

**Design and Development of
Quadruped Robot**

A Final Year Project Report

Presented to

SCHOOL OF MECHANICAL & MANUFACTURING ENGINEERING

Department of Mechanical Engineering

NUST

ISLAMABAD, PAKISTAN

In Partial Fulfillment
of the Requirements for the Degree of
Bachelors of Mechanical Engineering

by

Hinata Kashif

Jawad Akbar

June 2024

EXAMINATION COMMITTEE

We hereby recommend that the final year project report prepared under our supervision by:

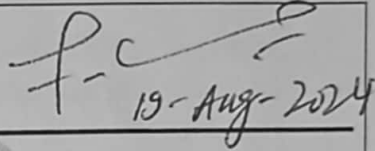
Hinata Kashif


337998

Jawad Akbar

332649

Titled: "Design and Development of Quadruped Robot" be accepted in partial fulfillment of the requirements for the award of Bachelor of Engineering: Mechanical Engineering degree with grade B-1

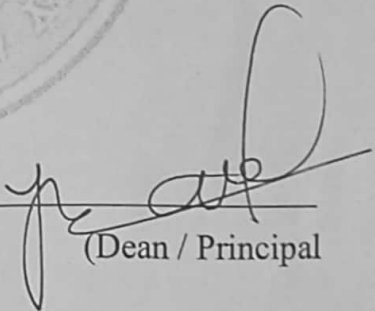
Supervisor: Dr. Khawaja Iqbal, Assistant Professor School of Mechanical and Manufacturing Engineering	 19-Aug-2024 Dated:
---	---


(Head of Department)

21/8/2024
(Date)

COUNTERSIGNED

Dated: 21/8/24


(Dean / Principal)

ABSTRACT

This report outlines the complete journey of designing and developing a quadruped robot, covering mechanical design, kinematic and dynamic analysis, control system development, and prototype testing. The project begins with an extensive literature review highlighting the evolution of quadrupedal robotics, emphasizing innovations in leg design, articulated joints, and sensor integration. The methodology section discusses the kinematic analysis, gait strategy, MATLAB simulations, and control algorithms implemented in the robot.

A kinematic analysis explores forward and inverse kinematics, crucial for the robot's gait and stability. The team used finite element analysis (FEA) to validate their choice of leg material, Polylactic Acid (PLA), ensuring it met stress and deflection criteria for durability. The control system employs PID controllers and Model Predictive Control, focusing on balancing the robot's locomotion and maintaining stability.

The results and discussions include detailed insights into the microcontroller, actuator, battery, and IMU selection for precise control. MATLAB Simulink simulations validated the design approach, leading to refined control strategies and improved performance. The prototype testing demonstrated accurate inverse kinematics through its servo calibration and foot placement, with ongoing adjustments for stable walking and overcoming noise in PWM signals. The integration of a PWM driver improved signal integrity, leading to smoother robot movement.

The conclusion emphasizes the project's achievements in gait analysis, mechanical design, and control systems. Recommendations include exploring new materials for enhanced durability, integrating learning-based strategies, and refining energy storage. Future work focuses on terrain testing, multi-robot communication, and user interface development for broader applicability.

ACKNOWLEDGEMENTS

We are profoundly grateful to our parents, whose unwavering encouragement and support have been the cornerstone of our journey throughout this project.

Our sincere gratitude also goes to our project advisor, Dr. Khawaja Fahad Iqbal, whose expertise, understanding, and patience added considerably to our experience.

Additionally, we would like to thank our classmates for not only giving us friendship but also providing constructive criticism in fulfilling our project ideas. A special thanks to Saad Jameel from the Department of Robotics and AI, whose assistance was vital and hugely appreciated. Finally, our thanks go to Lincoln Corner at NUST Central Library for providing us with the financial and material support that enabled us to use the 3D printer, making this project possible.

CONTENTS

Abstract.....	ii
Acknowledgements	iii
Originality Report	1
List of Tables:.....	2
List of Figures:	2
Abbreviations	5
CHAPTER 1: INTRODUCTION	6
Motivation.....	6
Problem We are Solving.....	6
Objective	7
CHAPTER 2: LITERATURE REVIEW	8
History.....	8
Leg Design.....	10
Kinematic Analysis	10
Forward Kinematics (FK).....	11
Inverse Kinematics (IK).....	11
Denavit-Hartenberg (D-H) Convention	11
Jacobian Matrix.....	12
Dynamic Analysis.....	13

The Lagrangian mechanics	13
Newton Euler Method.....	14
Locomotion and Gait Analysis.....	16
Foundational Research and Innovations in Quadruped Robot Locomotion	
.....	16
Dynamic Stability and Adaptability in Legged Robots.....	17
Inverted Pendulum Model.....	18
Raibert's Strategy	19
Control Strategies and Algorithms.....	21
PID Controller.....	21
Other Controllers	21
CHAPTER 3: Methodology	23
Kinematics	23
Gaits and Algorithm.....	25
Matlab Simulation.....	27
Finite Element analysis:	31
CAD:.....	34
Assembly:	38
Self-Balancing Algorithm:.....	39

