Hexapod Robot: Follow Line Trajectory using Robot Operating System (ROS)

[1]Bryan Pedraza
Department of Electrical and Computer Engineering
The University of Texas at San Antonio
San Antonio, TX 78249
Email: bryanjp53@gmail.com

***Abstract*—This project aims to present and demonstrate the process of constructing a two degree-of-freedom (DOF) hexapod robot capable of following any trajectory line using OpenCV and transitioning between different gaits. This paper will also showcase how gait transitions are executed whenever the robot needs to change gaits depending on the type of turn. The ultimate goal of this project is to use ROS Melodic for the hexapod robot to perform autonomous tasks. This paper will discuss the type of state equations used (as shown in Andrew Gibiansky's paper), kinematics equations used for each leg, rotation/transformation matrices, forces, and more throughout this report in different sections.**

***Keywords*—Robotics, ROS, OpenCV, Simulation, MATLAB, Gait, Hough Line Transformation**

I. Introduction

The overall design of the robot was intended to resemble Figure 1 and involve the process of mounting the hexapod robot. It took approximately 10-14 hours to build the robot up until this point. It could have been shorter, but there were difficulties encountered with the legs, and experienced a lot of hand cramps. Initially, six legs were assembled, but it was later discovered that the six right legs were not in their proper place. Hence, it took more time to disassemble three legs and then reassemble them in reverse order, which took a lot of time and effort, especially since the proper tools were not available. Nonetheless, it looked good at this point.

Afterward, the twelve servos mounted on the robot were tested, but unfortunately, two of the twelve servos did not function appropriately. These could be defective servos, and hopefully, other servos can be obtained to replace these malfunctions. But for now, it is not something to worry about. Getting everything installed is important first, and then later, the minor issues can be easily fixed by replacing the non-functioning servos with a working one. Mounting the Pi camera and the Adeept Robot Hot was also challenging, as putting it all together caused the ribbon to bend a lot. Hopefully, the ribbon did not get cut or damaged because if it did, there would be some problems. However, the issue can be easily resolved by replacing the ribbon with a working one.

Upon returning to the robot, it was discovered that the legs were assembled in the wrong way. This meant that the front legs were facing the back of the robot, and the back of the legs were in the front. However, it did not take too long to

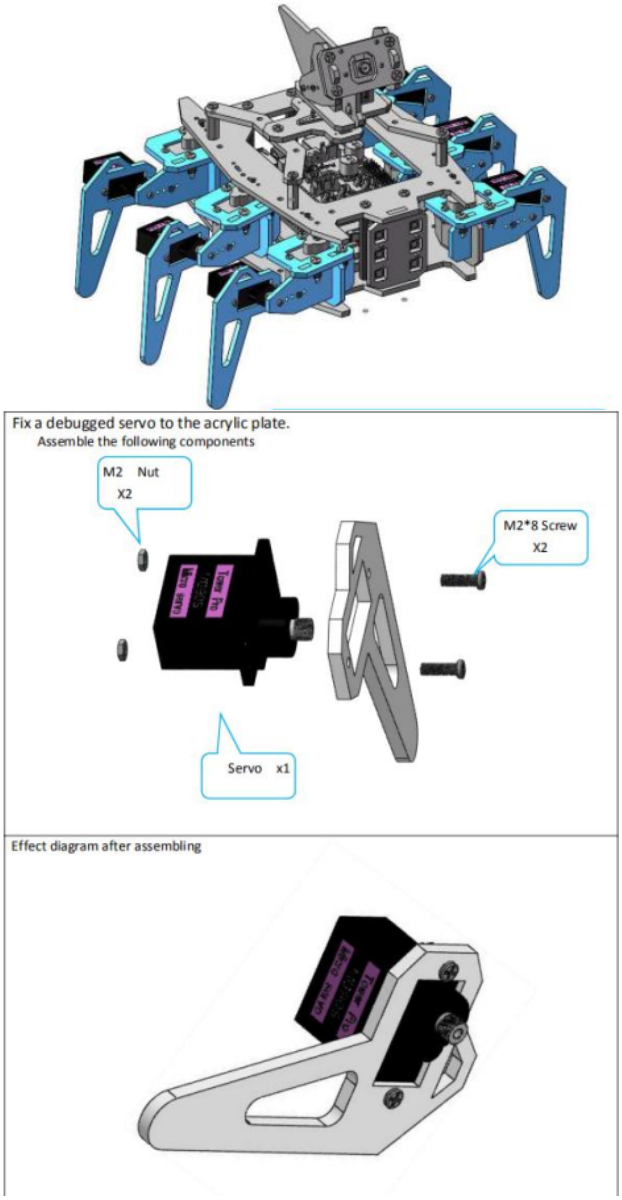

Fig. 1: The idle look of the robot (Adeept PDF Tutorial Instruction)

