NUST COLLEGE OF
ELECTRICAL & MECHANICAL
ENGINEERING

DESIGN AND FABRICATION OF A FOUR-LEGGED ROBOT

PROJECT REPORT

DE-41 (DME)

Submitted by

ALI ASIF ABDUL MUTAAL

MUHAMMAD INAM-UL-HAQ

BACHELORS

IN

MECHANICAL ENGINEERING

YEAR

2023

PROJECT SUPERVISOR

DR. RAJA AMER AZIM

NUST COLLEGE OF
ELECTRICAL AND MECHANICAL ENGINEERING
PESHAWAR ROAD, RAWALPINDI

DECLARATION

We hereby declare that no portion of the work referred to in this Project Thesis has been submitted in support of an application for another degree or qualification of any other university or other institute of learning. If any act of plagiarism is found, we are fully responsible for every disciplinary action taken against us depending upon the seriousness of the proven offence, even the cancellation of our degree.

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

We express our heartfelt gratitude to Allah Almighty, the most gracious and the most merciful, for granting us the strength, perseverance, and wisdom to complete this project successfully.

We would also like to extend our sincere appreciation to our esteemed supervisor, Dr. Raja Amer Azim, for providing us with valuable guidance, encouragement, and feedback throughout the project's development.
Our gratitude also goes out to all the instructors who contributed their time, knowledge, and expertise to assist us in achieving our goals.

Finally, we would like to acknowledge our families and friends for their unwavering support, understanding, and patience during the project's execution. We are indebted to all those who have played a significant role in helping us bring this project to fruition.

ABSTRACT

This report presents the design and implementation of a four-legged robot that can perform various locomotion tasks. The robot hardware consists of four legs, each with three degrees of freedom, and a central body that houses the control of electronics and power supply.

The legs are actuated using servo motors and are equipped with sensors for measuring joint angles and ground contact. The control system is based on an embedded microcontroller, which runs custom software for coordinating the leg movements and maintaining balance. The robot is designed to be versatile and robust, with the ability to traverse on uneven surfaces. The proposed design offers a promising platform for exploring the capabilities of legged robots and advancing the field of robotics.

The report discusses the electronic components used in a four-legged robot, including the servos, Raspberry Pi 4b, Arduino Uno, PCA9685 PWM servo driver, MPU 6050, and battery. Each leg consists of three servos: one for the hip, one for the upper part of the leg, and one for the lower part of the leg. The servos are attached to the PCA9685 PWM servo driver, which is controlled by the Arduino Uno.

The Raspberry Pi acts as the brain of the robot and is responsible for the overall geometry of the robot. The report also discusses the use of ROS (Robot Operating System) to control robots and how it is connected to the hardware components.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

v	**Velocity**
τ	**Torque**
ρ	**Density**
m	**Mass**

CHAPTER - 1 INTRODUCTION

1.1 Background:

Legged robots move about by using limbs that can be bent, resembling the mechanics of natural legs, exhibit a remarkable capacity to traverse uneven surfaces, navigate challenging obstacles. They can adapt to various terrains and are more adaptable than wheeled robots. They often copy legged animals. Robots with four legs, or quadrupeds, are those that move like four-legged creatures like dogs, cats, or horses. In the mid-twentieth century, pioneering researchers embarked on a quest to harness the potential of legged robots for traversing rugged terrains, which is when these robots first began to take shape. The first versions were big and heavy, and it wasn't until the 1980s that more sophisticated models started to appear [1].

The complexity and advancements of four-legged machines have increased due to advances in robotics technology. They have potential uses in a variety of industries, including the military, agriculture, and rescue and search operations. These robots can move through environments that are challenging for wheeled or tracked robots, ascend steps, and traverse rough terrain.

Research in this field has led to the development of several notable robots, such as the Big Dog, developed by Boston Dynamics, which can carry heavy loads across rough terrain, and the Cheetah, which can run at high speeds. The advancements in control algorithms, sensing technologies, and materials science have enabled the development of more agile and versatile four-legged robots, opening new possibilities for their use in a variety of applications. Modern four-legged robots have sophisticated sensing capabilities, including cameras, LiDAR, and other sensors, which allow them to sense their surroundings clearly. They also employ intelligent control systems that improve movement efficiency and enable them to adjust to various terrain conditions.

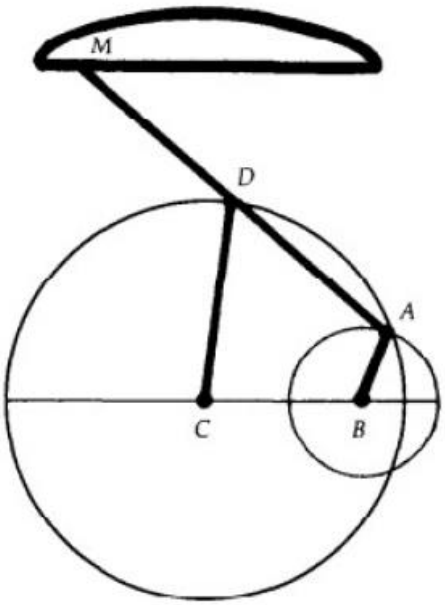
The building components for these robots have also evolved, making it possible to create designs that are lighter and more robust.

1.2 Motivation:

The main reasons for inventing the legs for moving include the following:

1.2.1 Mobility:

Vehicles that can traverse challenging terrain are required, as certain vehicles cannot go to the places that have uneven terrains. Tires perform best in rails and hard surfaces, but when there is a need to travel on soft surfaces, they perform poorly. Because of this, half of earth land is accessible to existing wheeled vehicles. Legged robots can provide a more range of mobility when it comes to walking on uneven terrain.



$$CD - AD - DM = \frac{3+\sqrt{7}}{2}$$

$$BC = \frac{4+\sqrt{7}}{3}$$

Figure 1-1 Mechanism used in Early Walking Machines

As shown in Figure 1-1 when the input crank AB rotates, the output point M moves along a straight path during one part of the cycle and an arched path during the other part. Two identical linkages are arranged to operate out of phase so at least one always provides a straight motion. The body is always supported by feet connected to the straight moving linkage; linkages of this sort are simple means of generating patterned motion.

One reason for legged robots to provide better mobility is that they can adjust their foots according to the terrain to optimise support and traction [2]. With the recent technological

advancement, as in computer vision and machine learning, the legged robot can better optimise the path and bear more harsh conditions.

1.2.2 Understanding Animal Locomotion:

We are still at a basic stage in terms of understanding the control principles of walking and running, despite the abilities we utilise for moving our legs. Building a legged robot is one method of learning about more acceptable methods to comprehend animal behaviours. [2] An initiative for search and rescue robots starts in early 90s after earthquake in Kobe, Japan. Recent incidents like disaster in Fukushima Nuclear plant in Japan emphasised the need for robotic solution that can be used to assist human search and rescue operations in non-human friendly environment [3].

1.3 Aims and Objectives:

1. Developing low-cost product having maximum accuracy.
2. Design of legs and links on SOLIDWORKS.
3. Fabrication of the assembly.
4. Searching for a way to get components at a low cost with maximum efficiency.

1.4 Future Scope:

Development of Reinforcement learning models so that the robot can learn new tasks in real world-environment without the need to run simulation and training it beforehand.

- Research on hybrid systems that combines the power of leg and wheel to make it more robust to tasks.
- Research on autonomous driving, artificial intelligence and emotion expression has opened new areas for research.

1.5 SDGs:

Following SDGs are targeted.

GOAL 9: Industry, Innovation, and Infrastructure

Enhance scientific research, modernize the industrial sectors' technological expertise in all nations, especially developing nations, and, by 2030, encourage innovation by increasing both public and private spending on research and development as well as the ratio of workers engaged in it per million people.

GOAL 12: Responsible Consumption and Production

Encourage developing nations to expand their scientific and technical capabilities so they are able to more resilient patterns of production and consumption.

1.6 End-goals:

- Less Cost
- Fabrication
- Achieving an effective gait
- Stability

1.7 Research work layout:

Overall research work is divided into several parts and chapters that can be seen in the table of contents. Different chapters contain different aspects related to the study of the project. Chapter 2 contains the Literature Review that comprises of research in the relevant area.

Chapter 3 contains the information regarding the hardware involved in this project. It includes the electronics, different sensors that are used for this project, and the solid model that was made using SOLIDWORKS.

Chapter 4 contains the software aspects of this project. It includes information regarding the microcontroller used and the programming developed. Programming related to different actuators and components is described in this chapter.

Chapter 5 basically focuses on the testing phase of the product. It also contains an instruction manual for the person who will acquire this product, the instructions will clearly mention how to operate this product and troubleshooting in case of any problem.

Chapter 6 comprises of the results and the conclusions drawn. It also depicts the efforts and the hurdles which were there during the making of this project. This chapter also contains future scope and recommendations linked to the project.

CHAPTER - 2 LITERATURE REVIEW

2.1 Introduction:

This chapter involves a review of the literature conducted on the four-legged robots, their working, locomotion and the study of the individual links and joints. The chapter includes overall research and the study related to these types of robots.

2.2 Historical Work:

The history of legged robots is shown by figure:

- To study the machines that walk, an early model was devised in 1870. The feet moved up and down to provide support when stepping, and a linkage was utilised to move the entire body in a straight line as in Figure 2-1.

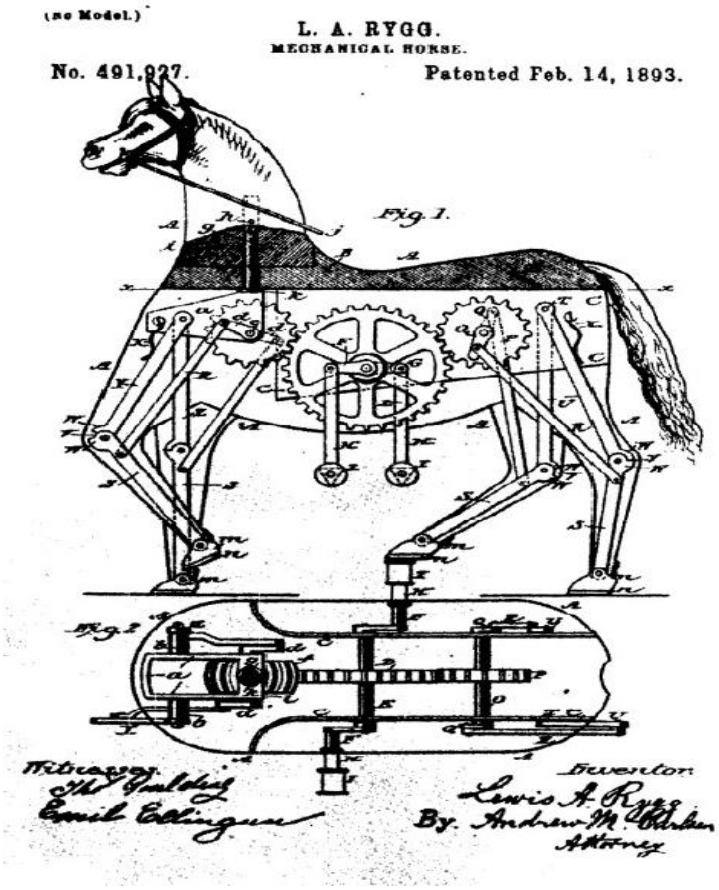


Figure 2-1 Early Model Devised in 1870

Building walking machines was considered in the 1980s and 1990s as a problem of creating connections that would provide appropriate stepping movements when powered. Since these robots could not adapt to the challenging terrain, it became obvious that control was required, and that fixed motion would not be sufficient [2].

Utilising a person was an alternative strategy. This method was utilised by Mosher at General Electric in the middle of the 1960s to create a walking truck with four legs. There are videos of the machine climbing a stack of railway ties, lifting a vehicle out of mess, and allowing a drum to fit into some hooks [2].

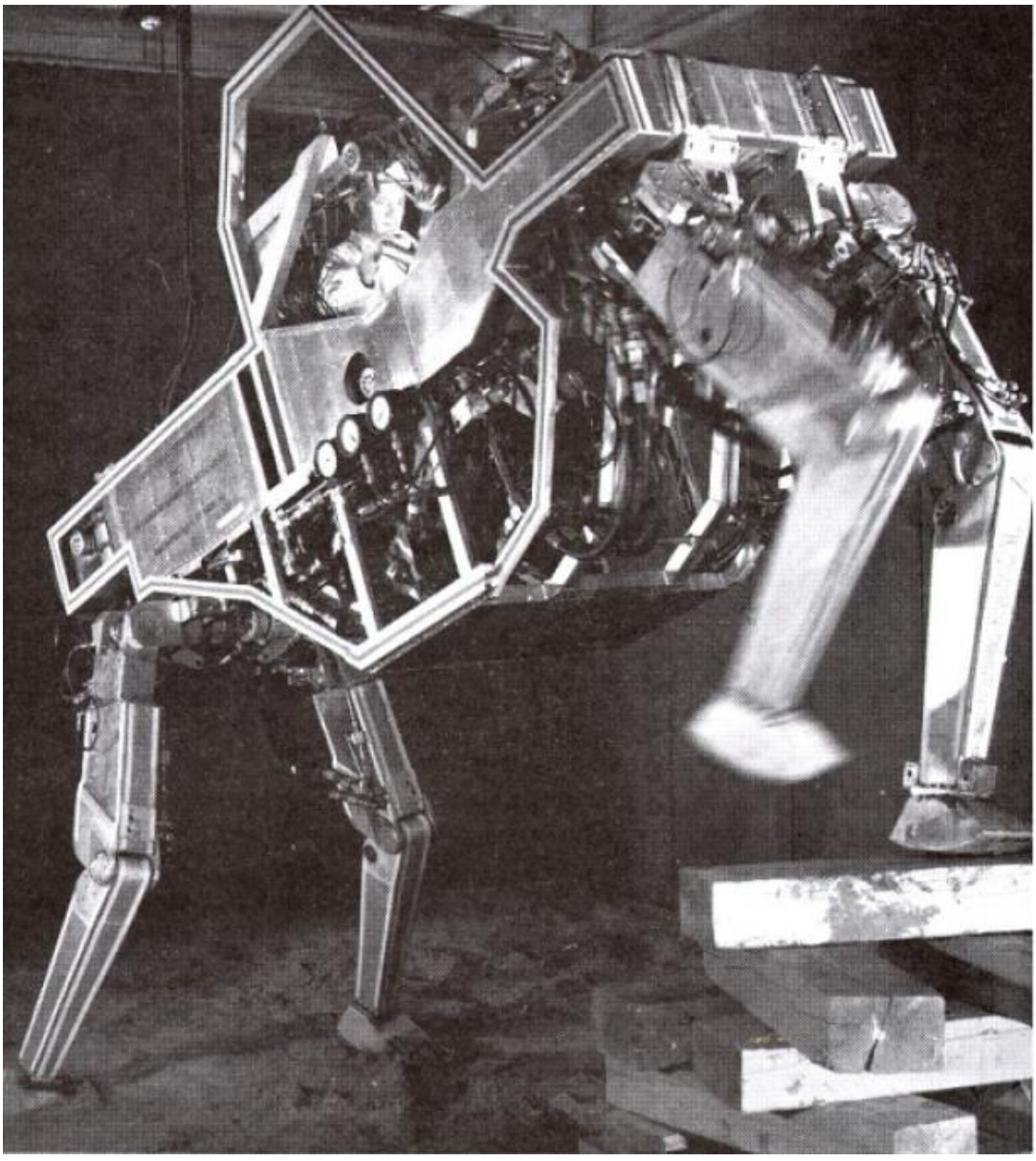


Figure 2-2 Walking Truck

A hexapod that could walk, turn, walk sideways, and pass through small obstacles was created by the McGhee group at Ohio State University in the 1970s. The main responsibility of the computer was to coordinate the 18 motors by solving kinematic equations. This allowed the legs to move while ensuring that the machine's centre of mass remained above the support given by the feet [2].

2.3 Legged Robot Types:

Robots with legs can be divided into several categories based on how many limbs they employ. The stability improves with the number of legs.
The following eight types of robots often have legs:

- One-legged
- Two legged
- Three-legged
- Four-legged
- Five-legged
- Six-legged
- Eight-legged
- Hybrid

2.3.1 One-leg Robots:

Robots with only one leg navigate via hopping. The hopping robot on one leg is great for simulating quick movement, like that of a running animal. Despite having a straightforward leg mechanism, the dynamics are complicated and demand non-linear complex analysis [4].
Roboticists at UC Berkeley have built a small robot that jumps into the air or perform vertical jumps. Salto was inspired by the most vertical agile creature, the galago Figure 2-3, which can jump about 5 times in just four seconds to gain 8.5 meters in height [4].