CHAPTER 4: RESULTS and DISCUSSIONS.....	45
Microcontroller, Actuator, Battery and IMU selection.....	41
Kinematics equations	45
Forward Kinematics.....	45
Inverse Kinematics.....	45
FEA Results	46
Trajectory Generation	48
PID controller.....	49
Simulation: PID controller	50
Prototype Testing	51
CHAPTER 5: CONCLUSION AND RECOMMENDATION.....	53
REFERENCES	55
APPENDICES.....	67
Appendix 1: MATLAB Controller Code (Trot Gait)	67
Appendix 2: MATLAB Variable Initialization:.....	73
Appendix 3: Arduino Code (Balancing)	75
3.1 Maincode.....	75
3.2 Structures.h	79
3.3 functions.h.....	81
3.4 Classes.h.....	81

3.5 PID.cpp	82
3.6 IK.cpp.....	83
3.7 constructBody.cpp	87
3.8 constructLeg	87

ORIGINALITY REPORT

Hinata Kashif - FYP Report.docx

ORIGINALITY REPORT

11 %	8 %	8 %	3 %
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.dtic.mil Internet Source	1 %
2	archive.org Internet Source	1 %
3	www.coursehero.com Internet Source	1 %
4	forum.arduino.cc Internet Source	<1 %
5	"Advances in Mechanism and Machine Science", Springer Science and Business Media LLC, 2019 Publication	<1 %
6	www.phoenixlidar.com Internet Source	<1 %
7	Khan, Mahmud Safat. "Modelling and Simulation of Small Scale Fixed-Wing Autonomous Aerial Vehicles", Sheffield Hallam University (United Kingdom), 2023 Publication	<1 %

LIST OF TABLES:

Table 1: Properties of Material	32
Table 2: Parameters of Robot Dimensions	44
Table 3: FEA Results	48

LIST OF FIGURES:

Figure 1: Jacobian Matrix Format.....	12
Figure 2: Inverse Pendulum Working.....	18
Figure 3: Inverse Pendulum Mathematical Formulation	19
Figure 4: Reference Frame assignment and Link definition.....	24
Figure 5: Inverse Kinematics - 2.....	Error! Bookmark not defined.
Figure 6: Inverse Kinematics - 1.....	Error! Bookmark not defined.
Figure 8: Hip Actuation Mechanism.....	37
Figure 9: Leg Modeling in Simulink	29
Figure 10: Thigh Link model in COMSOL	33
Figure 11: Thigh link Meshing in COMSOL	33
Figure 12: Calf Link Model in COMSOL	33

Figure 13: Calf Link Meshing in COMSOL.....	34
Figure 14: Hip Assembly.....	34
Figure 15: Leg Assembly.....	35
Figure 16: Robot Body.....	36
Figure 17: Robot Body (Top View).....	36
Figure 18: Hip Actuation Mechanism.....	37
Figure 19: Complete Assembly.....	37
Figure 20: Arduino Mega.....	41
Figure 21: DS3218 - 20KG Digital Servo.....	42
Figure 22: MPU-6050.....	42
Figure 23: 3S 2800 mAh LiPo Battery.....	43
Figure 24: Simulink model of a single leg.....	46
Figure 25: FEA Analysis – Thigh Link – Von Mises.....	46
Figure 26: FEA Analysis – Thigh Link – Total Displacement.....	47
Figure 27: FEA Analysis – Calf link – Von Mises Stress.....	47
Figure 28: FEA Analysis - Calf link – Total Displacement.....	47
Figure 29: Foot Trajectory.....	49

Figure 30: Stance Hip Trajectory.....	49
Figure 31: Simulink Multibody Simulation - Trot Gait (Sideview).....	50
Figure 32: Simulink Multibody Simulation - Trot Gait (Isometric View).....	51
Figure 33: Stance Position under self-weight with no support.....	51
Figure 34: Step motion using IK (a) : Upward motion.....	52
Figure 35: Step motion using IK (b) :Forward motion.....	52
Figure 36: Step motion using IK (c) : Step complete.....	52

ABBREVIATIONS

AI	Artificial Intelligence
ASME	American Society of Mechanical Engineers
CAD	Computer Aided Design
CPG	Central Pattern Generators
DOF	Degrees Of Freedom
FEA	Finite Element Analysis
FK	Forwards Kinematics
FL	Forward Left
FR	Forward Right
IEEE	Institute of Electrical and Electronics Engineers
IK	Inverse Kinematics
IMU	Inertial Measurement Unit
MPC	Model Predictive Control
PD	Proportional Derivative
PID	Proportional Integral Derivative
PLA	Polylactic Acid
PWM	Pulse Width Modulation
WBC	Whole Body Control
ZMP	Zero Moment Point

CHAPTER 1: INTRODUCTION

Motivation

The advent of robotics has ushered in a transformative era in technology, engineering, and automation, marked by the aspiration to transcend human limitations and venture into realms hitherto unattainable. This project is born out of an acute awareness of the burgeoning need for advanced robotic systems capable of performing tasks in environments that are beyond human reach, either due to inherent dangers or physical inaccessibility. The motivation behind this endeavor is to harness the potential of robotics to foster innovations that significantly reduce human exposure to hazardous conditions, enhance operational efficiency, and open new avenues for exploration and development.

