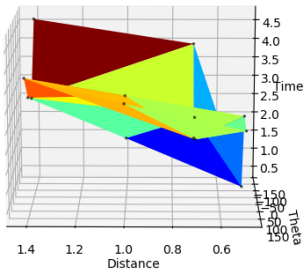
(d) Top Down

Figure 6: X,Y location of robot vs Time (Z axis) in seconds for robot to reach ball

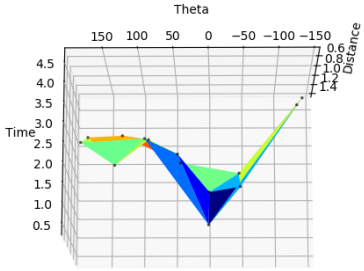
From Figure 6.b, we notice the notice the time is significantly greater at the positive Y values compared to the negative Y values. This is in contrast to Figure 6.c, where it is roughly symmetric for positive and negative X's. To better understand this data, Figure 7 shows the times in relation to the distance and angle.



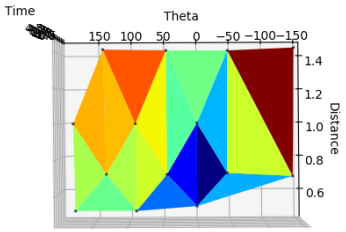
(a) Overall View



(b) Distance vs Time View



(c) Angle Vs Time view



(d) Top Down

Figure 7: Distance and angle of the robot relative to the ball VS time taken to reach ball

From Figure 7.c, we can see the effect θ has on the time, and that the time significantly increases as the robot is less aligned to the ball. As expected, Figure 7.b shows that the time is increasing the with the distance.

Overall, the robot took about 3.1297 seconds for each meter, and 1.304 seconds per radian.

4 Conclusion

Our project was a robot that was successfully able to drive to a ball, pick it up, and return it to a player. Our robot was built in the robotics simulator "V-Rep". Our robot used wheels and a robotic arm to move and interact with the environment. It had a 100% pickup rate, and moved fairly quickly to the targets. While doing this project, we learned a lot about the simulator its self, and the programming language LUA. We also learned a lot about forward and inverse kinematics, as well as motion profiles. If we had more time, we would like to add some of the searching algorithms learned in class to avoid obstacles. We would also like to use a camera to implement vision, like how we learned in Lab.

5 References

[1] V-Rep robotics simulator. `http://coppeliarobotics.com/`

[2] Kuka Youbot `www.kuka-youbot.com`

[3] Bubble Robot tutorial
`http://www.coppeliarobotics.com/helpFiles/en/bubbleRobTutorial.htm`

[4] Inverse Kinematics Tutorial
`http://www.coppeliarobotics.com/helpFiles/en/inverseKinematicsTutorial.htm`

[5] V-Rep API Documentation
`http://www.coppeliarobotics.com/helpFiles/en/apiFunctionListAlphabetical.htm`

[6] Lua Documentation `https://www.lua.org/manual/5.3/`