Figure 2-3 One- Legged Robot

2.3.2 Two-legged Robots:

These robots walk on two legs. They struggle with two main issues. Motion control and stability control. For bipedal systems, stability is challenging since they have to keep their equilibrium even while at rest.

Having broad feet can help with this problem since they increase stability while reducing movement. More sophisticated systems include sensors like accelerometers or gyro metres, which balance the body using a closed feedback loop, to boost stability. Nowadays, machine learning is used to offer a more effective route. A rolling polygon with sides that are equal in length to one step can simulate basic bipedal action. The number of sides increasing, and the motion gets closer to that of a circle as the step length gets shorter.

2.3.3 Boston- Dynamics Atlas

Boston Dynamics created the bipedal humanoid robot, Atlas. Originally, this was intended for a range of search and rescue duties. It has two vision systems, a stereo camera, and a laser rangefinder. There are a total of 28 degrees of freedom in its limbs. It can climb and move across uneven terrain utilising its arms and legs. It is capable of backflips and box jumps [5].

Figure 2-4 DARPA Robot

The following are the eight challenges that Atlas was able to perform in the 2015 DARPA robotics competition:

1. Achieve traversal through rubble while on foot.
2. Travel the area in a utility vehicle.
3. Clear any obstructions from an entrance.
4. Open a door and walk into a structure.
5. Climb a commercial ladder and stroll across a commercial walkway.
6. Make a hole in a concrete panel using a tool.
7. Find and shut a valve close to a leaking pipe.
8. Attach the fire hose to the standpipe and open the valve [5].

2.3.4 Four-legged Robot:

Robot with four legs moving in quadrupedal fashion. Compared to two-legged robots, they are more stable. When moving down a path, they maintain stability by establishing a triangle between the three legs. This guarantees that CG is contained within the triangle. Another advantage of four-legged robots over two-legged robots is that they have a lower centre of gravity [5].

Figure 2-5 Four- Legged Robot

Three degrees of freedom are available for each leg, allowing for hip abduction and adduction, hip flexion and extension, and knee flexion and extension. The robot's leg is moved by a four-bar system that is turned by a crank and a rope. It takes four links and a revolute joint to create this mechanism.

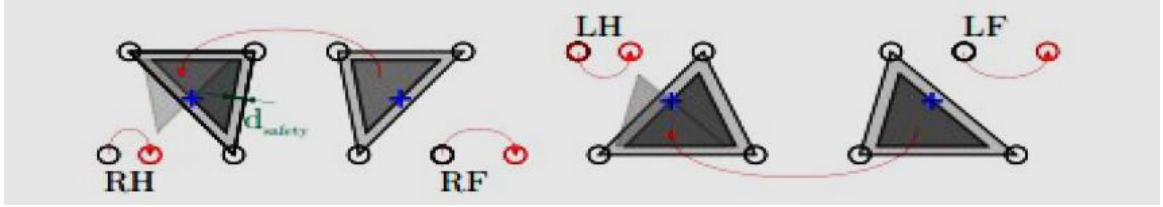


Figure 2-6 C.O.G trajectory generation

Three degrees of freedom are included in this kinematic mechanism, two of which are used to spin the leg and one of which is related to the rotation of the crank that is attached to the frame and has an extra revolute joint connecting to the cord that is used to initiate the movement of the four-bar mechanism. One example of such a robot is Mini Cheetah by MIT [6].

2.3.5 Mini Cheetah

Mini Cheetah is a compact, affordable, and mechanically durable quadruped robot designed to facilitate the quick development of control systems for legged robots [6].

Figure 2-7 Mini Cheetah

2.3.6 MIT Super Mini Cheetah:

The robot is exceptional for its size and price since it can precisely manage the force and impedance between each foot and the ground as well as create enough force and motion for a variety of dynamic behaviours [7].

Figure 2-8 MIT Super Mini Cheetah

2.3.7 Six-legged Robots:

Six-legged or hexapods have greater stability than four legged. It often mimics the mechanics of insects. Examples contain:

- In the 1980s, ODETICS created the 375-pound hexapod ODEX.
- Rodney Brooks at MIT created Genghis, one of the first autonomous six-legged robots, in the same decade [7].

It demonstrated how complex motion can emerge from a network on motors. Due to this feat, it was proposed that exploration of the solar system should be based on cheap, fast missions using mass-produced autonomous robots rather than more complex and costly spacecraft.

Figure 2-9 Six- Legged Robot

2.3.8 Eight- legged Robots:

Eight-legged legged robots are inspired by spiders, and some underwater walkers. They are superior in stability as compared to other bots.

Figure 2-10 Eight-Legged Robot

Dante is a tethered walking robot with the capacity to ascend challenging inclines. It was developed by Carnegie Mellon University and sent to Antarctica in 1992 to investigate Mount Erebus, an active volcano [8].

2.3.9 Hybrid:

Some robots employ the use of both legs and wheels. ETH Zurich Animal bot is a prime example. It introduced extra degree of freedom by actuated wheels. It combines the rough terrain capability with wheels. Fast, efficient, and versatile robot over long distances and in rough terrain. Hybrid and dynamic motion in flat, inclined, and rough terrain [9].

Figure 2-11 ETH Zurich ANYmal Robot

2.4 Advancements in the field of Robotics:

There have been many recent advancements in four-legged robots, also known as quadruped robots, particularly in the areas of locomotion and control.

One of the most significant advancements has been the development of advanced control algorithms that allow quadruped robots to walk, run, climb, and even perform acrobatic manoeuvres with incredible agility and speed. These algorithms use advanced sensing and feedback systems to constantly adjust the robot's movements in real-time, allowing it to maintain balance and stability even on difficult terrain [10].

Another important advancement has been the use of lightweight, high-strength materials and components, such as carbon fibre and titanium, which allow quadruped robots to move quickly and efficiently while also withstanding extreme stresses and strains.
In addition, researchers have also been working on developing new forms of locomotion for quadruped robots, such as the ability to crawl, roll, and even swim. These new forms of locomotion could allow quadruped robots to operate in a wider range of environments and perform a greater variety of tasks [11].

Overall, these advancements in quadruped robot technology are paving the way for a new generation of robots that can perform complex tasks and operating in challenging environments, from search and rescue missions to planetary exploration.

2.4.1 DARPA Robotics:

The DARPA Robotics Challenge was a competition organized by the United States (DARPA) to promote the development of advanced humanoid robots that can assist in disaster relief and other emergency situations. The competition was held from 2012 to 2015 and featured a series of challenges designed to test the robot's capabilities.

Robots that were used in DARPA Robotics Challenge were some of the most advanced humanoid robots ever created. They were designed to operate in hazardous environments and perform tasks such as opening doors, turning valves, and manipulating objects. The robots were equipped with advanced sensors, cameras, and other technologies that allowed them to navigate through complex environments and interact with their surroundings.
One of the most famous robots to participate in the DARPA Robotics Challenge was the Atlas robot, developed by Boston Dynamics [5]. It is capable of walking, running, and performing a variety of complex tasks, and includes a range of sensors and cameras allowing it to navigate through challenging environments.
Other notable robots that participated in the DARPA Robotics Challenge include the Valkyrie robot, developed by NASA, and the Hubo robot. These robots and others like them represent the cutting edge of robotics technology and are paving the way for a future in which robots will play an increasingly important role in a wide range of applications, from disaster relief to space exploration.

There are many different types of four-legged robots, each with their own unique names and characteristics. Here are a few examples:

2.4.2 AIBO (1999):

AIBO was developed by Sony, AIBO is a four-legged robot developed by Sony, a Japanese electronics company. AIBO is designed to be a robotic pet dog, that is able to perform different movements and behaviours, such as walking, playing, and expressing emotions. Aibo is equipped with a range of advanced sensors and cameras that allow it to link with its surroundings [11]. It also can recognize and respond to its owner's voice and facial expressions, allowing it to form a bond with its owner and respond to their cues.

Aibo has a range of features that make it a unique and engaging pet. For example, it can perform tricks, such as rolling over or playing fetch, and it can also express a range of emotions, such as happiness, sadness, and excitement. Aibo can also learn from its experiences and adapt its behaviour over time, becoming more responsive and engaged with its owner [13].

In addition to its role as a pet, Aibo has also been used in research applications, such as studying the interaction between humans and robots. Aibo represents a significant advancement in robotics technology, particularly in the field of social robotics, which aims to create robots that can interact with humans in meaningful and engaging ways.

2.4.3 Big Dog (2005):

Big-Dog is designed to be highly mobile and adaptable, with the ability to operate in a range of environments, from rough terrain to steep slopes. It is equipped with advanced sensors and cameras that allow it to navigate through complex environments and interact with its surroundings, and it can also carry payloads and perform a range of tasks, such as carrying supplies or equipment for military personnel.

One of the most impressive features of Big Dog is its mobility. It is capable of walking, trotting, and running at speeds of up to 4 miles per hour, and it can also navigate over rough

terrain and climb hills. Big-Dog's advanced control systems allow it to maintain balance and stability, even when carrying heavy payloads or operating in challenging conditions.

Big Dog has been employed in a variety of fields, such as the military and defense, search and rescue, and the delivery of supplies and equipment to hard-to-reach areas. It represents a significant development in robotics technology, especially for legged robots, which are intended to travel and function in conditions that are challenging or impossible for wheeled or tracked vehicles.

2.4.4 ANYmal (2016):

ANYmal is a quadruped robot developed by ANYbotics, a robotics company based in Switzerland. ANYmal is designed to operate in a range of environments, including industrial, construction, and search and rescue.
ANYmal is equipped with a range of advanced sensors and cameras that allow it to navigate through complex environments and interact with its surroundings. It is also highly mobile, with the ability to walk, trot, and climb stairs. ANYmal is also capable of carrying payloads and performing a range of tasks, such as inspection and monitoring [14].
One of the most impressive features of ANYmal is its adaptability. It is designed to operate in a range of different environments, including harsh and hazardous environments where human operators may not be able to operate safely. ANYmal's advanced control systems allow it to maintain balance and stability, even in challenging conditions.

ANYmal has a range of applications, including inspection and monitoring of industrial sites, search and rescue operations, and explore in the field of robotics. It represents significant advancement in robotics technology, particularly in the field of autonomous robotics, which aims to create robots that can operate independently and adapt to changing environments. Overall, ANYmal is a versatile and advanced robot that has the potential to revolutionize a range of industries and applications.

2.4.5 MINI CHEETAH (2018):

MINI CHEETAH is a four-legged robot developed by MIT's Biomimetic Robotics Laboratory. As the name suggests, Mini Cheetah is designed to mimic the movements and behaviors of a cheetah, one of the fastest land animals in the world.

Mini Cheetah is a small, responsive quadruped robot that can run, jump, and perform a range of acrobatic maneuvers. It is equipped with advanced sensors and cameras that allow it to navigate through complex environments and interact with its surroundings. It is also highly durable, with the ability to withstand falls and impacts without sustaining damage [6].

One of the most impressive features of Mini Cheetah is its mobility. It is capable of running at speeds of up to 5 miles per hour. Mini Cheetah's advanced control systems allow it to maintain balance and stability, even when performing complex movements and maneuvers.

Mini Cheetah has a range of applications, including entertainment. Its small size and agility make it well-suited for applications where larger, more cumbersome robots would not be suitable. It also represents a significant advancement in robotics technology, particularly in the field of bio-inspired robotics, which aims to create robots that mimic the movements and behaviors of animals in nature.

2.4.6 SPOT (2020):

Spot was developed by Boston Dynamics, Spot is a quadruped robot that is invented for a range of applications, including inspection, search, and rescue. It was equipped with advanced sensors and cameras that allow it to navigate through complex environments and interact with its surroundings, and it can also carry payloads and perform a range of tasks, such as inspection and monitoring [12].

One of the most impressive features of Spot was its mobility. It is capable of walking, trotting, and even running at speeds of up to 3.5 miles per hour, and it can also navigate over rough terrain and climb stairs. Spot's advanced control systems allow it to maintain balance and stability, even when performing complex movements and maneuvers.

These are just a few examples of the many different types of four-legged robots that exist. As robotics technology continues to advance, we can expect to see even more advanced and versatile robots with unique potential and capabilities.

CHAPTER - 3 HARDWARE

3.1 Electronic Components:

The electronic components of the project are discussed in this chapter. The electronic hardware, controllers and the actuators used are mentioned in this part.

3.1.1 Motor Selection:

TD-8325 MG Servo specification:

Dimensions: 40.6mm x 20.4mm x 39mm

Net weight: 60g

Operating Speed: 0.23sec/60° (6.6V)

Stall Torque: 25.3 kg-cm at 6.6V

Operation Voltage: 4.8 - 6.6 DC Volts

Working Frequency: 1520us/333hz

3.1.2 MPU-6050 Sensor:

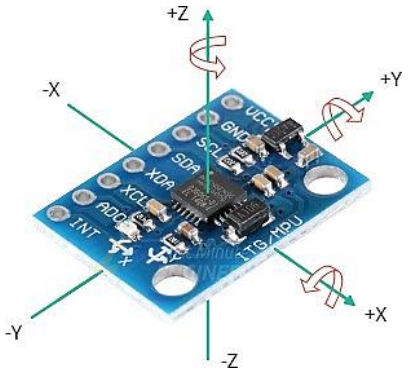


Figure 3-1 MPU 6050 Sensor

MPU6050 is a MEMS device that combines a 3-axis accelerometer and 3-axis gyroscope, enabling precise measurement of various motion-related properties such as acceleration, velocity, direction, displacement, and more.

3.1.3 PCA-9685 Servo Driver:

A single microcontroller can precisely control servo motors with the 16-channel PWM servo driver PCA9685. It uses I2C for communication, has programmable addresses for connecting multiple boards, allows variable PWM frequency, needs an external power source for servo motors, and requires I2C for communication. It makes servo motor control easier and makes accurate positioning possible.

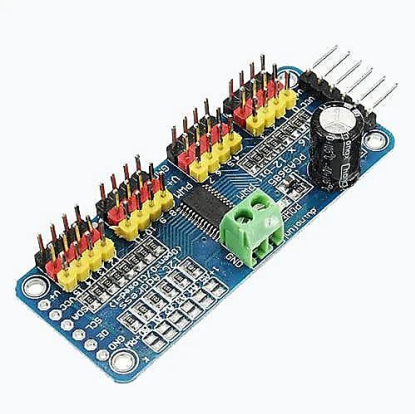


Figure 3-2 PCA-9685 Servo Driver

3.1.4 DC-DC Buck Converter:

An electrical circuit known as a DC-DC buck converter transforms a greater DC voltage into a lower DC value. It modifies the pulse width using PWM to regulate the output voltage. The converter distributes energy from an input source to an inductor, then to the output capacitor and load by opening and closing a power switch. The average output voltage of the PWM signal is determined by the duty cycle, allowing for voltage management. Buck converters are frequently employed in power supply and voltage regulation applications because they are effective, small, and portable.

3.2 Development:

3.2.1 Design of the Robot:

Inverse Kinematics:

The Denavit-Hartenberg (DH) parameters are a commonly used method for modeling the kinematics of robotic systems, particularly for serial link manipulators. DH parameters define the coordinate frames of each link in a robot arm in terms of four parameters:

1. θ: The angle between the Z_{n-1} and Z_n axes about the X_{n-1} axis.
2. d_n: The distance between the Z_{n-1} and $Z_n$axes along the X_{n-1} axis.
3. a : The length of the common normal between the Z_{n-1} and $Z_n$axes , calculated along the X_{n-1} axis.
4. α: Angle between the X_{n-1} and X_n axes about the Z_{n-1} axis.

Using DH parameters, the transformation matrix from one link to the next can be computed using a set of standard equations. DH parameters are useful for modeling the kinematics of robotic systems because they allow for a standardized and systematic approach to deriving the kinematic equations of the system. They are also relatively simple to compute and can be used to derive closed-form solutions for simple robot configurations. [15] The frame assignment for this robot was done using DH-method.