The concept of quadrupedal robots has emerged as a beacon of versatility and resilience in the robotics landscape. These robots, inspired by the locomotion of animals, offer a dynamic solution [2] to the challenges of navigating through terrains that would otherwise be insurmountable for traditional wheeled or tracked robots. Their ability to adapt to various surfaces, from the rugged outcrops of disaster-hit zones to the uneven grounds of natural landscapes, presents a compelling case for their development and deployment. The pioneering work of companies such as Boston Dynamics and Ghost Robotics in creating quadruped robots that can carry payloads, scout dangerous areas, and perform an array of tasks autonomously, serves as both inspiration and a benchmark for this project. These industrially applicable solutions not only showcase the technological prowess achieved thus far but also highlight the immense potential for quadruped robots in revolutionizing sectors ranging from disaster management and defense to agriculture and beyond.

However, the journey towards realizing such advanced robotic systems is fraught with challenges. The primary concern addressed by this project revolves around the current limitations faced by existing robotic models in navigating uneven and complex terrains [27, 28]. The rugged or unpredictable nature of such landscapes poses significant obstacles to the operational viability of conventional robots, thereby limiting their effectiveness in critical applications such as disaster response, environmental monitoring, and exploration. Moreover, the high cocompliantsts associated with developing and manufacturing sophisticated robotic systems further exacerbate the issue, putting these technological marvels out of reach for many potential applications, particularly in resource-limited settings.

Problem We are Solving.

Considering these challenges, this project aims to conceptualize, design, and develop an affordable, autonomous quadruped robot capable of expertly navigating a multitude of

terrains. This endeavor seeks to combine the agility and adaptability inherent in quadrupedal locomotion with cutting-edge design and control mechanisms, thereby producing a robot that is not only cost-effective but also robust enough to operate autonomously in diverse environments. The ultimate goal is to enhance the capabilities of disaster response teams, provide viable solutions for tasks that are risky or impractical for humans, and pave the way for future innovations in robotic mobility and autonomy.

Objective

To achieve these ambitious goals, the project is structured around a series of key deliverables, each focusing on a distinct aspect of the robot's development. The first deliverable concentrates on the mechanical design of the leg mechanism and its actuation systems, laying the groundwork for the robot's movement and interaction with its environment. Following this, the second deliverable delves into the kinematic and dynamic analysis of the robot [2], ensuring that its movements are not only precise but also optimized for energy efficiency and adaptability. The third deliverable transitions the project from theoretical models to tangible reality through the manufacture, assembly, and exhaustive testing of the prototype. This phase is critical for identifying and addressing any unforeseen challenges that may arise, thereby refining the robot's design and functionality. Lastly, the fourth deliverable encompasses the integration of a sophisticated control system and the comprehensive evaluation of the robot's performance across a range of scenarios. This final stage is instrumental in validating the project's success and setting the stage for further enhancements and applications.

Through these deliverables, the project aims to forge a path towards a new generation of robotic systems that are not only technologically advanced but also accessible and practical for a wide array of applications. By addressing the critical challenges of terrain adaptability and cost-effectiveness, this project aspires to make a significant contribution to the field of robotics, pushing the boundaries of what is possible and opening up new horizons for exploration, innovation, and impact.

CHAPTER 2: LITERATURE REVIEW

History

The evolution of quadruped robotics is a testament to the field's relentless pursuit of creating machines capable of navigating the complexities of real-world environments with agility and precision. From Marc Raibert's initial [3] experiments that introduced the foundational principles of dynamic balance and legged locomotion, the journey of quadruped development has been marked by significant milestones.

Early in this journey, the focus was on developing reflexive responses to slipping in bipedal running robots [1, 3], alongside the creation of bounding gaits in quadrupeds such as the Scamper [4] and Patrush robots [5,6]. These robots, leveraging sophisticated control schemes, laid the groundwork for advanced locomotion strategies. Notably, Patrush's control was inspired by neural oscillator theories, emphasising the role of bio-inspired mechanisms in robotic movement which also underlies the control of the simulated planar biped of Taga et al [19].

The exploration of adaptive dynamic walking techniques on irregular [6] and natural terrains [7] brought forward innovations in control systems based on Central Pattern Generators (CPGs) [9] and compliance strategies [8], significantly enhancing terrain adaptability[10]. This period also witnessed the emergence of robots like Boston Dynamics' BigDog [11, 12, 13] and LittleDog [14, 15, 16, 17], which showcased autonomous navigation and advanced terrain adaptability through bio-inspired leg compliance and sophisticated control strategies.

The introduction of the HyQ [18] robot marked a leap in actuation methods, incorporating torque-controlled hydraulic and electrical actuators. This era also saw the refinement of control methodologies, including PID control [20] for precise actuation and fuzzy controllers[21] for managing uncertain environments, alongside advancements in detection technologies for improved environmental interaction [22, 23, 24]

Modern quadrupeds feature highly advanced mechanical designs that prioritise efficiency, durability, and adaptability. For instance, MIT's Cheetah [43] robot incorporates lightweight, high-strength materials and novel actuation systems that allow for rapid, energy-efficient movement. Its design emphasises minimising inertia, enabling the robot

to perform high-speed manoeuvres and jumps with remarkable agility. Similarly, the ANYmal [26, 42] robot utilises a modular design, allowing for easy customization and repair, which is crucial for its deployment in challenging environments like industrial inspection sites.