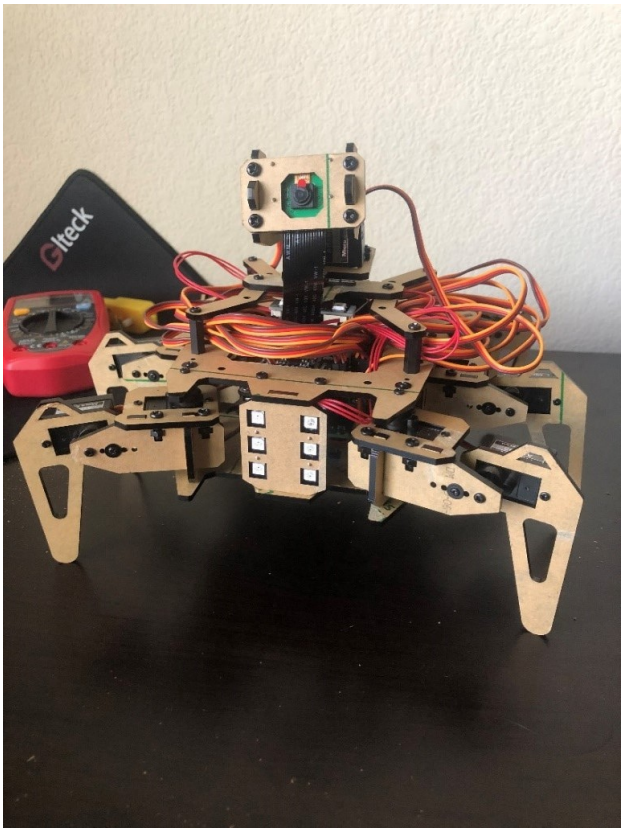

Fig. 2: First Assembled Hexapod Robot

fix the problem, as it only took about 35 minutes in total to detach and reattach the legs to the proper positions. The image in Figure 2 depicts the look of the hexapod robot up until that point in time.

II. Process

The robot has now been mounted correctly, and the wires have been plugged into the right pins. However, while disassembling and tinkering with the robot, I noticed a brown cardboard piece sticking out on the edge of some parts. I soon realized that the cardboard could be taken out of each part. Unfortunately, I didn't know this until now, but there's nothing to worry about since everything is already mounted, plugged, and ready to go. So why should I worry about the looks when I can continue to research, learn, and simulate the robot to understand the mathematics and then apply that to my physical robot in the real world? That's what's most important, rather than making the robot look pretty, although that is a nice bonus.

Figure 3 depicts a robot created in MATLAB, which has six legs and shares the same degree-of- freedom (DOF) as the RaspClaw robot (2 DOF). It is currently being prepared for simulations and testing, and once completed, it can be applied to the robot. The robot's leg length and body size have been adjusted, but two more joints are required to be added to the middle of the body for the camera joints. However, there is not

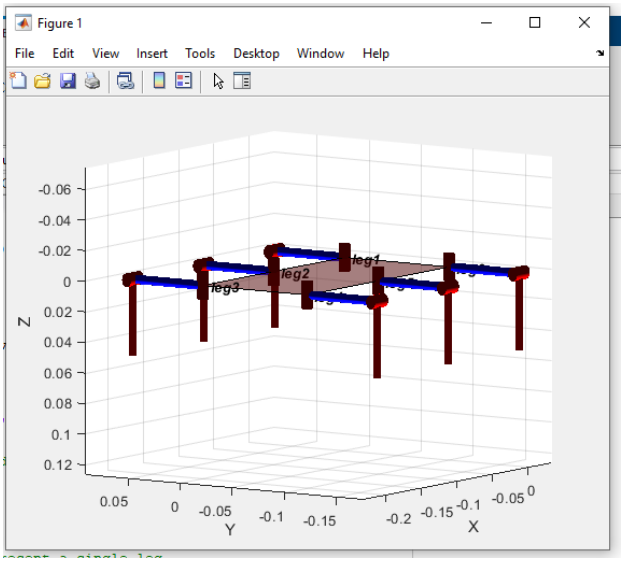

Fig. 3: Simulation Robot with correct measurements to the physical hexapod robot

much to worry about as the primary focus of this simulation is to get the math in rotation for the legs of the robot. Getting all the legs to work together will be the most challenging part. Additionally, the computer vision part will be discussed later in this paper under ROS.