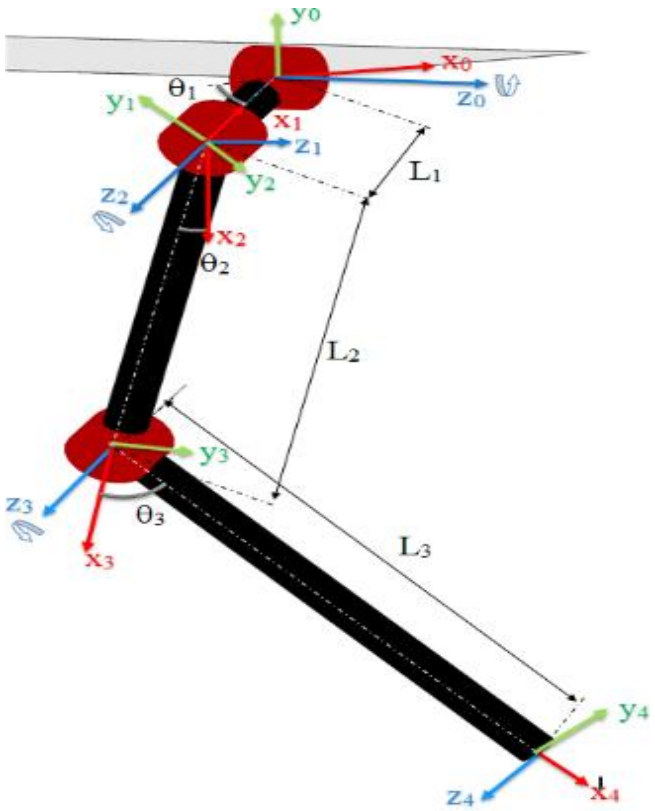


Figure 3-3 Different leg parameters and coordinates shown.

Inverse kinematics refers to the computation of joint angles required for a robot to achieve a desired end position and orientation in the workspace. The inverse kinematics of a four-legged robot can be determined using various techniques, such as analytical methods, numerical methods, and optimization-based approaches.

One common approach for inverse kinematics of a four-legged robot is to use a geometric method called the geometric Jacobian method. In this method, the position and orientation of the end-effector of each leg are determined using the forward kinematics equations. The geometric Jacobian matrix is then computed using the partial derivatives of the end-effect or positions and orientations with respect to the joint angles. [16]

The inverse kinematics solution can then be obtained by solving the following equation:

$$\Delta q = J^{-1}\Delta x$$

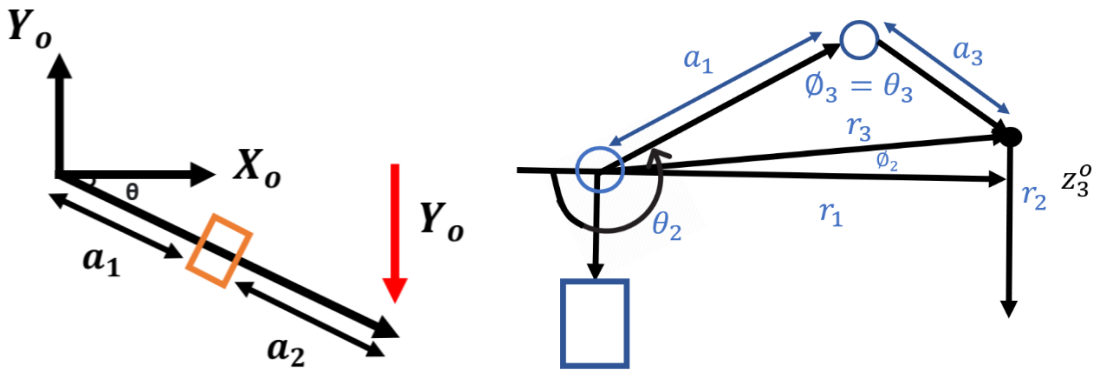
Equation 1: Inverse Kinematics Equation

where Δq is the change in joint angles required to achieve the desired change in end-effector position and orientation Δx , and J^{-1} is the inverse of the geometric Jacobian matrix.

Another approach for inverse kinematics is to use an optimized method such as the gradient descent or Newton-Raphson method. These methods involve iteratively updating the joint angles to reduce the error between the desired and actual end-effector location and direction.

The choice of inverse kinematics method depends on the specific requirements and constraints of the robot and its task. For this task, we used a geometric approach, and the frame assignment was done using DH method.

3.2.2 Calculations:



$$\theta_1 = tan^{-1}\left(\frac{Y^{0-3}}{X^{0-3}}\right)$$

$$\theta_2 = 180 + (\phi_1 + \phi_2)$$

$$\theta_3 = \phi_3$$

$$a_3{}^2 = a_2{}^2 + r_3{}^2 - 2a_2 * r_3 * cos\phi_1$$

$$\phi_1 = arccos\left(\frac{a_3{}^2 - a_2{}^2 - r_3{}^2}{-2 * a_2 * r_3}\right)$$

$$\theta_3 = arccos\left(\frac{-a_3{}^2 - a_2{}^2 + r_3{}^2}{-2 * a_2 * a_3}\right) = \phi_3$$

$$r^2{}_1 = Y^{0^2}{}_3 + X^{0^2}{}_3$$

$$r_2 = Z^0{}_3 - a_1$$

$$r_3 = (r_1 + r_2)^{1/2}$$

3.3 Program Code:

The equations were written to code:

```
r_1 = sqrt((y0_3*y0_3)+(x0_3*x0_3));
r_2 = z0_3 - a_1;
r_3 = sqrt((r_1*r_1)+(r_2*r_2));

phi_1 = acos(( (a_3*a_3) - (a_2*a_2) - (r_3*r_3) )/ ( -2*a_2*r_3 ) );
phi_2 = atan(r_2/r_1);

theta_2 = ((PI)+phi_1+phi_2)*RAD_TO_DEG;

theta_3 = acos((-(a_3*a_3)-(a_2*a_2)+(r_3*r_3))/(-2*a_2*a_3))*RAD_TO_DEG;
theta_1 = atan2(y0_3,x0_3)*RAD_TO_DEG;

if (theta_1<0)
{
  theta_1 = theta_1+360;
}

float pulse_1 =servo1_min_pulse+(pulse_factor_1)*(theta_1-180);
float pulse_2 =servo2_min_pulse+(pulse_factor_2)*(theta_2-90);
float pulse_3 =servo3_min_pulse+(pulse_factor_3)*(theta_3);
```

Figure 3-4 Program code written through the equations.

The pulse(milli-seconds) will be given to PWM command of the driver.

3.3.1 Results:

The following results were achieved:

For X=30mm, Y= -50mm, Z=30mm:

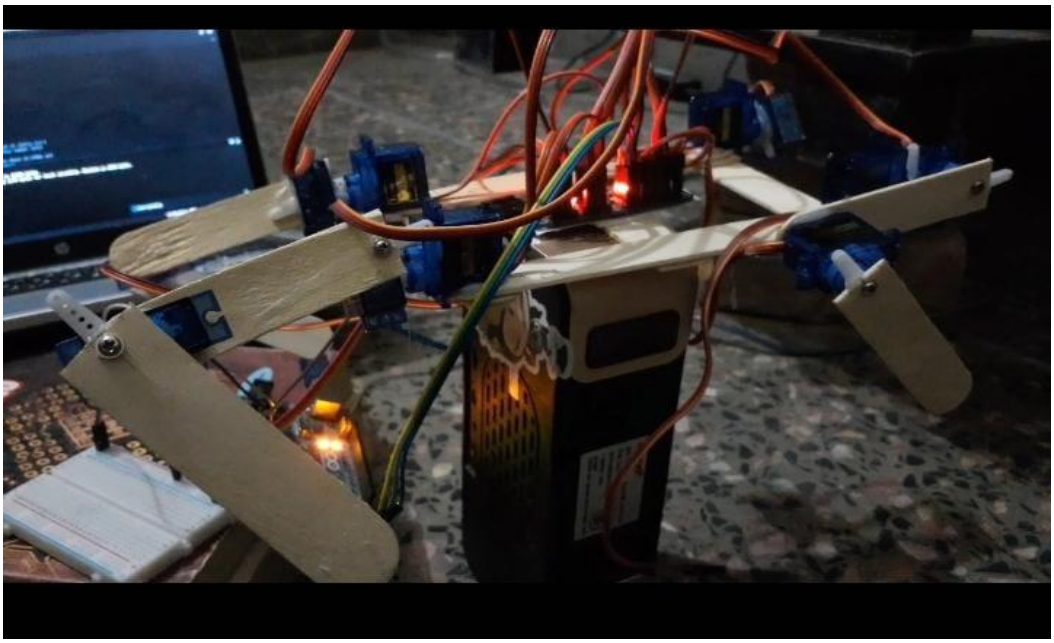


Figure 3-5 Leg movement on mentioned coordinates

For X=100mm:

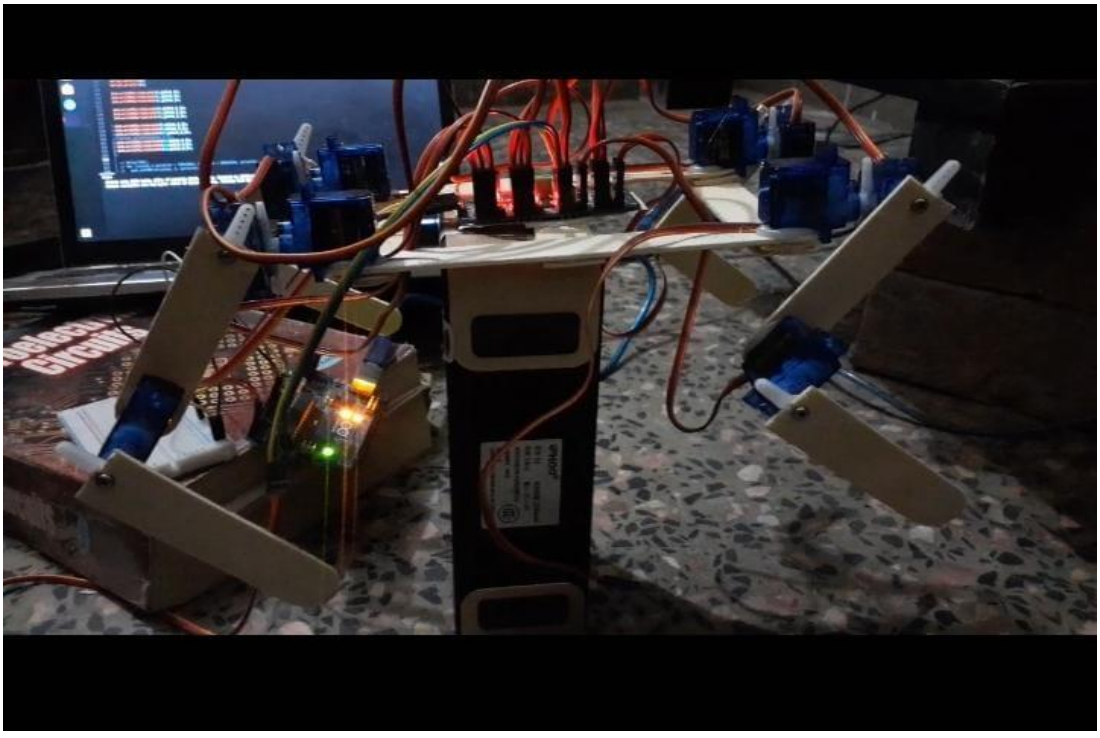


Figure 3-6 Movement of legs along x-direction

3.4 On- Board Electronics:

The on-board electronics of the robot are as follows:

- 12 × 25Kg cm Servos (TD-8325 MG)
- Raspberry Pi 4b
- Arduino Uno
- PCA9685 PWN Servo driver.
- MPU 6050
- Battery

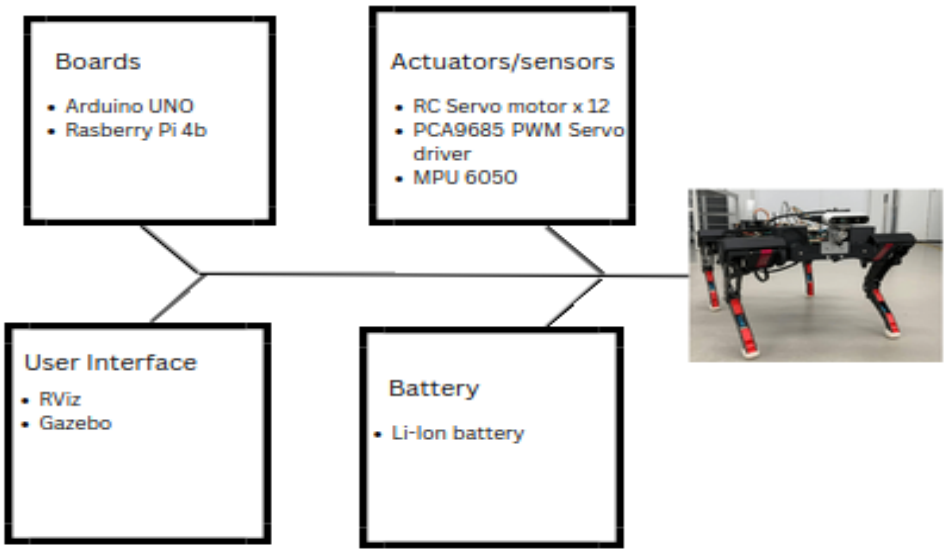


Figure 3-7 Project functional diagram.

Each leg consists of three servos: For hip, upper and lower leg. The servos are connected to PCA9685 PWM servo driver to control the servo and the servo driver is connected to motor controller (Arduino UNO) that sends commands to the servo.
Arduino UNO sends and receives command from the Raspberry Pi module. Raspberry Pi acts as the brain of the robot and is responsible for the overall geometry of the robot.

3.5 Power Supply:

Keeping in mind the voltage and current requirements for motors, microcontroller, and Arduino, the power supply was chosen.

Component	Voltage-Current	Quantity
Raspberry PI	5V-3A	1
Arduino UNO	6V	1
25kg cm Servo	6V-(0.5 A-0.9 A)	6-12 in use at any instance

Table 3-1 Table showing components with specifications.

CHAPTER - 4 SOFTWARE

4.1 ROS Robot Control:

The robot is powered by Robot Operating system (ROS). ROS (Robot Operating System) is a set of software libraries and tools that help developers build robot applications. It provides architecture for writing and distributing software, as well as a set of conventions for naming, structuring, and communicating between components of a robot system. ROS is often used to control robots in research and industry, and it supports a wide range of hardware platforms, including robots with wheels, arms, and flying drones. ROS runs on Raspberry Pi module that makes necessary decisions for the robot and can provide real-time simulation. This is done with the help of RViz, a 3D visualization tool for ROS and Gazebo, a physics simulator.

Node: A node is an executable that uses ROS to communicate with other nodes. A node can be Arduino in this case.

Topic: Nodes can *publish* messages to a topic as well as *subscribe* to a topic to receive messages.

To begin with, a simple 3 DOF leg simulation was built up on ROS and to connect it to the real world, Arduino was treated as node. A 3D model of leg was generated that included the 3 servos indicating the 3DOF. It is a type of Unified Robotics Description format written in XML that generates 3D model in ROS programs. A RViz view of the leg is as follows:

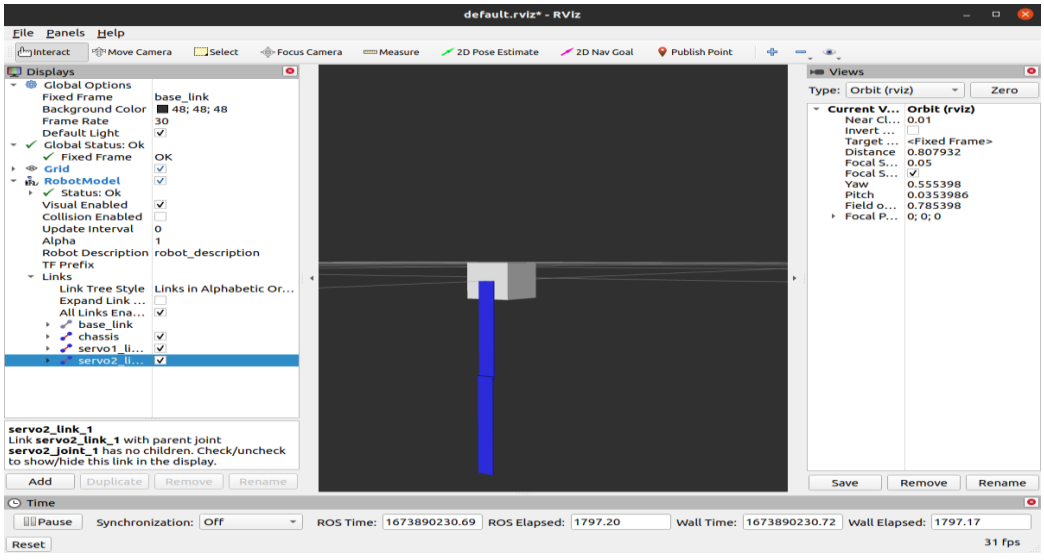


Figure 4-1 ROS interface showing leg.