The control strategies employed by modern quadrupeds have evolved significantly, incorporating advanced algorithms like Model Predictive Control (MPC) [45], Whole-Body Control (WBC) [44, 45], and learning-based methods. MPC, used by robots like ANYmal, allows for real-time planning and adaptation to terrain, optimising the robot's movements to ensure stability and efficiency. WBC, exemplified by the Mini-Cheetah, integrates the control of all joints and limbs, enabling the execution of complex manoeuvres like backflips and dynamic recovery from falls. Learning-based methods, including reinforcement learning, have been explored in the context of these robots to enhance their adaptability and performance in unpredictable environments.

Software development [46, 48] for modern quadrupeds leverages sophisticated simulation [2s] environments, real-time operating systems, and modular, open-source frameworks. This ecosystem allows for rapid prototyping, testing, and deployment of control algorithms, sensor fusion strategies, and autonomy modules. The use of simulation not only accelerates the development process but also enables the safe exploration of the robots' capabilities in virtual environments that mimic real-world challenges.

Research on modern quadrupeds has focused on improving locomotion efficiency, autonomy, and interaction with humans and the environment. Notable advancements include the development of energy-efficient gaits, autonomous navigation capabilities in unstructured terrains, and the integration of advanced perception systems for environment mapping and obstacle avoidance. These advancements have been validated in real-world applications, from search and rescue missions and industrial inspection to entertainment and educational purposes

This rich history of quadruped development showcases the field's evolution from foundational experiments to the sophisticated, adaptable machines of today, capable of seamlessly navigating our complex world.

Leg Design

The evolution of quadrupeds in leg design perspective has witnessed significant advancements over time, reflecting a journey of innovation driven by both biological inspiration and engineering ingenuity. Initially, early quadrupeds often emulated simplistic leg structures, typically featuring rigid designs with limited degrees of freedom. However, as research progressed, there was a notable shift towards more biomimetic designs, drawing inspiration from the intricate biomechanics of animals to enhance agility, stability, and adaptability.

1. **Articulated Joints:** Unlike earlier rigid structures, modern quadruped legs often incorporate multiple articulated joints, providing greater flexibility and enabling a wider range of motion. This increased degree of freedom allows for more dynamic and adaptive locomotion, essential for navigating complex and uneven terrains.
2. **Compliant Mechanisms:** Many contemporary leg designs integrate compliant mechanisms, such as springs or pneumatic actuators, to enhance shock absorption and energy efficiency. These compliant elements mimic the elasticity and damping properties observed in biological limbs, improving stability and reducing impact forces during locomotion.[\[29\]](#)
3. **Modular Construction:** Modular leg designs have gained popularity due to their versatility and ease of maintenance. By utilising interchangeable components, such as modular joints or actuators, designers can tailor leg configurations to specific tasks or environments, facilitating rapid prototyping and experimentation.
4. **Sensor Integration:** Advanced sensor technologies, including force sensors, encoders, and inertial measurement units (IMUs), are often integrated into quadruped legs to provide real-time feedback on terrain conditions, leg position, and ground contact forces. This sensory feedback enables precise control and adaptive responses, enhancing stability and performance in dynamic environments.
5. **Optimised Kinematics:** Modern leg designs leverage sophisticated kinematic algorithms and optimization techniques to achieve efficient and agile locomotion. By carefully tuning parameters such as leg length, joint angles, and trajectory profiles, designers can optimise gait patterns for speed, endurance, or terrain traversal, maximising overall performance.

Kinematic Analysis

Kinematic analysis in robotics is the study of motion without regard to the forces that cause it. It involves understanding how the joints and links of a robot move to achieve desired

positions, orientations, and paths of the robot's end-effector or other points of interest. The kinematic analysis is fundamental for designing robotic systems, programming their movements, and ensuring they perform tasks accurately and efficiently. The primary approaches in kinematic analysis include Forward and Inverse kinematics. [30][32[]

Forward Kinematics (FK)

This approach determines the position and orientation of the robot's end-effector given the set of joint parameters (angles for revolute joints and distances for prismatic joints). FK involves the sequential application of transformations from the robot's base to its end-effector, using the kinematic chain of the robot. It's relatively straightforward since it directly computes the end-effector's pose from known joint angles through a series of matrix multiplications.

Forward kinematics calculates the position and orientation of the end-effector of a robot given the joint parameters. In a robotic manipulator with n joints, the forward kinematics equations can be represented as a series of homogeneous transformation matrices, typically denoted as T .

$$T_0^n = T_1^0 T_2^1 \dots T_n^{n-1}$$

Where:

- T_i^{i-1} represents the transformation matrix from frame $i-1$ to frame i .
- T_0^n represents the transformation from the base frame to the end-effector frame.

Inverse Kinematics (IK)

IK is the process of calculating the joint parameters needed to place the robot's end-effector at a desired position and orientation. Unlike FK, IK can be significantly more complex due to the possibility of multiple solutions, or sometimes no solution, and often requires solving a set of nonlinear equations. IK is crucial for planning and control [32] in robotics, especially when the robot must interact with objects or environments in precise ways.

Denavit-Hartenberg (D-H) Convention

The D-H Convention [33, 36] is a systematic method to describe the geometry and kinematics of a robot arm. It simplifies the representation of the robot by assigning four

parameters to each link: two for its dimensions (link length and link twist) and two for its position (joint angle and joint distance). These parameters are used to construct transformation matrices that define the spatial relationship between adjacent links, facilitating the calculation of FK and IK solutions:

1. **Link Length (a):** The distance between the axes of adjacent joints along the common normal.
2. **Link Twist (α):** The angle about the common normal from the old z-axis to the new z-axis.
3. **Joint Angle (θ):** The angle about the previous z-axis, from the old x-axis to the new x-axis.
4. **Joint Distance (d):** The distance along the previous z-axis to the point where the common normal intersects the old z-axis.