III. Mathematical (Research)

Although a lot of work and time was invested in building the physical robot and the code simulation, a significant amount of time was also dedicated to researching the robot. While there are many reports on crawl robots available on the internet, many of them differ from this one, with only the number of joints being the distinguishing factor. However, the concept remains the same, and there is plenty of useful information that can be used in this project. Adjustments will be made to fit our robot's parameters, joints, size, and other specifications.

From these reports, the focus is on the way they apply velocity and angular velocity, joint positioning, the use of end effector equations on the legs, forces on each joint, kinematics equations, rotation matrix, the use of Jacobian on the angles, torques for each joint, and more importantly, how to apply all of these to the physical robot.

A. Transformation Matrix

A transformation matrix was modeled using D-H (Denavit-Hartenberg) parameters to determine how each leg would move. However, it's not until these matrices are multiplied together that the final transformation matrix is obtained. This final matrix holds the x, y, and z end coordinates in the last column, which can be used to position the robot as it follows the rotation on each leg. The final transformation provides the end coordinates, which can then be used to construct the rectangle segment path for the 2 degree-of-freedom robot. The

0.0500	0.0900	0.1000
-0.0500	0.0900	0.1000
-0.0500	0.1100	0.0500
0.0500	0.1100	0.0500
0.0500	0.0900	0.1000
-0.0500	0.0900	0.1000
-0.0500	0.1100	0.0500
0.0500	0.1100	0.0500

Fig. 4: Final Transformation on the Leg's x,y,z coordinates

```
qcycle = leg.ikine( SE3(xcycle), 'mask', [0 1 1 0 0 0] );
```

Fig. 5: Inverse Kinematics used to find the angles of the legs by just end-coordinates

x, y, and z cycle segments calculated for the 2 DOF robot are shown in Figure 4.

You can see that this will be the path of rectangle rotation that our robot will follow for each leg. By calling the "mstraj" command, we will get the position coordinates for each leg. We can then obtain the angle/servo coordinates (q1 and q2) by simply performing the inverse kinematics of the end coordinates found from the mstraj command. Figure 5 below shows the inverse kinematics used to find the angle positions on the x, y, and z coordinates, as well as the mask used to compute the inverse for the 2 DOF robot.

There are, of course, more rows than shown in Figure 6: Rotation Matrix Angles. However, it is possible to see how the two angle coordinates are changing (because x, y, and z coordinates are changing as well). Hence, inverse kinematics works effectively when applied to find the angles.

With the transformation matrix, we can find our end coordinates and use them to form the rotation matrix that each leg will cycle through. Finally, we can obtain the angle/servo coordinates by using inverse kinematics of the x, y, and z coordinates. This process was utilized for this robot project.

B. Gait

Gaits are formed by a formula and a chosen sequence for the rotation along a path, determining how the robot will maneuver using all six legs. There are multiple types of gaits, but the ones that will be used are explained below, along with the idea behind each gait.

The gait described here is the most stable walk for the robot and will be used when the robot is walking straight or normally. Figure 7 demonstrates how each leg moves for this gait.

190	-0.5071	-0.0296
191	-0.5071	-0.0296
192	-0.5071	-0.0296
193	-0.5071	-0.0296
194	-0.5071	-0.0296
195	-0.5071	-0.0296
196	-0.5071	-0.0296
197	-0.5068	-0.0294
198	-0.5046	-0.0282
199	-0.4993	-0.0251
200	-0.4895	-0.0198
201	-0.4747	-0.0119
202	-0.4543	-0.0016
203	-0.4282	0.0107
204	-0.3968	0.0242
205	-0.3608	0.0380
206	-0.3218	0.0513
207	-0.2812	0.0631
208	-0.2397	0.0734
209	-0.1974	0.0820
210	-0.1543	0.0889
211	-0.1107	0.0942
212	-0.0666	0.0977