The upper and lower leg are represented as purple sticks while the hip joint is represented as grey cube. The shape will be later changed to the final design of the leg.
The physical implementation of it is as follows:

Top- View:

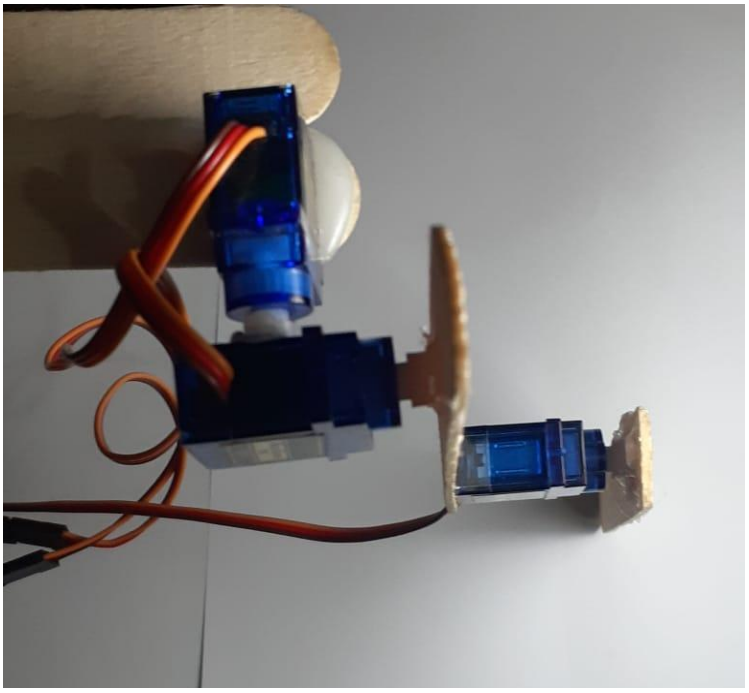


Figure 4-2 Top View.

Side- View:

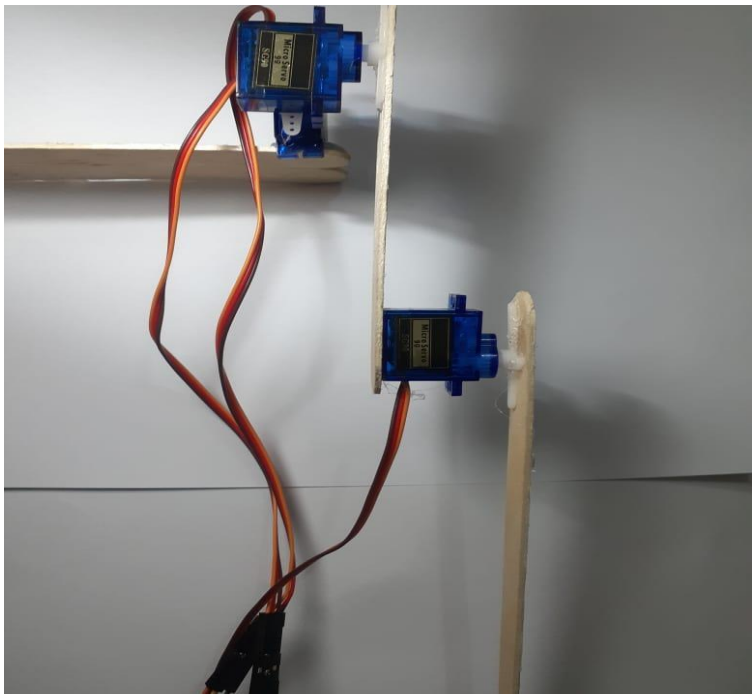


Figure 4-3 Side View.

Front- View:

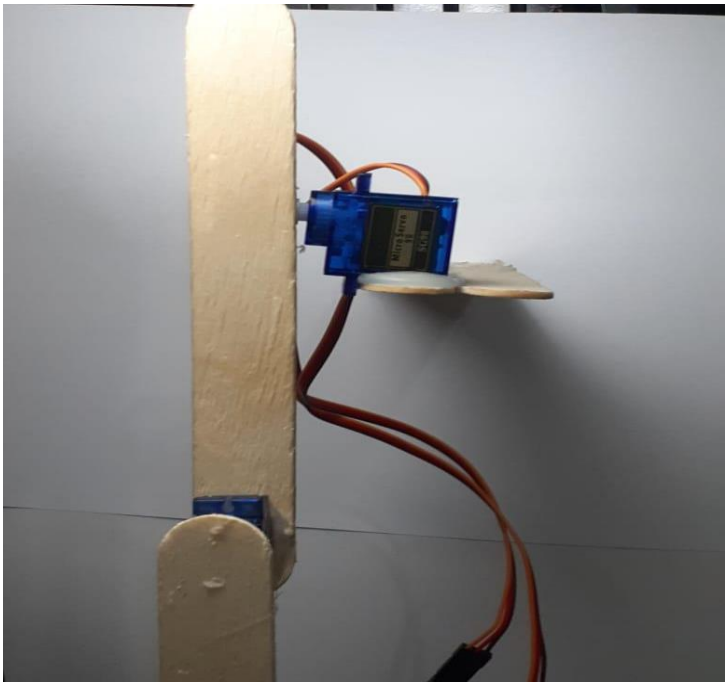


Figure 4-4 Front View.

Here a simple SG-90 Hobby servo is used to demonstrate the leg and represent the joints. The three servos connect to Arduino (The motor controller). Communication via ROS to Arduino occurs with another communication protocol named as “ROS serial”.

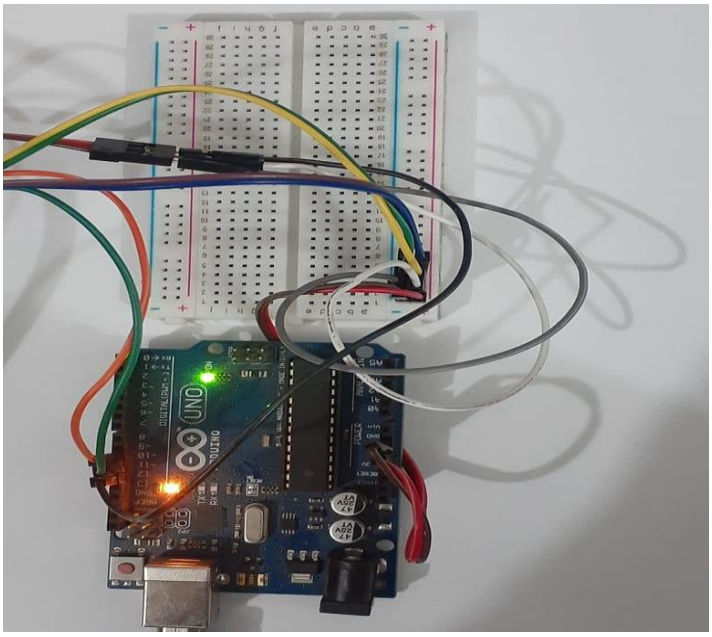


Figure 4-5 Connections Shown.

The whole ROS computing graph, also called RQT- Graph, comes out as follows:

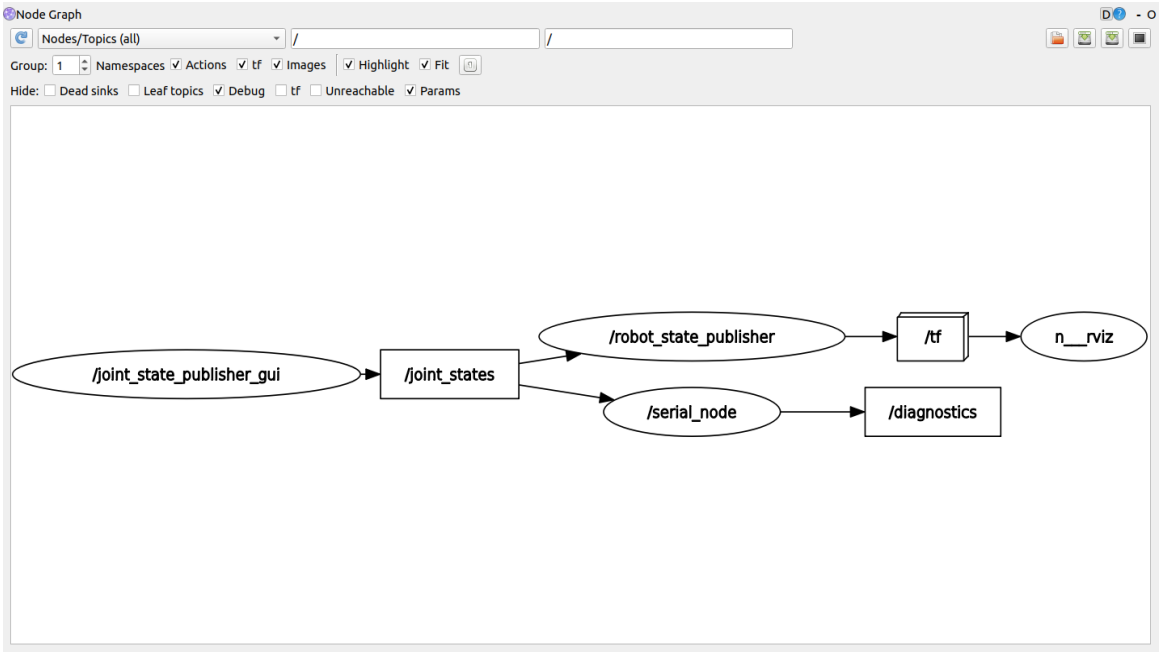


Figure 4-6 ROS Computing Graph

The circular shapes are known as topics and the rectangular box are called nodes.
The “robot state publisher” takes angle values from “/joint states” and sends them to the rviz node, which can be seen in action virtually.
“/joint_state_publisher_gui” gives the angle values to the “joint states” topic that let the robot know what the angle of each leg joint should be. These values are then sent to Arduino via serial node topic, that publishes the angle values to servos.

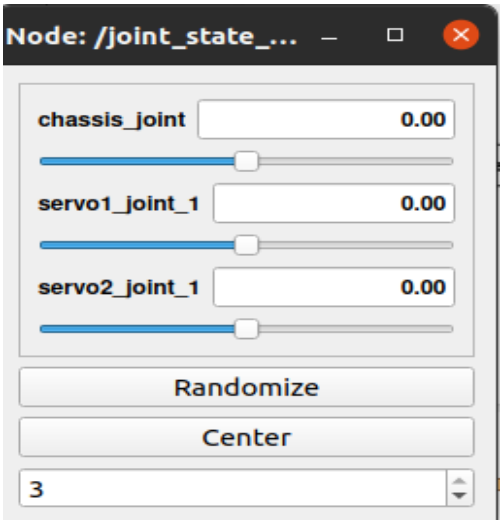


Figure 4-7 Joint Values.

Each line represents the angle in radian. When moved, the angle value changes which is then send to the Arduino and servos act accordingly.

For the second version of the ROS architecture, each process of robot was broken into smaller modules. The following were the modules:

- Controller node
- Gait generator
- Bezier Curve
- Inverse kinematics
- Joint states
- Arduino node (ROS Serial)
- Rviz

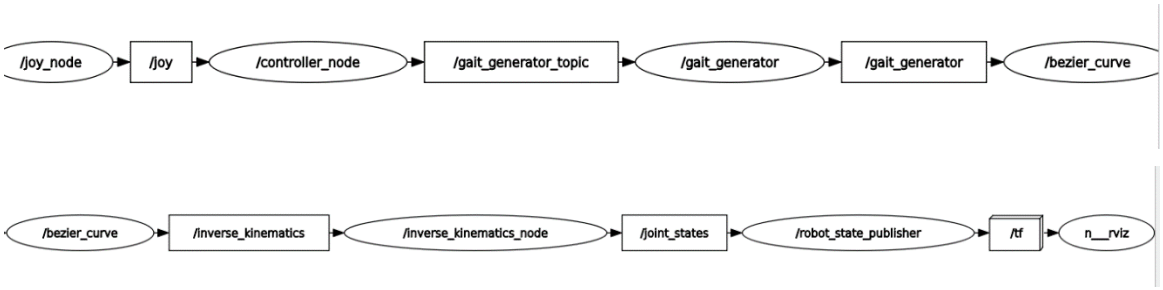


Figure 4-8 RQT- Graph.

4.1.1 Controller node:

The controller node took values from the Xbox controller and had it sent to the gait generator node. It sends to the gait generator via giat_generator_topic message. The message was the controller parameters namely of joystick, trigger and a button.

4.1.2 Gait Generator:

The values received from the node was an array consisting of values representing the buttons and joystick values. For a motion like trot which is a diagonal motion of the legs, the gait generator generates diagonal array values. {Leg1, Leg2, Leg3, Leg4}. For the array, each leg is either 1 or 0, indicating the activation for 1 and not activated for 0.

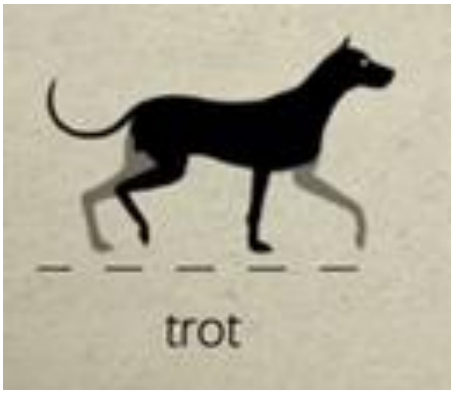


Figure 4-9 Trotting motion.

Array gait indicates Joy node values retrieved from the joystick. It indicates the different motion:

- array_gait[0]: Forward and backward motion. It varies from 1 to -1. 1 means forward and -1 means backward.
- array_gait[1]: Left and Right motion. It varies from 1 to -1. 1 means left and -1 means right.
- array_gait[2]: Rotate Left and Right motion. It varies from 1 to -1. 1 means rotate left and -1 means rotate right.
- array_gait[4]: Gait switch. It varies from 1 to 0. 1 means optional gait and 0 means trot.

Figure 4-10 X-Box Controller.

```
int array[H][W] = {
  {array_gait[0], array_gait[1], array_gait[2], array_gait[3]},
  {1, 0, 1, 0},
  {0, 1, 0, 1}};
```

Figure 4-11 Program Code.

For a motion like trot, the diagonal motion of legs is achieved by sending a sequence array matrix of 3x4. The 2nd and 3rd row indicate which leg to activate and which to not.

4.1.3 Bezier curve node:

This node is the crucial part for the motion of the leg. It generates way points for the leg to move, either forward, backward, left, or right depending on the user choice.
Bezier curve generates the following curve for the forward motion:

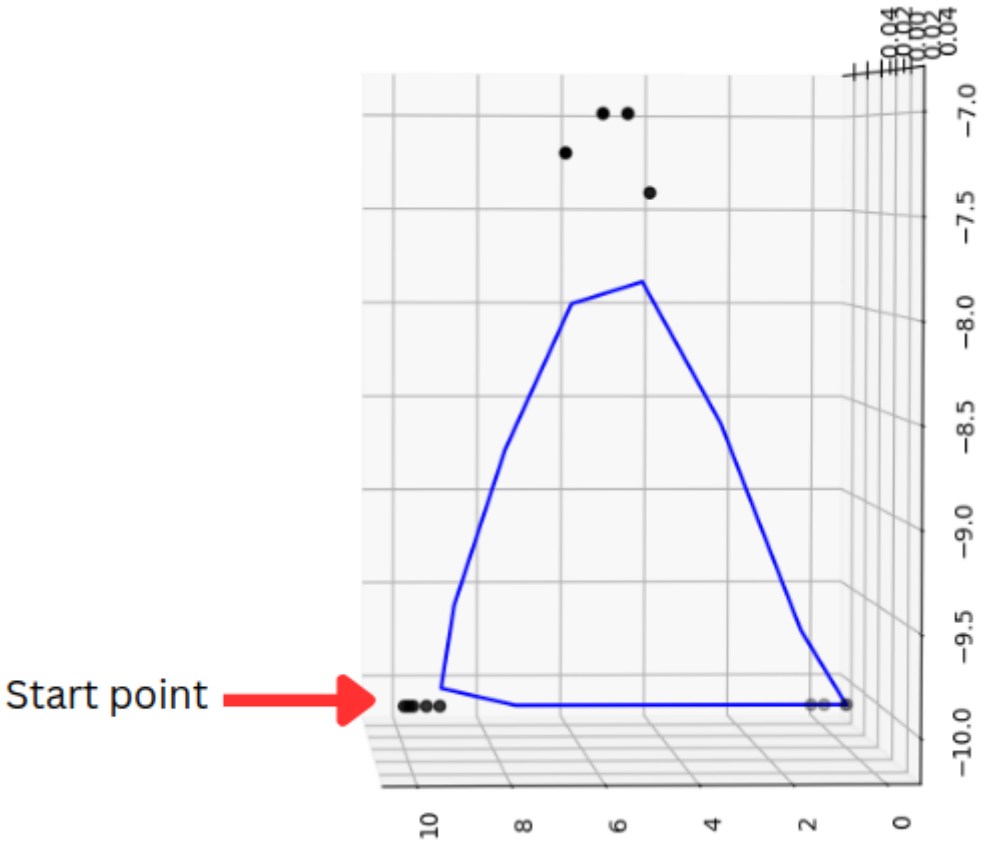


Figure 4-12 Forward Motion Curve.