Jacobian Matrix

The Jacobian matrix is a fundamental tool in kinematic analysis that relates the velocities of the robot's joints to the velocity of the end-effector in both linear and angular terms. It is essential for understanding how variations in joint speeds affect the movement of the end-effector, and is used in controlling the robot's motion, especially for tasks requiring speed and precision. The Jacobian also plays a key role in dynamics analysis, force control, and dealing with singularities in robotic manipulators. For a robot with n joints, the Jacobian matrix can be represented as

$$J = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \dots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \dots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \dots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Figure 1: Jacobian Matrix Format

Here, T is given by $T(x_1, x_2, \dots, x_n) = (y_1, y_2, \dots, y_m)$.

The Jacobian matrix, derived from the transformation matrix, establishes the connection between joint velocity and end-effector velocities. It comprises three elements for positional velocity and three for rotational velocity.

$$V = J \times \dot{\theta}$$

To relate joint accelerations to end-effector accelerations, we utilize the velocity equation [38, 39]. This equation encompasses three translational and three rotational components.

$$A = J\ddot{\theta} + \dot{J}\dot{\theta}$$

Dynamic Analysis

The Lagrangian mechanics

The Lagrangian mechanics approach is predicated on the principle of stationary action, with the Lagrangian (L) representing the difference between the kinetic energy (T) and potential energy (V) of a system:

$$L = T - V$$

To derive the equations of motion, the Lagrange equation is applied for each degree of freedom:

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}_i} \right) - \frac{\partial L}{\partial q_i} = Q_i$$

where:

- q_i are the generalized coordinates, representing the configuration of the system,
- \dot{q}_i represent the velocities (time derivatives of q_i),
- Q_i denotes the generalized forces acting on the system.

The kinetic energy (T) is often expressed as a quadratic form of the velocities, while the potential energy (V) depends on the configuration of the system. The mass matrix $M(q)$, appearing in the kinetic energy term, is symmetric and positive definite, reflecting the inertial properties of the system. The equations can be further elaborated in matrix form:

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

where:

- $M(q)$ is the mass matrix, dependent on the generalized coordinates,
- $C(q, \dot{q})$ represents the Coriolis and centrifugal forces, which are velocity-dependent,
- $G(q)$ is the gravity force vector, a function of the configuration,
- Q is the generalized force vector, including external forces and torques.

This approach is particularly powerful for systems with complex interactions and constraints, offering a systematic way to derive the dynamics of multi-body systems with multiple degrees of freedom.

Advantages

- Provides a systematic and elegant approach for deriving equations of motion.
- Utilizes generalized coordinates, which can simplify the representation of complex systems.
- Suitable for systems with many degrees of freedom.
- Offers clear physical insights into the system dynamics.

Newton Euler Method

The Newton-Euler approach combines Newton's second laws of motion for translational dynamics with Euler's rotation equations for rotational dynamics, applied to each rigid body in a robotic mechanism. For linear motion, Newton's second law is given by:

$$F = ma$$

where F is the total external force, m is the mass, and a is the linear acceleration. For rotational motion, Euler's equations can be expressed as:

$$\tau = I\alpha + \omega \times (I\omega)$$

where:

- τ is the total external torque acting on the body,
- I is the moment of inertia matrix (which may also depend on the configuration for complex systems),
- α is the angular acceleration,
- ω is the angular velocity vector, and
- $\omega \times (I\omega)$ represents the gyroscopic term.

Advantages

- Particularly suitable for serial-link manipulators and articulated systems.
- Provides a recursive algorithm (the recursive Newton-Euler algorithm) that can be computationally efficient for systems with a tree-like structure.
- Suitable for real-time control applications where computational speed is crucial.

The Newton-Euler method systematically applies these principles to each link of a robot, starting from the base and moving towards the end-effector (forward recursion) to calculate velocities and accelerations, and then proceeding in the reverse direction (backward recursion) to compute forces and torques. This method is particularly useful for deriving explicit dynamic equations and for real-time control applications, as it can be efficiently

implemented and provides a clear physical interpretation of the forces and torques involved.

In robotics, particularly among quadruped and biped robots, the choice of dynamic analysis method—ranging from Lagrangian mechanics, Newton-Euler methods, to a combination of both—is pivotal for modelling and controlling their complex movements [37]. For instance, the MIT Cheetah [34] and ETH Zurich's StarlETH and ANYmal extensively employ Lagrangian mechanics to optimize energy efficiency and ensure dynamic stability, enabling high-speed maneuvers and adaptive locomotion across challenging terrains. On the other hand, robots like Boston Dynamics' BigDog and the bipedal hyq primarily utilize Newton-Euler methods for precise real-time control of forces and torques, critical for achieving stability and agility on rough terrains. Spot, also by Boston Dynamics, along with IIT's HyQ and HyQ2Max, demonstrates the effectiveness of combining both approaches, leveraging the advantages of Lagrangian mechanics for planning and Newton-Euler methods for execution, ensuring agility, efficiency, and stability in dynamic environments. This strategic selection and integration of dynamic analysis methods facilitate the development of robots capable of navigating the physical world with an unprecedented level of sophistication.