Fig. 6: Rotation Matrix Angles of the Two Angles (servos) using the Transform Matrix

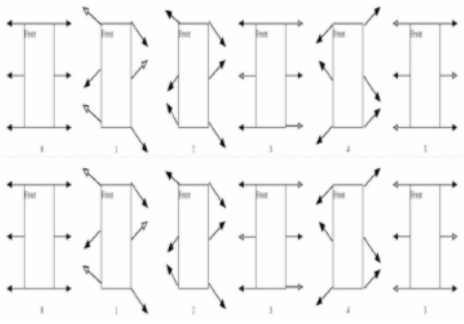

Fig. 7: General Idea: Tripod Gait Walk (Referenced Below)

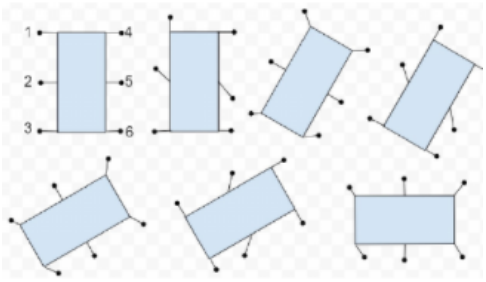

Fig. 8: General Idea: Swivel Gait walking towards the right (Referenced Below)

Algorithm 1 Gait Transition Method

```
function GETGAIT(gait, power, T_elapsed)
    if T_elapsed > T_long then
        if gait = Tripod then
            if |avgPower − power| > P_tripod then
                T_elapsed ← 0
                return wave
        if gait = Wave then
            if |avgPower − power| > P_wave then
                T_elapsed ← 0
                return tripod
    return gait
```

Fig. 9: General Idea: Gait Transition Algorithm (Referenced Below)

The Ripple Gait is a stable but slower walk where the idea is to move one leg and one side at a time. This gait is typically used when a robot needs to slow down due to upcoming turns or events.

The Swivel Gait, on the other hand, is used to make sharp turns by getting the robot to turn quickly and fluently, either to the right or left. Figure 9 demonstrates a right turn, and the same can be applied to a left turn in the opposite direction.

C. Gait Transition

The gait transition algorithm tests certain parameters, and if the condition is broken, then a certain gait will be selected, depending on the condition. Below is the general idea of how to incorporate and code a gait transaction to select the proper gait. It is also important to note that between gait transitions or switching between gaits, there must be a gate delay timer between the gaits. This is necessary to allow time for the gait to settle before switching to another gait, as shown in Figure 9.

IV. Simulation Graphs

A. Graphs

In Figure 10, there is a simulation of the rotation of one leg. Angle1 being the joint on the body, and angle2 being the joint on the leg.

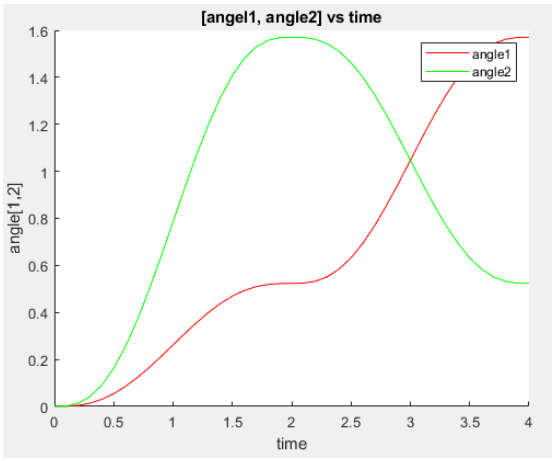

Fig. 10: Simulation Graph of a Leg on Hexapod Robot Simulation

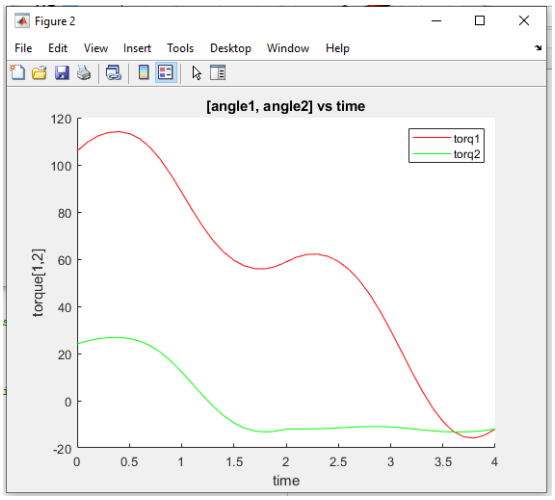

Fig. 11: 2nd Simulation Graph of a Leg on Hexapod Robot Simulation

Additionally, in Figure 11, another leg simulation was performed to move the leg to a different rotation position.