The leg is expected to have motion on the following path depending on the workspace for the leg. It moves on the x-y axis.

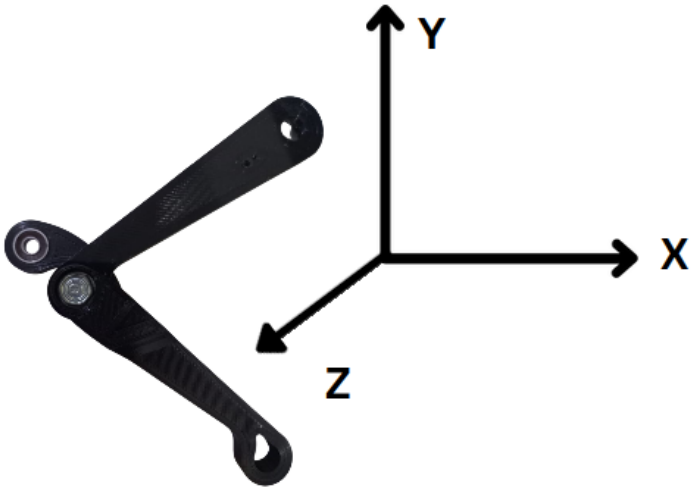


Figure 4-13 Axis Shown.

4.2 Inverse Kinematics:

The values/way points from the bezier curve are fed into the inverse kinematics node which then are responsible for the joint values of the three motors. The inverse kinematics is calculated by geometrical method and is fed into the motors via joint_state_node.

4.2.1 Joint State node:

Joint state node is a combination of different arrays in a ROS environment. It consists of name, position, velocity, effort.

```
std_msgs/Header header
string[] name
float64[] position
float64[] velocity
float64[] effort
```

Here in this scenario, since only joint states are being calculated, therefore, the position of the motors is put in. Each motor is given a separate message variable.

```
sensor_msgs::JointState joint_state_msg_leg1_joint1;
sensor_msgs::JointState joint_state_msg_leg1_joint2;
sensor_msgs::JointState joint_state_msg_leg1_joint3;
sensor_msgs::JointState joint_state_msg_leg2_joint1;
sensor_msgs::JointState joint_state_msg_leg2_joint2;
sensor_msgs::JointState joint_state_msg_leg2_joint3;
sensor_msgs::JointState joint_state_msg_leg3_joint1;
sensor_msgs::JointState joint_state_msg_leg3_joint2;
sensor_msgs::JointState joint_state_msg_leg3_joint3;
sensor_msgs::JointState joint_state_msg_leg4_joint1;
sensor_msgs::JointState joint_state_msg_leg4_joint2;
sensor_msgs::JointState joint_state_msg_leg4_joint3;
```

Figure 4-14 Variables for Joint Values.

4.3 ROS-Serial:

ROS-Serial is a communication library in ROS (Robot Operating System) that allows communication between a ROS-based computer and a microcontroller or embedded system. In this case, the microcontroller used is Arduino which is used to communicate with the computer. Arduino has 12 servos connected to PCA9685 where 12 servos are connected.

4.4 CAD- Model:

4.4.1 Leg Design:

The legs were designed using SolidWorks software. Designing a leg for a robot dog requires a multidisciplinary approach that involves mechanical and electrical engineering. Here are some general steps to consider when designing a leg for a robot dog:

Determine the leg's function: Decide on the intended use for the leg, such as walking, running, jumping, or climbing. This will help guide the design process and determine the necessary range of motion and strength requirements.

Determine the leg's dimensions: Determine the size and shape of the leg based on the intended use and the size of the robot dog.

Choose the joint types: Select the type of joints for the leg, such as rotary or linear actuators. This will determine the range of motion for the leg.

Determine the actuator type: Choose the type of actuator, such as a servo or a stepper motor, to drive the joints.

Design the mechanical structure: Create the mechanical structure of the leg using CAD software. This should include joints, actuator mounts, and linkages.

Developing the control system: Develop a control system to command the leg's movements, such as through a microcontroller or computer.

Testing and iterations: Test the leg and make necessary adjustments to improve its performance and durability.

Overall, designing a leg for a robot dog requires a thorough understanding of mechanics and electronics. It's also important to consider the aesthetics of the leg, as it should look and move like a natural dog’s leg.

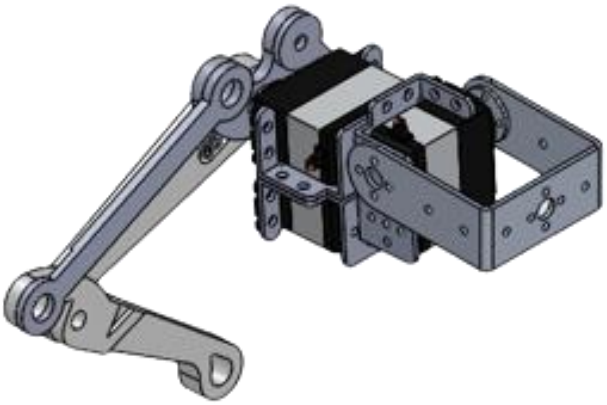


Figure 4-15 Leg Assembly

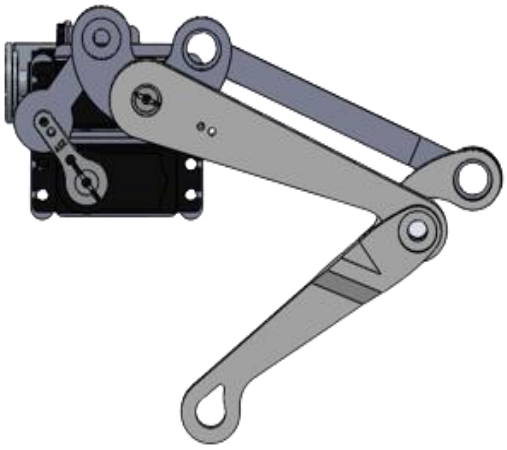


Figure 4-16 Side Close-up View

Design of the robot leg was created in SolidWorks, by following these steps:

1. Create a new assembly file.
2. Design the leg components (upper/lower leg segments, joints).
3. Import/insert leg components into the assembly file.
4. Define joints between leg components using mechanical mates.
5. Test the movement of the leg assembly.
6. Add actuators and sensors to the leg assembly.
7. Refine and iterate the leg assembly design.
8. Create engineering drawings of the leg assembly.

4.4.2 CAD- Assembly:

The assembly of the overall four-legged robot looks like this as shown in:

Figure 4-17 CAD Design showing links and legs connection.

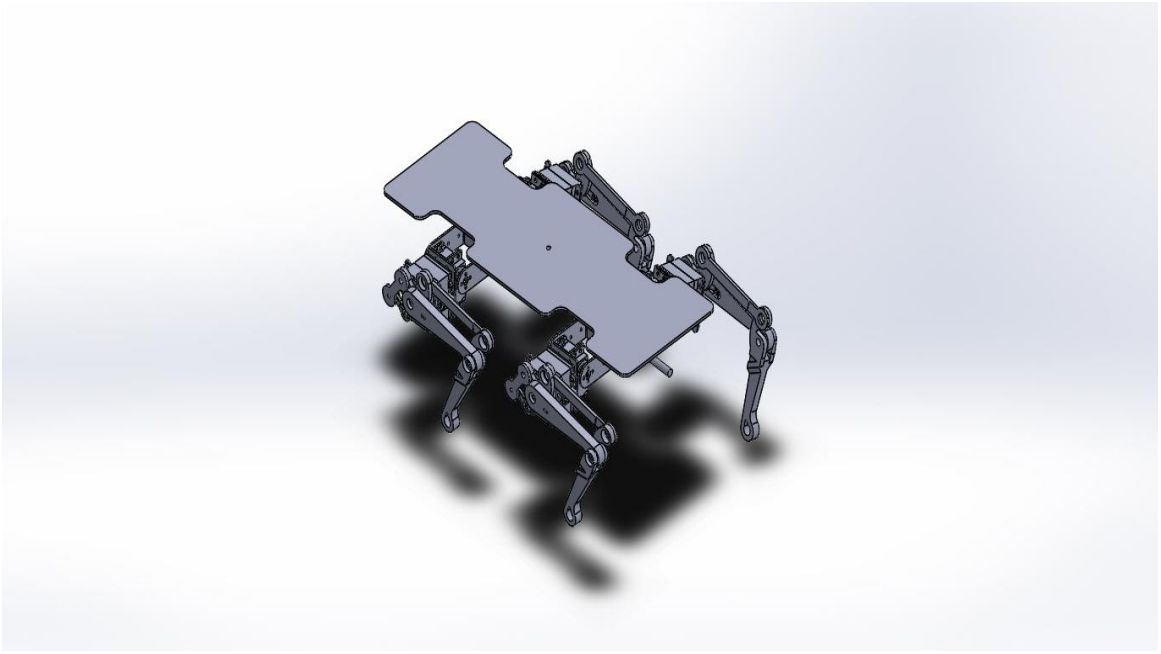


Figure 4-18 Complete Design.

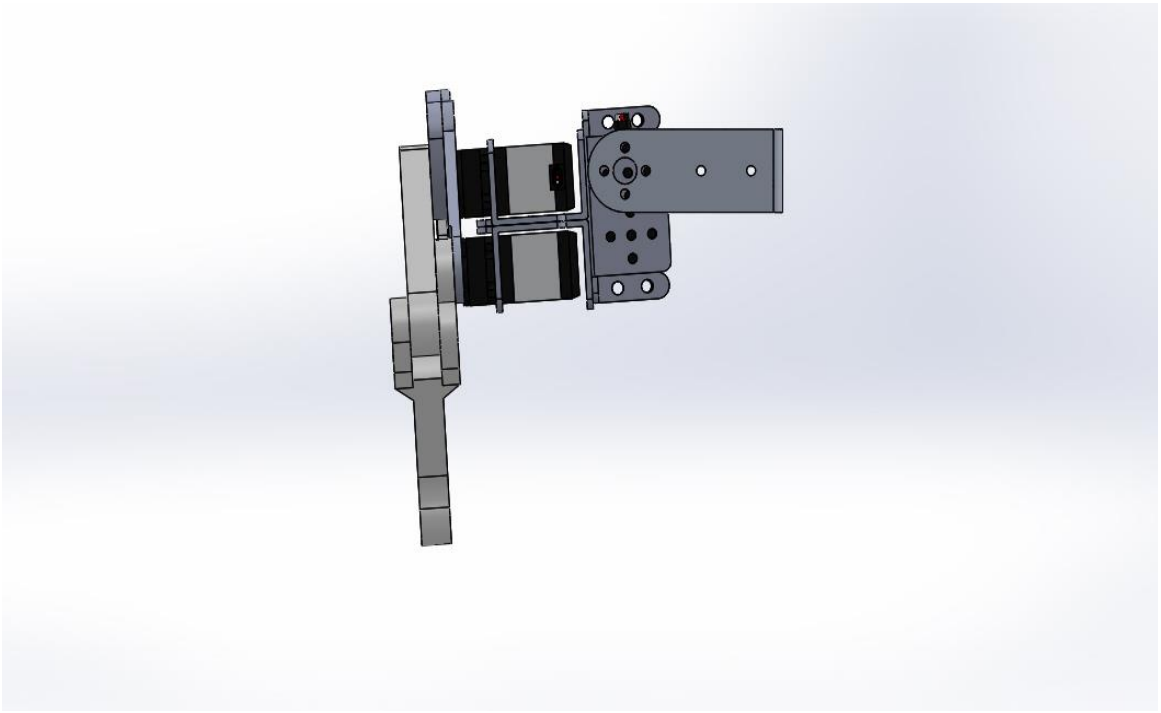


Figure 4-19 Leg Assembly (Front View).

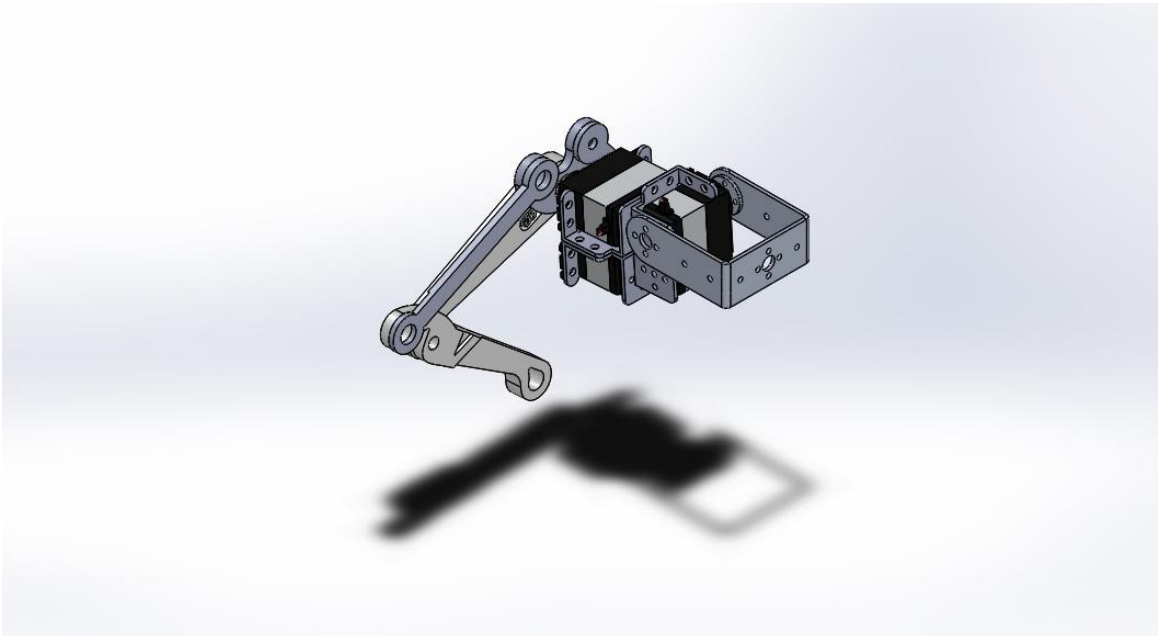


Figure 4-20 Leg Assembly (Side view)

4.5 Design Parameters:

Length	400 mm
Width	150 mm
Height (Standing)	110 mm
Net Weight	3 kg
DOF	12 (3DOF / Leg)

Table 4-1 Table Showing Design Parameters

The robot designed was a 12-DOF robot with 3-DOF for each leg. For an optimal design the length of the robot was chosen to be 400 mm with 150 mm width, and the standing height achieved was 110 mm. After assembling the components, the net weight was around 3 kg.

CHAPTER - 5 Testing & Operation Guidelines

The robot was programmed to follow the user input commands. Using different commands input by the user, the legs were actuated using servo motors, equipped with sensors for measuring joint angles and ground contact. The control system allowed custom software for coordinating the leg movements and maintaining balance.

5.1 Testing:

The following procedure was followed for testing the control circuit that how it senses and controls the movement while achieving stability on different terrains.

a. The circuit was provided with the voltage and tested for variety of inputs from the controller and output was recorded. The leg was first positioned and then mounted where it produced maximum output.
b. The motor- control was programmed through Raspberry Pi.
c. For additional stability feedback signal was taken, and the controller was programmed to follow certain gait, and if there is certain obstacle that it couldn't overcome the motor/ actuator should stop instead of stalling.
d. Upon successful testing the legs were attached to their respective positions.