Justification for using Lagrange Method for our robot

- The Lagrange method is well-suited for systems with relatively simple mechanical structures and a limited number of degrees of freedom.
- Lagrange's equations of motion are derived from the principle of least action, which inherently captures the underlying physics of the system. This makes the Lagrange method particularly suitable when you seek a physically intuitive model of your robot's dynamics.
- The Newton-Euler method, particularly its recursive algorithm, is known for its computational efficiency, making it suitable for real-time control applications or simulations where fast computation of dynamic responses is crucial. If your project evolves to involve more complex robot designs or if computational efficiency becomes a significant concern, you may consider transitioning to the Newton-Euler method or other methods optimized for such applications.

Locomotion and Gait Analysis

Locomotion and Gait Analysis in quadruped robots is essential for designing robots capable of efficiently navigating various terrains. It involves studying different movement patterns—such as walking, trotting, galloping, and bounding—to determine their suitability for different speeds and environmental challenges. Walking gaits prioritise static stability, ensuring the robot remains balanced over slippery or complex terrains with minimal energy use. Faster gaits like trotting and galloping rely on dynamic stability, where balance is maintained through motion and speed, suitable for smoother terrains requiring quick navigation. Energy efficiency is crucial across all gaits, with strategies like the Spring-Loaded Inverted Pendulum (SLIP) model [46] being employed to conserve energy by mimicking natural muscle elasticity. Speed optimization entails selecting the appropriate gait for the task and terrain, using advanced control algorithms and sensory feedback to adapt the robot's movement in real-time. This analysis underpins the development of quadruped robots that balance stability, energy use, and speed to tackle a wide range of operational environments effectively.

Foundational Research and Innovations in Quadruped Robot Locomotion

Raibert's initial research [3] focused on applying balance and dynamic stability principles to create terrain-navigating vehicles, starting with simplified models like one-legged hoppers. This laid the groundwork for future legged robotics innovations. A key milestone was the development of a 3D One-Legged Hopping Machine, proving that simple control algorithms could manage balance and dynamic locomotion in legged systems. This foundation was crucial for advancing to more complex systems such as quadrupeds. Research on the "planar dog" demonstrated the simulation of distinct gaits using decomposed control algorithms, a vital step towards applying these strategies to multi-legged robots and emphasising the need for leg coordination. The practical creation of a dynamic quadruped robot represented a significant advancement, showing the potential for legged robots to agilely navigate complex terrains.

The concept that mechanical systems' natural oscillation modes can influence gait selection and transition suggested a reduced need for explicit neural control [35]. Innovative methods allowing robots to change direction using lateral foot placement and hip torques marked a major improvement in maneuverability and efficiency. Raibert's work [3] culminated in exploring ways to refine control algorithms and foot placement strategies, underlining the importance of experimental validation and theoretical development for enhancing dynamic legged locomotion.

His notable study [3] on quadruped robot running gaits—trotting, pacing, and bounding—managed by a unified control algorithm through a virtual biped concept, enabling adaptation with minimal adjustments. It bypasses the complexity of animal biomechanics, utilizing digital control for robot movement, emphasizing simple gaits involving leg pairs in alternation. Key aspects include propelling the body at desired velocities and directions while maintaining posture and vertical stability. The control strategy involves adjusting foot positioning for speed, using hip actuators for posture, and managing vertical motion with hydraulic leg adjustments. Implementation includes virtual leg coordination, yaw control, and state machine sequencing for gait transitions, noting the absence of clear gait selection criteria, and suggesting factors like energy efficiency and mechanical effectiveness.

Dynamic Stability and Adaptability in Legged Robots

Dynamic stability in legged robots is crucial for ensuring that robots can navigate a variety of terrains while maintaining balance, adjusting to disturbances, and performing tasks efficiently. Several models are used to analyse and enhance dynamic stability:

1. The **Single Leg Hopper Model**, foundational to the dynamics of vertical hopping and attributed to Marc Raibert, is exemplified by MIT Leg Lab's one-legged robots, showcasing controlled hopping and stability. [3]
2. The **SLIP** model, emphasizing leg compliance and body inertia for running and hopping, informs the MIT Cheetah's energy-efficient design and control across terrains.
3. The **Zero Moment Point (ZMP)** model [40], crucial for dynamic stability in robots like Honda's ASIMO, identifies the ground point where the net moment from gravity and inertia is zero, guiding balance maintenance during movement.
4. **Dynamic Walking Using Limit Cycles** leverages closed trajectories in the robot's state space for stable, natural walking patterns, as exemplified by RHex's robust navigation across challenging terrains.
5. The **Linear Inverted Pendulum Model (LIPM)** simplifies a robot to a point mass atop an inverted pendulum for CoM control, used by Toyota's Human Support Robot (HSR) for stable navigation and walking phase transitions.
6. **Hybrid dynamic** models integrate discrete and continuous dynamics to capture the complexities of legged locomotion, enabling robots like Boston Dynamics' Spot and Atlas to achieve stable, adaptive movement across varied terrains.

7. **Whole-Body Dynamic Control** optimizes forces and torques across a robot's joints for stable, efficient motion, exemplified by ANYmal's versatile operations in complex environments.