V. ROS

A. Nodes

In the ROS environment, four nodes are configured to assist the robot in walking and maneuvering along a line trajectory. The first node will act as the master and serve as the primary computer. The second node will be the robot's camera, which uses OpenCV computer vision and publishes to three topics: raw frame, line frame, and command. OpenCV enables computers to perceive the world and make decisions based on conditions and thresholds. The second node represents the robot's body and legs, where it moves straight or turns left or right depending on the curve. To transmit this information to the second node, a subscriber is implemented to the command topic where the first node is publishing. The publish is a String class, which is part of the std_msg library in ROS. The second

node also subscribes to the third node, which is the mpu6050 used to measure the 3-axis gyroscope of the robot. This publish uses a Float32 class, also part of the std_msg library. From there, the node acquires the axis data and publishes it to the axis topic, which the second node also subscribes to. As a result, the second node has all the information it needs to make the robot move and adjust depending on the conditions and decisions.

B. OpenCV

In ROS, the computer vision part detects lines using the Hough Line Transformation algorithm. However, before using the algorithm, there are some things to consider and do. Firstly, the raw frame must be captured, and this serves as the input image to detect what the robot needs. From there, the raw image is sent to the Canny algorithm to detect the edges of the line. The values in the parameters of the Canny algorithm are used to measure how easily or hard the robot can detect an edge. For example, the robot can detect a black line (permanent marker) but won't detect a pencil line because a threshold for the robot was applied to differentiate between these two edges. Lowering the threshold will result in detecting lighter markings like a pencil line, but since a pencil line is not ideal for this project, a permanent black marker was chosen to draw the lines with the right threshold.

C. Line Detection Algorithms

After finding the edges using Canny's algorithm, the same frame is passed through another algorithm called the Hough Line Transformation. This algorithm is primarily used for image analysis and takes a point in pixels and places it in line space (x,y space), which is then transformed into Hough space, consisting of polar coordinates (ρ, θ), representing length vs angle. The algorithm is then used to derive the position and angle of the line in camera pixels. Once the angle and position are detected, the probability is calculated, which is used to determine the probable start and end point of the line, resulting in accurate length and placement of the line in pixels for the robot's view. Figure 12 depicts the input raw frame and the output frame of what the robot sees. A lot of mathematical calculations are involved to determine the amount of curve the robot needs to make for the rotation of the legs. This process is done in the camera/detection node and then sent/published to the move/body node. The move/body node then knows how much to turn based on the current state of the body and will maneuver in the direction of the curve.

VI. Final Product

Figure 13 shows the track that was used for the hexapod robot. The red circles were particularly challenging to navigate due to the complex math involved, but the robot was able to successfully complete the track. However, the Raspberry Pi 3 hardware used in the robot has a low frame rate per second, which limits the speed at which the robot can move since it can't detect as fast as it walks. Therefore, the walking speed needs to be slowed down in order to get the next updated frame