5.2 Legs Testing:

5.2.1 Gait Testing:

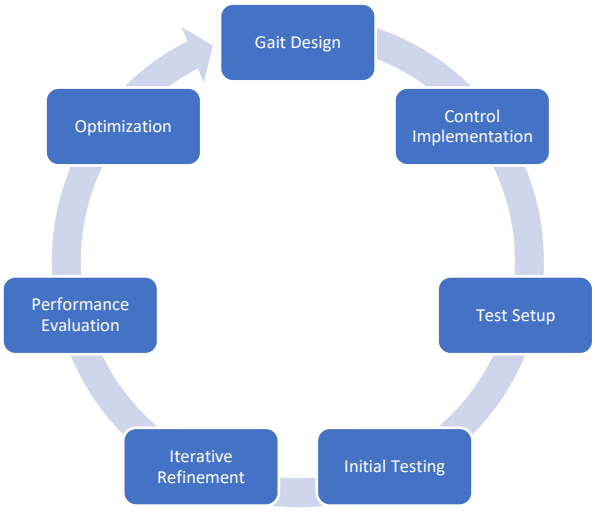


Figure 5-1 Mean & Extreme positions Shown.

CHAPTER - 6 Results & Conclusions

This chapter includes the results and documentation of the whole project. Keeping in mind the importance of these robots, suggestions and improvements are also discussed for future work and research in this area.

6.1 Fabrication of Parts:

The parts of the robot like the legs and supporting links were 3D Printed using PETG material. The parts were printed using a honeycomb structure that is light weight and has high strength to weight ratio. Honey-comb structure additionally allows efficient usage of material while reducing wastage during printing. In addition to being more elastic and robust than other designs, honeycomb infill can withstand higher levels of stress and vibration without breaking or deforming. For 3D printed components that need to have great strength, durability, and stability, such as mechanical components, tools, or functional prototypes, honeycomb infill is appropriate.

Figure 6-1 Picture of the 3D- Printed leg.

Figure 6-2 Connecting links for Lower Leg.

Figure 6-3 Connecting Link.

6.1.1 PETG for 3D- Printing:

The fabrication of the parts was done using 3D- Printing and PETG was used as printing material as it has the following characteristics:

a. Good mechanical properties such as impact-strength, tensile properties, heat deflection. It can withstand impact and stress and can bear loads without significant deforming and breaking.

b. It has good flexibility properties that help isolate the vibrations caused due to roughness and other parameters.

c. It can be post processed and enables processes like sanding, painting, customization, and refining.

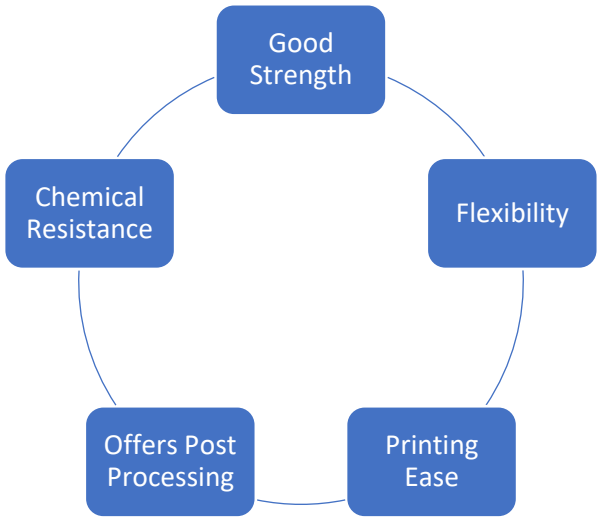


Figure 6-4 PETG Benefits

Considering high strength and good mechanical and load bearing capabilities and flexibility PETG was used for fabricating the legs and the connecting links which connected upper and lower parts (links) of the leg.

Infill	Solid
Filament Color	Black, Pink
Filament Material	PETG

Table 6-1: Material Properties

6.2 Performance Parameters of the Robot:

DOF of Legs	3 DOF/ Leg
Maximum Load	2 kg
Stall Current (Per Motor)	2.5 A at 6Volts
Motor Max Rotation Angle (Degrees)	180°

Table 6-2: Robot Performance Parameters

6.3 Leg- Design:

The legs and links were designed to meet the requirements and were printed locally thereby ensuring the low cost and an efficient design, the servo motors were mounted on the shoulder and a compliant linkage was made in order to have less weight on the leg links and joints. The design is illustrated below:

Figure 6-5 Leg with upper and lower links connected.

6.4 Suggestions and Recommendations:

6.4.1 Research and Future Work:

- For narrower paths and congested places, the size of the overall robot should be smaller ideally, but as far as our product is concerned if the size reduction is carried out the fitting of the actuators and the components will be restricted.

- The current robot can be programmed to follow different walking patterns e.g., galloping, trotting, and crawling while maintaining stability on different terrains.

- Artificial Intelligence could be implemented to achieve maximum benefits without human intervention, the AI adaptive learning skills will help the machine to adapt different structures and would enable the machine to respond to different hurdles while walking and will also make it familiarize with the walking patterns like the biological species.

- For future designing the legs can be re-modelled by increasing the DOF, the links can also be re-designed in such a way to achieve maximum joint angles that can help the robot reverse the link direction instead of turning on a whole to move backward.

- Motors should be cost effective and efficient in order to serve the desired purpose, the rated specs as mentioned on the motor data sheet shouldn't be fully trusted, the motor can be damaged if limit, less than the mentioned stall limit is reached. Torque and RPM should be carefully monitored.

- To minimize errors and frequent faults, closed-loop feedback systems can be implemented, along with regular fault detection and indication procedures.

- Material Selection should be of the main importance while fabricating such kind of a robot, PETG is widely used for 3D printing applications but for the applications involving a high loads PTEG should be replaced with some metal alloys as PETG

can be fragile, since our project is a prototype and would be mainly used for the research purposes to be carried out so PETG is acceptable, whereas for the commercial purposes it should be replaced with some suitable hard material.

References

[1] P. Britton, "Engineering the new breed of walking machines," *Popular Science,* pp. 66-69, 1984.

[2] M. Raibert, Legged Robots That Balance, Massachusetts : The MIT Press, 1986.

[3] C. D. B. M. B. L. W. K. Holtmann, "Advances in Real-World Applications for Legged," p. 22, 2017.

[4] T. E. a. T. M. S H Hyon, "Dynamics-based control of a one-legged hopping robot," *SAGE Journals,* pp. 83-98, 2016.

[5] E. G. EVAN ACKERMAN, "The Next Generation of Boston Dynamics' ATLAS Robot Is Quiet, Robust, and Tether Free," *IEEE Spectrum,* FEB 2016.

[6] B. D. C. J. &. K. S. Katz, "A platform for pushing the limits of Quadruped Control," *ICRA IEEE,* pp. 6295-6301, 2019.

[7] W. K. S. &. H. N. Bosworth, "The mit super mini cheetah: A small Low Cost Quadrupedal robot for dynamic locomotion," *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR),* pp. 1-8, 2015.

[8] D. T. C. &. W. R. Wettergreen, "xploring Mount Erebus by walking robot.," *Robotics and Autonomous Systems,* pp. 171-185, 1993.

[9] M. &. B. D. &. S. D. &. T. F. &. J. F. &. M. Bjelonic, "ANYmal on Wheels: Exploiting the Advantages of Wheeled and Legged Robots on Varying Terrain," *Marco,* 2018.

[10] P. B. Prases K. Mohanty, "Development of quadruped walking robots: A review," *Ain Shams Engineering Journal,* vol. 12, no. 2, pp. 2017-2031, Jun 2021.

[11] Genghis, "robots.ieee.org/robots/genghis," *Spectrum IEEE,* 2018.

[12] H. Kalouche, "Design and fabrication of a biomimetic quadruped robot.," *Bioinspiration & biomimetics 3.3,* 2008.

[13] C. Semini, "Design and control of the quadruped robot," *The International Journal of Robotics Research,* pp. 400-430, 2013.

[14] C.-L. Tsai, "Design and implementation of a biomimetic quadruped robot.," *Journal of Bionic Engineering,* pp. 329-339, 2013.

[15] P. &. M. P. K. Biswal, "Development of quadruped walking robots," *Ain Shams Engineering Journal,* 2021.

[16] S. e. a. Kim, "Design and fabrication of an improved robot quadruped with enhanced mobility," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*, 2008.

[17] M. H. Raibert, "Bigdog, the rough-terrain quadruped robot.," *Field and Service Robotics. Springer, Berlin, Heidelberg,* pp. 341-350, 2008.

[18] H. a. A. K. Park, "Design and control of a quadruped robot for rough terrain locomotion," *Journal of Field Robotics ,* pp. 141-152, 2005.

Appendix

Appendix I - Inverse Kinematics Code

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "sensor_msgs/JointState.h"
#include "kinematics_legged/bezier.h"
#include "std_msgs/Float64MultiArray.h"
#include <typeinfo>
#define PI 3.14159265358979323846264338327795
#define HALF_PI 1.57079632679489661923132169163975
#define TWO_PI 6.28318530717958647692528676655901
#define DEG_TO_RAD 0.017453292519943295769236907684886
#define RAD_TO_DEG 57.295779513082320876798154814105
// void chatterCallback(const kinematics_legged::bezier::ConstPtr& msg)
// {
//   std::vector<float> y;
//   y.reserve(msg->y.size());  // allocate memory for the output vector
//   for (const auto& element : msg->y) {
//     y.push_back(element.data);  // copy the value of each element to the output vector
//   }

//   ROS_INFO("I heard Y: [%f]", msg->y.data());

// }
float red_servo_min_pulse = 600;
float red_servo_max_pulse = 1800;
float red_pulse_factor = ((red_servo_max_pulse-red_servo_min_pulse)/180);

float servo_min_pulse = 600;
float servo_max_pulse = 2400;
float pulse_factor = ((servo_max_pulse-servo_min_pulse)/180);

std::vector<std::vector<int>> rows;
std::vector<float> xBezier;
std::vector<float> yBezier;
std::vector<float> zBezier;
int ROWS=3;
 int COLS=4;
int size_vector;

void chatterCallback(const kinematics_legged::bezier::ConstPtr& msg) {
 ROS_INFO("Received %d elements in x, y, and z arrays", msg->x.size());

 // Print the contents of the x, y, and z arrays

 xBezier.resize(msg->x.size());
 yBezier.resize(msg->x.size());
 zBezier.resize(msg->z.size());

 float x=0;
 float y=0;
 float z=0;

 for (int i = 0; i < msg->x.size(); i++) {
   x=msg->x[i].data;
   y=msg->y[i].data;
```

```
  z=msg->z[i].data;
  size_vector = msg->x.size();


  // ROS_INFO("x[%zu]", i);
  xBezier[i]=x;
  yBezier[i]=y;
  zBezier[i]=z;
  //ROS_INFO("x=%f,y=%f,z=%f", xBezier[i],yBezier[i],zBezier[i]);

 }
    ROS_INFO("Received %d elements ROW SIZE", msg->paramset[0].layout.dim[0].size);

 if (msg->paramset[0].layout.dim[0].size==3)
    {
     ROWS=3;
    }
    else if (msg->paramset[0].layout.dim[0].size==5)
    {
     ROWS=5;
    }
     else if (msg->paramset[0].layout.dim[0].size==2)
    {
     ROWS=2;
    }
 if (msg->paramset[0].layout.dim[0].size == ROWS && msg->paramset[0].layout.dim[1].size == COLS)
  {
      // Access the matrix elements

    for (int i = 0; i < ROWS; i++)
    {
       std::vector<int> rowArray;
       for (int j = 0; j < COLS; j++)
       {
          int element = msg->paramset[0].data[i * COLS + j];
          rowArray.push_back(element);
       }
       rows.push_back(rowArray);
    }
  }

}

int main(int argc, char **argv)
{
 float r_1,r_2,r_3,phi_1,phi_2,theta_1,theta_2,theta_3;

 float a_1 = 1;//cm
 float a_2 = 10;//cm
 float a_3 = 10;//cm


 ros::init(argc, argv, "inverse_kinematics_node");
 ros::NodeHandle n;
 ros::Subscriber sub = n.subscribe("inverse_kinematics", 1000, chatterCallback);
 ros::Publisher pub = n.advertise<sensor_msgs::JointState>("joint_states", 1000);
 ros::Rate loop_rate(15);
 sensor_msgs::JointState joint_state_msg_leg1_joint1;
 sensor_msgs::JointState joint_state_msg_leg1_joint2;
 sensor_msgs::JointState joint_state_msg_leg1_joint3;
```

```
sensor_msgs::JointState joint_state_msg_leg2_joint1;
sensor_msgs::JointState joint_state_msg_leg2_joint2;
sensor_msgs::JointState joint_state_msg_leg2_joint3;
sensor_msgs::JointState joint_state_msg_leg3_joint1;
sensor_msgs::JointState joint_state_msg_leg3_joint2;
sensor_msgs::JointState joint_state_msg_leg3_joint3;
sensor_msgs::JointState joint_state_msg_leg4_joint1;
sensor_msgs::JointState joint_state_msg_leg4_joint2;
sensor_msgs::JointState joint_state_msg_leg4_joint3;


joint_state_msg_leg1_joint1.name.resize(1);
joint_state_msg_leg1_joint1.name[0] = "servo1_joint_1";
joint_state_msg_leg1_joint1.position.resize(1);
joint_state_msg_leg1_joint1.position[0] = 0;
joint_state_msg_leg1_joint1.velocity.resize(1);
joint_state_msg_leg1_joint1.velocity[0] = 0;
joint_state_msg_leg1_joint1.effort.resize(1);
joint_state_msg_leg1_joint1.effort[0] = 0;

joint_state_msg_leg1_joint2.name.resize(1);
joint_state_msg_leg1_joint2.name[0] = "servo2_joint_1";
joint_state_msg_leg1_joint2.position.resize(1);
joint_state_msg_leg1_joint2.position[0] = 0;
joint_state_msg_leg1_joint2.velocity.resize(1);
joint_state_msg_leg1_joint2.velocity[0] = 0;
joint_state_msg_leg1_joint2.effort.resize(1);
joint_state_msg_leg1_joint2.effort[0] = 0;

joint_state_msg_leg1_joint3.name.resize(1);
joint_state_msg_leg1_joint3.name[0] = "servo3_joint_1";
joint_state_msg_leg1_joint3.position.resize(1);
joint_state_msg_leg1_joint3.position[0] = 0;
joint_state_msg_leg1_joint3.velocity.resize(1);
joint_state_msg_leg1_joint3.velocity[0] = 0;
joint_state_msg_leg1_joint3.effort.resize(1);
joint_state_msg_leg1_joint3.effort[0] = 0;

joint_state_msg_leg2_joint1.name.resize(1);
joint_state_msg_leg2_joint1.name[0] = "leg_2_servo1_joint_1";
joint_state_msg_leg2_joint1.position.resize(1);
joint_state_msg_leg2_joint1.position[0] = 0;

joint_state_msg_leg2_joint2.name.resize(1);
joint_state_msg_leg2_joint2.name[0] = "leg_2_servo2_joint_1";
joint_state_msg_leg2_joint2.position.resize(1);
joint_state_msg_leg2_joint2.position[0] = 0;

joint_state_msg_leg2_joint3.name.resize(1);
joint_state_msg_leg2_joint3.name[0] = "leg_2_servo3_joint_1";
joint_state_msg_leg2_joint3.position.resize(1);
joint_state_msg_leg2_joint3.position[0] = 0;

joint_state_msg_leg3_joint1.name.resize(1);
joint_state_msg_leg3_joint1.name[0] = "leg_3_servo1_joint_1";
joint_state_msg_leg3_joint1.position.resize(1);
joint_state_msg_leg3_joint1.position[0] = 0;

joint_state_msg_leg3_joint2.name.resize(1);
joint_state_msg_leg3_joint2.name[0] = "leg_3_servo2_joint_1";
```