Inverted Pendulum Model

A classic problem in dynamics and control theory, the inverted pendulum model is used to demonstrate and study the principles of equilibrium, stability, and control in systems that are inherently unstable. It consists of a pendulum with its mass above its pivot point, requiring active control to maintain its upright position. This model is analogous to many real-world applications, including Atlas and humanoid robots, where maintaining balance is crucial. The simplicity yet challenging nature of the inverted pendulum makes it a staple in control systems education and research, serving as a benchmark for testing control strategies.

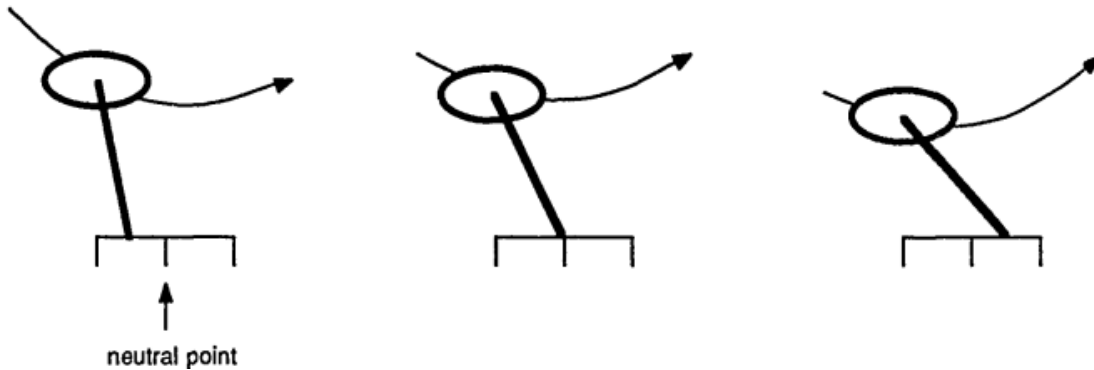


Figure 2: Inverse Pendulum Working

Applying Forward Speed Method inverted pendulum model to obtain forward position of foot at the touchdown:

$$x_{F,d} = \frac{T_s \dot{x}}{2} + k_{\dot{x}}(\dot{x}_f - \dot{x}_d)$$

Where,

T_s , is expected duration of the next stance phase,

\dot{x} , is the expected forward speed during the stance phase,

\dot{x}_f is the present forward speed (during flight),

\dot{x}_d is the desired forward speed for the next flight phase, and

$k_{\dot{x}}$, is an empirically determined gain.

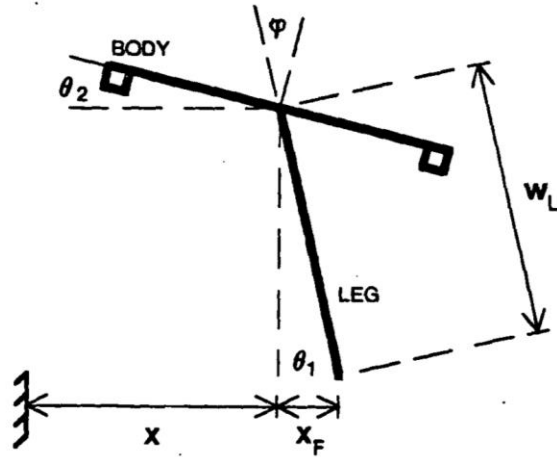


Figure 3: Inverse Pendulum Mathematical Formulation

Raibert's Strategy

Developed by Marc Raibert in the 1980s as part of his pioneering work on dynamic legged locomotion, Raibert's strategy focuses on the control of hopping robots and later extended to bipedal and quadrupedal robots. The core idea is to control the balance of a robot by adjusting the position of its center of mass (CoM) and manipulating its leg thrusts and placement to achieve stable hopping or walking. Raibert's strategy includes the division of control tasks into separate components for balance, forward motion, and body orientation, allowing for simplified yet effective management of dynamic locomotion. This approach has significantly influenced the development of dynamically stable walking and running robots.

The control system used the **three-part algorithms** of Marc Raibert to specify desired behavior for each virtual leg and it used the **rules of the virtual leg** to coordinate the behavior of the physical legs.

To control the forward running speed, the control system positioned the foot of the virtual leg with respect to the center of mass of the body during each flight phase:

$$x_{f,d} = \frac{\dot{x}T_s}{2} + k_{\dot{x}}(\dot{x} - \dot{x}^*)$$

$$y_{f,d} = \frac{\dot{y}T_s}{2} + k_{\dot{y}}(\dot{y} - \dot{y}_d)$$

Where,

$x_{f,d}, y_{f,d}$ is the desired displacement of the foot with respect to the projection of the center of mass,

\dot{x}, \dot{y} is the forward running speed,

\dot{x}_d, \dot{y}_d is the desired forward running speed,

T_s is the duration of a support period, and

$k_{\dot{x}}, k_{\dot{y}}$ are gains.

To control the pitch and roll attitude of the body during stance, the control system applied torques about the virtual hips, using linear servos

$$\begin{aligned} u_x &= -k_{p,x}(\varphi_P - \varphi_{P,d}) - k_{v,x}(\dot{\varphi}_P) - k_{f,x}(f_x) - k_{\dot{y}}(\dot{y}_d) \\ u_y &= -k_{p,y}(\varphi_R - \varphi_{R,d}) - k_{v,y}(\dot{\varphi}_R) - k_{f,y}(f_y) \end{aligned}$$

Where,

u_x, u_y are the servovalve output signals for the hip actuators,

φ_P, φ_R are the pitch and roll angles of the body,

$\dot{\varphi}_d$ is the desired hip rate of the actuator,

f_x, f_y are the forces measured in the hip actuators, and

$k_p, k_v, k_f, k_{\dot{y}}$ are gains.