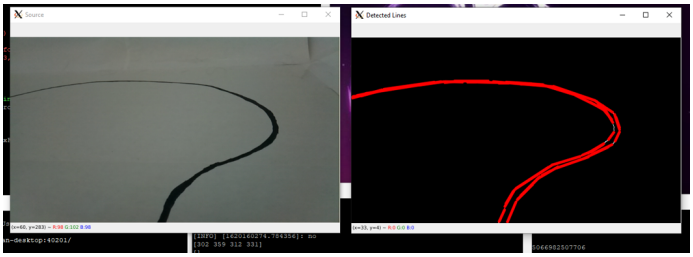

Fig. 12: Input Raw Frame and Output Frame of What Robot Views

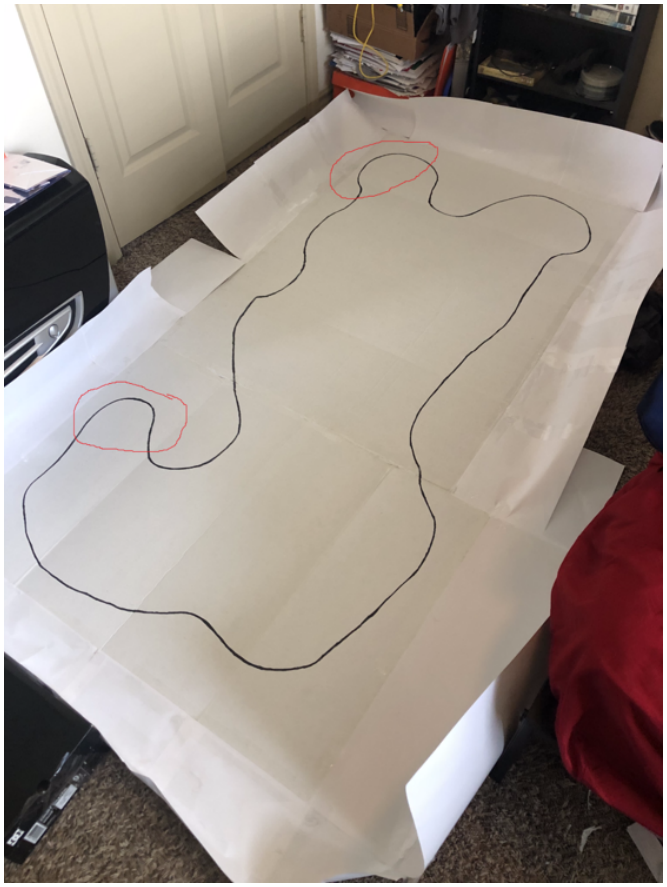

Fig. 13: Input Raw Frame and Output Frame of What Robot Views

and the next movement at that point. To accomplish this, the robot was programmed to wait until the next accurate frame was updated and then move accordingly based on the state it was in. This helped the robot to navigate each line and curve on the track successfully.

Figure 14 shows the robot up to date. The ribbon was replaced due to it got damaged during installation but after replacing it with a good ribbon the camera worked.

Figure 15 shows the physical hexapod robot moving and the final look of the robot.