```
joint_state_msg_leg3_joint2.position.resize(1);
joint_state_msg_leg3_joint2.position[0] = 0;

joint_state_msg_leg3_joint3.name.resize(1);
joint_state_msg_leg3_joint3.name[0] = "leg_3_servo3_joint_3";
joint_state_msg_leg3_joint3.position.resize(1);
joint_state_msg_leg3_joint3.position[0] = 0;

joint_state_msg_leg4_joint1.name.resize(1);
joint_state_msg_leg4_joint1.name[0] = "leg_4_servo1_joint_1";
joint_state_msg_leg4_joint1.position.resize(1);
joint_state_msg_leg4_joint1.position[0] = 0;

joint_state_msg_leg4_joint2.name.resize(1);
joint_state_msg_leg4_joint2.name[0] = "leg_4_servo2_joint_1";
joint_state_msg_leg4_joint2.position.resize(1);
joint_state_msg_leg4_joint2.position[0] = 0;

joint_state_msg_leg4_joint3.name.resize(1);
joint_state_msg_leg4_joint3.name[0] = "leg_4_servo3_joint_1";
joint_state_msg_leg4_joint3.position.resize(1);
joint_state_msg_leg4_joint3.position[0] = 0;


// joint_state_msg.name.resize(12);

// joint_state_msg.name[0] = "servo1_joint_1";
// joint_state_msg.name[1] = "servo2_joint_1";
// joint_state_msg.name[2] = "servo3_joint_1";
// joint_state_msg.name[3] = "leg_2_servo1_joint_1";
// joint_state_msg.name[4] = "leg_2_servo2_joint_1";
// joint_state_msg.name[5] = "leg_2_servo3_joint_3";
// joint_state_msg.name[6] = "leg_3_servo1_joint_1";
// joint_state_msg.name[7] = "leg_3_servo2_joint_1";
// joint_state_msg.name[8] = "leg_3_servo3_joint_3";
// joint_state_msg.name[9] = "leg_4_servo1_joint_1";
// joint_state_msg.name[10] = "leg_4_servo2_joint_1";
// joint_state_msg.name[11] = "leg_4_servo3_joint_1";




// joint_state_msg.position.resize(12);
// joint_state_msg.position[0] = 0;
// joint_state_msg.position[1] = 0;
// joint_state_msg.position[2] = 0;
// joint_state_msg.position[3] = 0;
// joint_state_msg.position[4] = 0;
// joint_state_msg.position[5] = 0;
// joint_state_msg.position[6] = 0;
// joint_state_msg.position[7] = 0;
// joint_state_msg.position[8] = 0;
// joint_state_msg.position[9] = 0;
// joint_state_msg.position[10] = 0;
// joint_state_msg.position[11] = 0;

float rad_theta_1 =0;
float rad_theta_2 = 0;
float rad_theta_3 = 0;
```

```
int count = 0;
while (ros::ok())
{
  ros::spinOnce();

    for (int i = 1; i < rows.size(); i++)
    {
      ROS_INFO_STREAM("Row " << i << ":");

        for (int h=0;h<size_vector;h++)
        {

        // ROS_INFO("SIZE = %d",size_vector);
        //ROS_INFO("x[%d] = %f, y[%d] = %f,z[%d] = %f",h,xBezier[h],h,yBezier[h],h,zBezier[h]);
        r_1 = sqrt((yBezier[h]*yBezier[h])+(xBezier[h]*xBezier[h]));
        r_2 = zBezier[h] - a_1;
        r_3 = sqrt((r_1*r_1)+(r_2*r_2));
        float phi_1_value = ((a_3*a_3) - (a_2*a_2) - (r_3*r_3)) / (-2*a_2*r_3);
        if (phi_1_value > 1) {
        phi_1_value = 1;
        }
        phi_1 = acos(phi_1_value);
        phi_2 = atan(r_2/r_1);

        theta_2 = ((PI)+phi_1+phi_2)*RAD_TO_DEG;

        float theta_3_value = (-(a_3*a_3)-(a_2*a_2)+(r_3*r_3))/(-2*a_2*a_3);
        if (theta_3_value < -1) {
            theta_3_value = -1;
        }
        theta_3 = acos(theta_3_value)*RAD_TO_DEG;
        theta_1 = atan2(yBezier[h],xBezier[h])*RAD_TO_DEG;

        if (theta_1<0)
        {
          theta_1 = theta_1+360;
        }
        // rad_theta_1 = (theta_1-180);
        // rad_theta_2 = (theta_2-90);
        // rad_theta_3 = theta_3;
        //(theta_1-90)*DEG_TO_RAD
        rad_theta_1 = 0;
        rad_theta_2 = (theta_2+270)*DEG_TO_RAD;
        rad_theta_3 = (180-(theta_3+theta_2-360)+90+90+90+90)*DEG_TO_RAD;
        // float pulse_1 =servo|| rows[i][1]==11_min_pulse+(pulse_factor_1)*(theta_1-180);
        // float pulse_2 =servo2_min_pulse+(pulse_factor_2)*(theta_2-90);
        // float pulse_3 =servo3_min_pulse+(pulse_factor_3)*(theta_3);

        // Striing p1=";";

        // servo1.writeMicroseconds(pulse_1);
        // servo2.writeMicroseconds(pulse_2);
        // servo3.writeMicroseconds(pulse_3);
        // delay(250);
        ROS_INFO("theta1 = %f, theta2 = %f, theta3 = %f", rad_theta_1,rad_theta_2,rad_theta_3);
        // joint_state_msg.position[0] =rad_theta_1;
        // joint_state_msg.position[1] = rad_theta_2;
        // joint_state_msg.position[2] = rad_theta_3;
        if (rows[i][0]==1)
        {
```

```
  float leg1_pulse_1 =red_servo_min_pulse+(red_pulse_factor)*(theta_1-180);
 float leg1_pulse_2 =red_servo_min_pulse+(red_pulse_factor)*(180-(theta_2-90));
 float leg1_pulse_3 =servo_min_pulse+(pulse_factor)*(180-(theta_3+theta_2-270));
 joint_state_msg_leg1_joint1.position[0] = leg1_pulse_1;
 joint_state_msg_leg1_joint2.position[0] = leg1_pulse_2;
 joint_state_msg_leg1_joint3.position[0] = leg1_pulse_3;




 joint_state_msg_leg1_joint1.header.stamp = ros::Time::now();
 pub.publish(joint_state_msg_leg1_joint1);
 joint_state_msg_leg1_joint2.header.stamp = ros::Time::now();
 pub.publish(joint_state_msg_leg1_joint2);
 joint_state_msg_leg1_joint3.header.stamp = ros::Time::now();
 pub.publish(joint_state_msg_leg1_joint3);
 }
 if(rows[i][1]==1)
 {
    float leg2_pulse_1 =servo_min_pulse+(pulse_factor)*(theta_1-180);
     float leg2_pulse_2 =servo_min_pulse+(pulse_factor)*((theta_2-90));
     float leg2_pulse_3 =servo_min_pulse+(pulse_factor)*((theta_3+theta_2-270));
 joint_state_msg_leg2_joint1.position[0] = leg2_pulse_1;
 joint_state_msg_leg2_joint2.position[0] = leg2_pulse_2;
 joint_state_msg_leg2_joint3.position[0] = leg2_pulse_3;




 joint_state_msg_leg2_joint1.header.stamp = ros::Time::now();
 pub.publish(joint_state_msg_leg2_joint1);
 joint_state_msg_leg2_joint2.header.stamp = ros::Time::now();
 pub.publish(joint_state_msg_leg2_joint2);
 joint_state_msg_leg2_joint3.header.stamp = ros::Time::now();
 pub.publish(joint_state_msg_leg2_joint3);
 }
 if(rows[i][2]==1)
 {
   float leg3_pulse_1 =servo_min_pulse+(pulse_factor)*(theta_1-180);
 float leg3_pulse_2 =red_servo_min_pulse+(red_pulse_factor)*((theta_2-90));
 float leg3_pulse_3 =servo_min_pulse+(pulse_factor)*((theta_3+theta_2-270));
 joint_state_msg_leg3_joint1.position[0] = leg3_pulse_1;
 joint_state_msg_leg3_joint2.position[0] = leg3_pulse_2;
 joint_state_msg_leg3_joint3.position[0] = leg3_pulse_3;




 joint_state_msg_leg3_joint1.header.stamp =ros::Time::now();
 pub.publish(joint_state_msg_leg3_joint1);
 joint_state_msg_leg3_joint2.header.stamp = ros::Time::now();
 pub.publish(joint_state_msg_leg3_joint2);
 joint_state_msg_leg3_joint3.header.stamp = ros::Time::now();
 pub.publish(joint_state_msg_leg3_joint3);
 }
 if(rows[i][3]==1)
 {
     float leg4_pulse_1 =red_servo_min_pulse+(red_pulse_factor)*(theta_1-180);
float leg4_pulse_2 =red_servo_min_pulse+(red_pulse_factor)*(180-(theta_2-90));
float leg4_pulse_3 =servo_min_pulse+(pulse_factor)*(180-(theta_3+theta_2-270));
```

```
        joint_state_msg_leg4_joint1.position[0] = leg4_pulse_1;
        joint_state_msg_leg4_joint2.position[0] = leg4_pulse_2;
        joint_state_msg_leg4_joint3.position[0] = leg4_pulse_3;




        joint_state_msg_leg4_joint1.header.stamp = ros::Time::now();
        pub.publish(joint_state_msg_leg4_joint1);
        joint_state_msg_leg4_joint2.header.stamp = ros::Time::now();
        pub.publish(joint_state_msg_leg4_joint2);
        joint_state_msg_leg4_joint3.header.stamp = ros::Time::now();
        pub.publish(joint_state_msg_leg4_joint3);

        }


        }


   }

  loop_rate.sleep();
  for (int i = 0; i < rows.size(); i++)
  {
    rows[i].clear();
  }
  rows.clear();
  }

  // ROS_INFO("YO");
  // ros::spinOnce();
  // ROS_INFO("ME");

return 0;
}
```

Appendix II – Gait Generator Code

```
#include <std_msgs/Float32MultiArray.h>
#include <ros/ros.h>
#include <sensor_msgs/Joy.h>
#include "std_msgs/Int32MultiArray.h"
#include "std_msgs/MultiArrayDimension.h"

int H =3;
int W =4;

std::vector<float> array_gait;
// std_msgs::Int32MultiArray dat;


void joyCallback(const std_msgs::Float32MultiArray::ConstPtr& msg)
{
  // ROS_INFO("Axis: %f", msg->axes.size());
  array_gait.resize(msg->data.size());

  float x = 0;
  for (int i = 0; i < msg->data.size(); i++)
  {
    x=msg->data[i];
    array_gait[i] = x;
  //  ROS_INFO("AxisX: %f", array_gait[i]);

  }

}

int main(int argc, char **argv)
{

  ros::init(argc, argv, "gait_generator");


  ros::NodeHandle n;
  //ros::Publisher inverse_pub = n.advertise<kinematics_legged::bezier>("inverse_kinematics", 1000);
  //kinematics_legged::bezier bezier_msg;
  ros::Publisher pub = n.advertise<std_msgs::Int32MultiArray>("gait_generator", 1000);

  ros::Subscriber sub = n.subscribe("gait_generator_topic", 100, joyCallback);

  ros::Rate loop_rate(31);

  while (ros::ok())
  {

    ros::spinOnce();
    if (!array_gait.empty())

   {
    if (static_cast<int>(array_gait[5]) ==0 && static_cast<int>(array_gait[2])==0 &&
(static_cast<int>(array_gait[1])==1 || static_cast<int>(array_gait[1])==-1 ||
static_cast<int>(array_gait[0])==1 || static_cast<int>(array_gait[0])== -1) )
    {
      ROS_INFO("AxisX: %f", array_gait[1]);
```

```
  // float forward_gait[] =[];
  int array[H][W] = {
    {array_gait[0], array_gait[1], array_gait[2], array_gait[3]},
    {1, 0, 1, 0},
    {0, 1, 0, 1}};
  std_msgs::Int32MultiArray dat;
// fill out message:
  dat.layout.dim.push_back(std_msgs::MultiArrayDimension());
  dat.layout.dim.push_back(std_msgs::MultiArrayDimension());
  dat.layout.dim[0].label = "height";
  dat.layout.dim[1].label = "width";
  dat.layout.dim[0].size = H;
  dat.layout.dim[1].size = W;
  dat.layout.dim[0].stride = H*W;
  dat.layout.dim[1].stride = W;
  dat.layout.data_offset = 0;
  std::vector<int> vec(W*H, 0);
  for (int i=0; i<H; i++)
      for (int j=0; j<W; j++)
          vec[i*W + j] = array[i][j];
  dat.data = vec;
  pub.publish(dat);
  array_gait.clear();

 }
 else if (static_cast<int>(array_gait[2])==1 || static_cast<int>(array_gait[2])==-1) //rotate left/right
 {
  ROS_INFO("AxisX: %f", array_gait[1]);
  // float forward_gait[] =[];
  int array[H][W] = {
    {array_gait[0], array_gait[1], array_gait[2], array_gait[3]},
    {1, 1, 0, 0},
    {0, 0, 1, 1}};
  std_msgs::Int32MultiArray dat;
// fill out message:
  dat.layout.dim.push_back(std_msgs::MultiArrayDimension());
  dat.layout.dim.push_back(std_msgs::MultiArrayDimension());
  dat.layout.dim[0].label = "height";
  dat.layout.dim[1].label = "width";
  dat.layout.dim[0].size = H;
  dat.layout.dim[1].size = W;
  dat.layout.dim[0].stride = H*W;
  dat.layout.dim[1].stride = W;
  dat.layout.data_offset = 0;
  std::vector<int> vec(W*H, 0);
  for (int i=0; i<H; i++)
      for (int j=0; j<W; j++)
          vec[i*W + j] = array[i][j];
  dat.data = vec;
  pub.publish(dat);
  array_gait.clear();

 }
 else
 {
  /* code */
   ROS_INFO("AxisX: %f", array_gait[1]);
  int array[H][W] = {
    {array_gait[0], array_gait[1], array_gait[2], array_gait[3]},
    {1, 1, 1, 1}};
```

```
    std_msgs::Int32MultiArray dat;
   // fill out message:
    dat.layout.dim.push_back(std_msgs::MultiArrayDimension());
    dat.layout.dim.push_back(std_msgs::MultiArrayDimension());
    dat.layout.dim[0].label = "height";
    dat.layout.dim[1].label = "width";
    int H=2;
    int W=4;
    dat.layout.dim[0].size = H; //H
    dat.layout.dim[1].size = W; //W
    dat.layout.dim[0].stride = H*W;
    dat.layout.dim[1].stride = W;
    dat.layout.data_offset = 0;
    std::vector<int> vec(W*H, 0);
    for (int i=0; i<H; i++)
        for (int j=0; j<W; j++)
            vec[i*W + j] = array[i][j];
    dat.data = vec;
    pub.publish(dat);
    array_gait.clear();
   }

  }
      loop_rate.sleep();

  }

  return 0;
}
```

Appendix III – Controller Node:

```
#include <std_msgs/Float32MultiArray.h>
#include <std_msgs/Int32MultiArray.h>
#include <ros/ros.h>
#include <sensor_msgs/Joy.h>


std::vector<int> joy_axes;

std_msgs::Float32MultiArray gait_gen;
void joyCallback(const sensor_msgs::Joy::ConstPtr& msg)
{
 // ROS_INFO("Axis: %f", msg->axes.size());
 int size_vector = 5;
 // joy_axes.resize(msg->axes.size());
 joy_axes.resize(size_vector);
 float x = 0;
 for (int i = 0; i < msg->axes.size(); i++)
 {
   x=msg->axes[i];
   joy_axes[i] = x;
   // ROS_INFO("AxisX: %f", joy_axes[i]);

 }
 joy_axes[4]=msg->buttons[7];

}


int main(int argc, char **argv)
{

 ros::init(argc, argv, "controller_node");


 ros::NodeHandle n;
 //ros::Publisher inverse_pub = n.advertise<kinematics_legged::bezier>("inverse_kinematics", 1000);
 //kinematics_legged::bezier bezier_msg;
 ros::Subscriber sub = n.subscribe("joy", 1000, joyCallback);
 ros::Publisher pub = n.advertise<std_msgs::Float32MultiArray>("gait_generator_topic", 1000);

 ros::Rate loop_rate(31);


 while (ros::ok())
 {
  ros::spinOnce();
  if (!joy_axes.empty())  // Check if the vector is not empty
  {
  //   if (joy_axes[1] ==1)
  // {
  //   ROS_INFO("AxisX: %f", joy_axes[1]);

  //   // ROS_INFO("AxisX: %f", joy_axes[1]);
  //   gait_gen.data.push_back(joy_axes[1]);
  //   pub.publish(gait_gen);