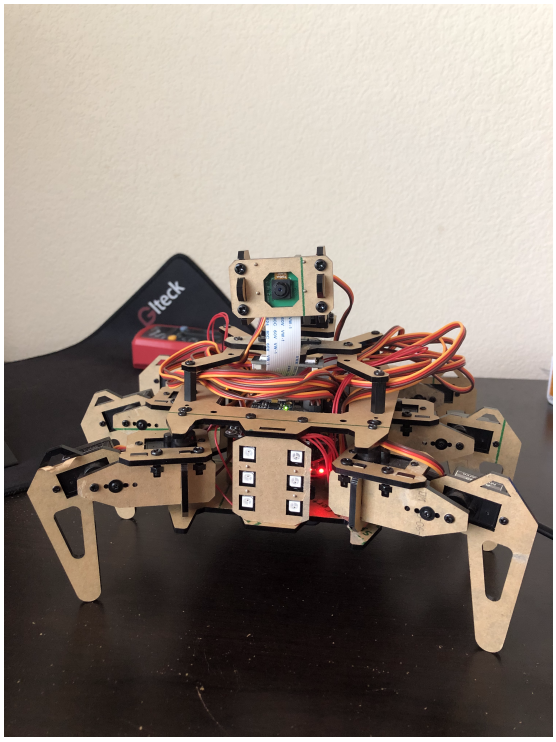

Fig. 14: Up to Date Hexapod Robot

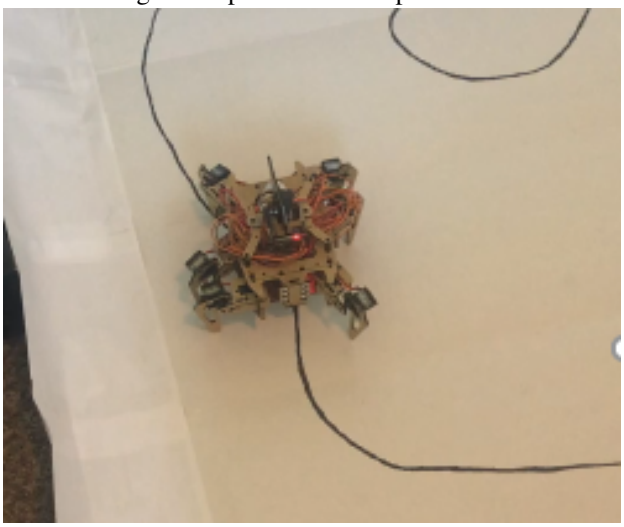

Fig. 15: Hexapod Robot Walking

VII. Conclusion

This paper introduces the concepts of obtaining the angle coordinates and transformation rotation matrices for the robot's legs using the Peter Corks toolbox. The simulation yielded accurate results in the rotation of the legs by employing inverse kinematics on the joint end-effector (x,y,z plane) of a robot to obtain servo angles. The paper also outlines the ideal way of maneuvering a hexapod robot, the math behind it, and the algorithm used for the project. The concept was then applied to the physical robot on how it should move and was subsequently taken over to ROS. The paper also explains the concept of computer vision (OpenCV) using candy and the Hough Line Transformation algorithm. With that, the math of the curve was derived for the robot to calculate so that it could have an accurate assumption on how big of a turn to do (rotation of the legs matrices) and when to do the turn. Although some curves were a bit rough and took some time for the hexapod to move and curve around, the hexapod experiment was a huge success after tweaking the math values. All in all, the end of the project was a significant stepping stone in learning more about robotics and ROS.

VIII. Future Work

The calculations involved in determining the curve and rotation for the robot still require some adjustments and tweaks to ensure precise turning along the curve. Even though the robot manages to traverse the path, it needs to incorporate advanced concepts to gauge the proximity of the path ahead. However, the robot's camera has a 2-axis frame and cannot accurately measure depth or distance. To improve accuracy and movement, the robot would benefit from the addition of ultrasonic sensors or LiDAR, which will provide a better view of the world and allow the robot to determine the distance and make more accurate turns. Another potential improvement is to upgrade the hardware to something more powerful, such as a Raspberry Pi 4 model B, to address the issue of low frame rates and latency in the camera view. This will help the robot detect lines more quickly and maneuver more efficiently due to the higher frames per second. To also further extend this project, it is necessary to include machine learning (ML) on the robot. For instance, the simulated robot could be trained under simulation over several episodes until it reaches optimal convergence. Then, it could be tested under simulation and validated with a pass test result before transferring its knowledge (weights and bias of the network) onto the actual physical robot to examine how well it would do in the real-world environment. This may be a future work project that could be done. Incorporating these ideas in the future could potentially enhance the hexapod robot's functionality and performance.

IX. References

[1] Adeept. "RaspClaws Spider Robot Kit for RPi - Adeept Learn." www.adeept.com/learn/detail-37.html.

[2] El Hansali, Hasnaa Mohammed, Bennani. (2017). "Gait Kinematic Modeling of a Hexapod Robot." International Review of Mechanical Engineering (IREME). 11. 200. 10.15866/ireme.v11i3.10987.

[3] Soltero, Daniel Julian, Brian Onal, C.D. Rus, Daniela. (2013). "A Lightweight Modular 12-DOF Print-and-Fold Hexapod." Proceedings of the ... IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE/RSJ International Conference on Intelligent Robots and Systems. 10.1109/IROS.2013.6696542.

[4] Kottege, Navinda Parkinson, Callum Moghadam, Peyman Elfes, Alberto Singh, Surya. (2015). "Energetics-Informed Hexapod Gait Transitions Across Terrains." Pro-

ceedings - IEEE International Conference on Robotics and Automation. 2015. 10.1109/ICRA.2015.7139915.

[5] Sayed, Abdelrahman. (2020). “Design and Control of an 18 DOF Hexapod Multi-agent Swarm for Search and Rescue Missions.” 10.13140/RG.2.2.34176.35845.

[6] Menezes, Jovan Das, Shubhankar Panchal, Bhavik Yelve, Nitesh Kumar, Praseed. (2021). “Mapping, Trajectory Planning, and Navigation for Hexapod Robots using ROS.”

[7] Wang, Ruyi. (2020). “Hybrid Gait Planning of A Hexapod Robot. Modern Electronic Technology.” 4. 11. 10.26549/met.v4i2.5075.