  //   loop_rate.sleep();
  // }
```

```
    // if (joy_axes[1] ==1)

    gait_gen.data.resize(joy_axes.size());
    for(int i=0;i<joy_axes.size();i++)
    {
      gait_gen.data[i]=joy_axes[i];
    }
    pub.publish(gait_gen);

    loop_rate.sleep();

   // ROS_INFO("AxisX: %f", joy_axes[1]);
   // if (joy_axes[1] ==1)
   // {
   //   // ROS_INFO("AxisX: %f", joy_axes[1]);
   //   gait_gen.data.push_back(joy_axes[0]);
   //   pub.publish(gait_gen);

   //   loop_rate.sleep();
   // }
    }
    loop_rate.sleep();
  }
  return 0;
}
```

Appendix IV – Bezier Curve:

```
#include "ros/ros.h"
#include <sstream>
#include "kinematics_legged/bezier.h"
#include <std_msgs/Int32MultiArray.h>
#include <vector>
#include "math.h"
int ROWS = 3;
int COLS = 4;
// std::vector<std::vector<int>> rows;
// rows.resize(1);
std::vector<std::vector<int>> rows;

void gait_generator_call_back(const std_msgs::Int32MultiArray::ConstPtr& msg)
 {
   if (msg->layout.dim[0].size==3)
     {
      ROWS=3;
     }
     else if (msg->layout.dim[0].size==5)
     {
      ROWS=5;
     }
     else if (msg->layout.dim[0].size==2)
     {
      ROWS=2;
     }
   if ((msg->layout.dim[0].size == ROWS  ) && msg->layout.dim[1].size == COLS)
   {
      // Access the matrix elements

     for (int i = 0; i < ROWS; i++)
     {
        std::vector<int> rowArray;
        for (int j = 0; j < COLS; j++)
        {
           int element = msg->data[i * COLS + j];


           rowArray.push_back(element);
        }
        rows.push_back(rowArray);

     }
   }
   else
   {
      ROS_ERROR("Received matrix with incorrect dimensions!");
   }
 }



const int cells=15;
```

```
int factorial(int n) {
   if (n <= 1)
      return 1;
   else
      return n * factorial(n - 1);
}

float Ni(int n, int i) {
   return static_cast<float>(factorial(n)) / (factorial(i) * factorial(n - i));
}

float* basisFunction(int n, int i, float* t) {
   float* J = new float[cells];
   for (int j = 0; j < cells; j++) {
      J[j] = Ni(n, i) * pow(t[j], i) * pow(1 - t[j], n - i);
   }
   return J;
}

int main(int argc, char **argv)
{

  ros::init(argc, argv, "bezier_curve");


  ros::NodeHandle n;
  ros::Publisher inverse_pub = n.advertise<kinematics_legged::bezier>("inverse_kinematics", 1000);
  ros::Subscriber gait_sub = n.subscribe("gait_generator", 1000, gait_generator_call_back);

  kinematics_legged::bezier bezier_msg;



  ros::Rate loop_rate(31);


  while (ros::ok())
  {
    ros::spinOnce();
    if (!rows.empty())
    {
//<-------------------------Bezier Curve------------------------->
      float x[] = {0,0,0,0,0,0,0,0};  //gcs system -x
            float z[] ={0.1,6,6,0.2,-0.2,-5.5,-6,0.1}; //gcs system -z
            float y[] ={-16,-15.5,-15.5,-12,-12,-16,-16,-16}; //gcs sytem y

   if (rows[0][1]== 1|| static_cast<int>(rows[0][1])== -1)
    {

    }
    if (static_cast<int>(rows[0][1])== 1|| static_cast<int>(rows[0][1])== -1)
    {


        if (static_cast<int>(rows[0][1])==1)
        {
           // float newX[] = {0,0,0}; //gcs system -x
           // float newY[] = {-12,-7,-12}; //gcs system -z
           // float newZ[] = {-10,1,10 }; //gcs sytem y
          float newX[] = {0,0,0,0,0,0,0,0};  //gcs system -x
```

```
        float newY[] = {-16,-15.5,-15.5,-12,-12,-16,-16,-16}; //gcs system -z
        float newZ[] = {0.1,6,6,0.2,-0.2,-5.5,-6,0.1};//gcs sytem y
        int size = sizeof(newX) / sizeof(newX[0]);
             for (int i = 0; i <size; i++) {
                x[i] = newX[i];
                y[i] = newY[i];
                z[i] = newZ[i];


             }
      }
      else if (static_cast<int>(rows[0][1])==-1)
      {
           float newX[] = {0,0,0,0,0,0,0,0};  //gcs system -x
        float newY[] = {-16,-15.5,-15.5,-12,-12,-16,-16,-16}; //gcs system -z
        float newZ[] = {0.1,6,6,0.2,-0.2,-5.5,-6,0.1};//gcs sytem y
        int size = sizeof(newX) / sizeof(newX[0]);

             for (int i = 0; i < size ; i++) {
                x[i] = newX[i];
                y[i] = newY[i];
                z[i] = newZ[i];
             }
      }
    int nCPTS = sizeof(x) / sizeof(x[0]);
    int n = nCPTS - 1;

    float t[cells]={0};

    float xBezier[cells]={0};
    float yBezier[cells]={0};
    float zBezier[cells]={0};

//<-------------------------X------------------------->

    bezier_msg.x.resize(cells);
    bezier_msg.y.resize(cells);
    bezier_msg.z.resize(cells);
    std_msgs::Float32 x_value;
    std_msgs::Float32 y_value;
    std_msgs::Float32 z_value;

       for (int j = 0; j < cells; j++)
       {
       t[j] = static_cast<float>(j) / (cells - 1);
    }
    int i = 0;
    for (int k = 0; k < nCPTS; k++) {
       float* b = basisFunction(n, i, t);

       for (int p = 0; p < cells; p++)
       {
       xBezier[p] += b[p] * x[k];
       yBezier[p] += b[p] * y[k];
       zBezier[p] += b[p] * z[k];
       }
       delete[] b;
       i++;
    }
    for (int v = 0; v < cells; v++) {
```

```
        x_value.data = xBezier[v];
       y_value.data = yBezier[v];
       z_value.data = zBezier[v];
       bezier_msg.x[v] = x_value;
       bezier_msg.y[v] = y_value;
       bezier_msg.z[v] = z_value;
    }

    std_msgs::Int32MultiArray dat;
    bezier_msg.paramset.resize(1);
     dat.layout.dim.push_back(std_msgs::MultiArrayDimension());
      dat.layout.dim.push_back(std_msgs::MultiArrayDimension());
      dat.layout.dim[0].label = "height";
      dat.layout.dim[1].label = "width";
      int H = 3;
      int W=4;
      dat.layout.dim[0].size = H;
      dat.layout.dim[1].size = W;
      dat.layout.dim[0].stride = H*W;
      dat.layout.dim[1].stride = W;
      dat.layout.data_offset = 0;
      std::vector<int> vec(W*H, 0);
      for (int i=0; i<H; i++)
          for (int j=0; j<W; j++)
              vec[i*W + j] = rows[i][j];
      dat.data = vec;
      bezier_msg.paramset[0] = dat;
    for(int p=0;p<cells;p++)
      {
        xBezier[p] = 0;
        yBezier[p] = 0;
        zBezier[p] = 0;

      }

    inverse_pub.publish(bezier_msg);

    }
    else{
         float x[] = {0}; //gcs system -x
             float y[] ={-16}; //gcs system -z
             float z[] ={0.1};



    // float z[] ={9,14,9};
    // float x[] =  {-10,1,10};
    // float y[] = {0,0,0};


    float xBezier[cells]={0};
    float yBezier[cells]={0};
    float zBezier[cells]={0};

//<-------------------------X------------------------->

     int nCPTS = sizeof(x) / sizeof(x[0]);

    int n = nCPTS - 1;
```

```
    int i = 0;
    float t[cells];


//<-------------------------X------------------------->

    bezier_msg.x.resize(cells);
    bezier_msg.y.resize(cells);
    bezier_msg.z.resize(cells);
    std_msgs::Float32 x_value;
    std_msgs::Float32 y_value;
    std_msgs::Float32 z_value;

        for (int j = 0; j < cells; j++) {
        t[j] = static_cast<float>(j) / (cells - 1);
    }

    for (int k = 0; k < nCPTS; k++) {
        float* b = basisFunction(n, i, t);

        for (int p = 0; p < cells; p++) {
            xBezier[p] += b[p] * x[k];
            yBezier[p] += b[p] * y[k];
            zBezier[p] += b[p] * z[k];
            x_value.data = xBezier[p];
        y_value.data = yBezier[p];
        z_value.data = zBezier[p];
        bezier_msg.x[p] = x_value;
        bezier_msg.y[p] = y_value;
        bezier_msg.z[p] = z_value;
        }

        delete[] b;
        i++;
    }
    std_msgs::Int32MultiArray dat;
    bezier_msg.paramset.resize(1);
     dat.layout.dim.push_back(std_msgs::MultiArrayDimension());
      dat.layout.dim.push_back(std_msgs::MultiArrayDimension());
      dat.layout.dim[0].label = "height";
      dat.layout.dim[1].label = "width";
      int H = 2;
      int W=4;
      dat.layout.dim[0].size = H;
      dat.layout.dim[1].size = W;
      dat.layout.dim[0].stride = H*W;
      dat.layout.dim[1].stride = W;
      dat.layout.data_offset = 0;
      std::vector<int> vec(W*H, 0);
      for (int i=0; i<H; i++)
          for (int j=0; j<W; j++)
              vec[i*W + j] = rows[i][j];
      dat.data = vec;
      bezier_msg.paramset[0] = dat;
    for(int p=0;p<cells;p++)
      {
        xBezier[p] = 0;
        yBezier[p] = 0;
        zBezier[p] = 0;
```

```
    }

   inverse_pub.publish(bezier_msg);
   }
   //Left/Right
//    else if (rows[0][1]==1 || row[0][1]=-1 || row[0][2]=1|| row[0][2]=-1)
//    {
//    float x[] = {0,0,0}; //gcs system -x
//    float y[] = {-13,-5,-13}; //gcs system -z
//    float z[] = {-10,1,10}; //gcs sytem y
//    int nCPTs=3;
//    int n_o= nCPTs-1;
//    // float z[] ={9,14,9};
//    // float x[] =  {-10,1,10};
//    // float y[] = {0,0,0};
//    int i=0;

//    float xBezier[cells]={0};
//    float yBezier[cells]={0};
//    float zBezier[cells]={0};

// //<-------------------------X------------------------->
//    int n = n_o;
//    bezier_msg.x.resize(cells);
//    bezier_msg.y.resize(cells);
//    bezier_msg.z.resize(cells);
//    std_msgs::Float32 x_value;
//    std_msgs::Float32 y_value;
//    std_msgs::Float32 z_value;
//    for (int j = 0; j < nCPTs; j++)
//    {

//      float* b;
//      b=basisFunction(n, j);
//    //   std::cout<<"----------------------"<<n<<"------n----------"<<std::endl;
//      for(int p=0;p<cells;p++)
//      {


//        xBezier[p] += (b[p] * x[j]);
//        yBezier[p] = (b[p] * y[j])+yBezier[p] ;
//        zBezier[p] += b[p] * z[j];
//        x_value.data = xBezier[p];
//        y_value.data = yBezier[p];
//        z_value.data = zBezier[p];
//        bezier_msg.x[p] = x_value;
//        bezier_msg.y[p] = y_value;
//        bezier_msg.z[p] = z_value;

//      }

//    }
//    if(row[0][1]==-1)
//    {
//        for (int p = cells - 1; p >= 0; p--)
//        {
//            x_value.data = xBezier[cells-1-p];
//            y_value.data = yBezier[cells-1-p];
//            z_value.data = zBezier[cells-1-p];
//            bezier_msg.x[cells-1-p] = x_value;
```

```
//            bezier_msg.y[cells-1-p] = y_value;
//            bezier_msg.z[cells-1-p] = z_value;
//         }
//     }

//     for(int p=0;p<cells;p++)
//       {
//         xBezier[p] = 0;
//         yBezier[p] = 0;
//         zBezier[p] = 0;

//       }

//     inverse_pub.publish(bezier_msg);
//     loop_rate.sleep();
//     }
  }
  loop_rate.sleep();
   for (int i = 0; i < rows.size(); i++)
  {
    rows[i].clear();
  }
  rows.clear();
  }
  return 0;
}


//clearing the row aaray
```

Appendix_V – ROS Arduino Code:

```
#include <Servo.h>

#include <ros.h>
#include <std_msgs/Int32.h>
#include <sensor_msgs/JointState.h>

#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

// called this way, it uses the default address 0x40
Adafruit_PWMServoDriver servo = Adafruit_PWMServoDriver();

ros::NodeHandle node_handle;

// Convert the joint state values to degrees, adjust for the center and write to the servo
float position

// Subscriber Callback to store the jointstate position values in the global variables
void servoControlSubscriberCallbackJointState(const sensor_msgs::JointState& msg) {
   // Check if the joint state messages have the expected sizes


  // Access the joint state values and assign them to variables

  if (msg.name[0] == "servo1_joint_1")
  {
    Serial.print(msg.position[0]);
    servo.writeMicroseconds(0,msg.position[0]);
  }
  else if (msg.name[0] == "servo2_joint_1")
  {

    servo.writeMicroseconds(1,msg.position[0]);

  }
  else if (msg.name[0] == "servo3_joint_1")
  {

    Serial.print(msg.position[0]);


    servo.writeMicroseconds(2,msg.position[0]);

  }

  else if (msg.name[0] == "leg_2_servo1_joint_1")
  {

    servo.writeMicroseconds(4,msg.position[0]);
  }
  else if (msg.name[0] == "leg_2_servo2_joint_1")
  {
    servo.writeMicroseconds(5,msg.position[0]);

  }
  else if (msg.name[0] == "leg_2_servo3_joint_1")
  {
    servo.writeMicroseconds(6,msg.position[0]);
```

```
 }
 else if (msg.name[0] == "leg_3_servo1_joint_1")
 {
      servo.writeMicroseconds(8,msg.position[0]);

 }

  else if (msg.name[0] == "leg_3_servo2_joint_1")
 {
         servo.writeMicroseconds(9,msg.position[0]);

 }
  else if (msg.name[0] == "leg_3_servo3_joint_1")
 {
             servo.writeMicroseconds(10,msg.position[0]);

 }
 else if (msg.name[0] == "leg_4_servo1_joint_1")
 {
   servo.writeMicroseconds(12,msg.position[0] );
 }
  else if (msg.name[0] == "leg_4_servo2_joint_1")
 {
   servo.writeMicroseconds(13,msg.position[0]);

 }
  else if (msg.name[0] == "leg_4_servo3_joint_1")
 {
      servo.writeMicroseconds(13,msg.position[0]);

 }

}
ros::Subscriber<sensor_msgs::JointState>                    servo_control_subscriber_joint_state("joint_states",
&servoControlSubscriberCallbackJointState);

void setup()
{
  // Initial the servo motor connections and initialize them at home position
  // for (unsigned int i = 0; i < 3; i++)
  // {
  //   robot_servos[i].attach(servo_pins[i]);
  //   robot_servos[i].write(mid_positions[i]);
  //   SERVO_CURRENT_POSITIONS[i] = mid_positions[i];
  // }

  // Set the communication BaudRate and start the node
  Serial.begin(9600);
  node_handle.getHardware()->setBaud(9600);
  node_handle.initNode();
  node_handle.subscribe(servo_control_subscriber_joint_state);
}

void loop() {
  // Keep calling the spinOnce() method in this infinite loop to stay tightly coupled with the ROS Serial
  node_handle.spinOnce();
  delay(1);
}
```

Appendix_VI – Vendors Info:

1. City Bearing Centre Babu Bazar, Saddar Rawalpindi
   Contact# 0333-5135436

2. New Bearing Centre Babu Bazaar Rawalpindi
   Contact# 0333-5139558

3. Electrobes Online
   Shop#23, Ground Floor, near Daewoo Fastex, Al-Ghaffar Mall, G-11 Markaz G 11 Markaz G-11, Islamabad, Islamabad Capital Territory
   Contact # 0337-000766

4. Digilog, 13Th Regal chowk, Shahrah-e-Quaid-e-Azam, Mozang Chungi, Lahore, Punjab 54000
   Contact # 0312-4002221

5. Nawaz Hardware Store Gawal Mandi Rawalpindi

6. Electronics World College Rd Rawalpindi

7. Smart hobby, Faisalabad, Punjab 38000
   Contact # 0334 5307120