To obtain the desired position of the virtual foot with respect to the virtual hip could be used as the desired position of the physical feet with respect to the physical hips:

$$\begin{aligned} x_{h,i,d} &= x_{h,j,d} = x_{f,d} \\ y_{h,i,d} &= y_{h,j,d} = y_{f,d} \end{aligned}$$

Where,

$x_{h,i,d}, y_{h,i,d}$ is the desired displacement of the i th foot with respect to the projection of the i th hip,

$x_{f,d}, y_{f,d}$ is the desired displacement of the virtual foot with respect to the projection of the virtual hip,

i, j are indices of two physical legs that form one virtual leg

An average yaw rate is taken for an entire stride. Use of the average yaw rate over the stride permits there to be variations in yaw rate within the stride, without interfering with the control of the machine's facing direction. Using average yaw rate to include turning:

$$x_{f,i,d} = \frac{\dot{x}T_s}{2} + k_x(\dot{x} - \dot{x}_d) + D \cos(\beta + \beta_{0,i})$$

$$y_{f,i,d} = \frac{\dot{y}T_s}{2} + k_y(\dot{y} - \dot{y}_d) + D \cos(\beta + \beta_{0,i})$$

Where,

i indicates the physical leg,

$\beta_{0,i}$ is $\arctan(W/L)$ for $i = 1,3$ and $-\arctan(W/L)$ for $i = 2,4$,

$$D = \sqrt{(W^2/2 + L^2/2)}$$

Control Strategies and Algorithms

PID Controller

Standing for Proportional-Integral-Derivative, the PID controller is a widely used feedback control mechanism in industrial control systems. It calculates an error value as the difference between a desired setpoint and a measured process variable, then applies a correction based on proportional, integral, and derivative terms. The PID controller adjusts the control input to minimise the error over time, ensuring the system's output behaves as intended. Its simplicity, ease of implementation, and effectiveness in a wide range of conditions make the PID controller a ubiquitous tool in the automation and control of processes, including the control of robots for tasks requiring precise movement and stability.

Other Controllers

Controllers for quadruped robots span a spectrum from basic reactive systems to complex algorithms that fuse perception, planning, and action to navigate and interact with varied environments. At the foundational level, **Proportional-Derivative (PD) Controllers** are pivotal for joint-level management, ensuring balance and precise movement by adjusting to the desired position and its rate of change, a technique employed across many quadrupeds for stabilizing and achieving various gaits. **Central Pattern Generators (CPGs)** [47], drawing inspiration from biological locomotion, generate rhythmic movement patterns, enabling dynamic walking and running, as exemplified by the Cheetah robot's agile maneuvers. **Model Predictive Control (MPC)** offers a forward-looking

strategy, optimizing control inputs over a predictive horizon to navigate uneven terrains efficiently, a method harnessed by robots like ANYmal and the MIT Cheetah for adaptive locomotion.

Whole-Body Control (WBC) treats the robot as an integrated system, optimizing command signals across all joints and limbs to follow desired trajectories or maintain balance while conserving energy, a strategy that underpins the Mini-Cheetah's capability to execute complex maneuvers and recover from disturbances. The **Raibert Control Strategy**, foundational to early hopping robots, focuses on foot placement and body equilibrium for maintaining momentum, influencing designs such as Boston Dynamics' BigDog for stable traversal. **Hybrid Control Systems** merge force and position control, enabling responsive adaptation to environmental interactions, seen in HyQ [\[18\]](#) and HyQ2Max as they dynamically adjust to varying terrains.

Feedforward and Feedback Controllers blend predictive model-based actions with real-time sensor feedback to refine locomotion precision, a combined approach enabling advanced navigation capabilities in robots like Spot. Lastly, **Adaptive and Learning-Based Controllers** leverage machine learning to evolve robot behavior over time, enhancing efficiency and robustness through continual interaction with the environment, a path pursued by ANYmal for ongoing locomotion improvement. Together, these diverse controller types equip quadruped robots with a broad toolkit, from the simplicity and directness of PD control to the nuanced, anticipatory strategies of MPC and adaptive learning, enabling them to tackle an expansive array of tasks across challenging and unpredictable environments.

CHAPTER 3: METHODOLOGY

Kinematics

Each leg of a quadrupedal robot can be treated as a separate manipulator to compute its forward and inverse kinematics. The result of these formulations can then be simply related to the world frame (frame connected to the COM of the body) via translation vectors. During the swing phase, the foot is considered the end-effector while during the stance phase, the prediction between the ground and the foot causes the body to move in relation to the ground hence the shoulder is considered the end-effector.

Mathematical Formulation:

1. We started off by assigning a **reference frame** to the bot. The selected convention is shown in the figure below. This frame was selected as the global frame of reference and all the calculations were carried out on these conventions.
2. The hip, thigh and calf lengths were identified. The Hip link is in the YZ plane while the thigh and calf links lie in the XZ plane. Cumulatively, this forms the XYZ coordinated form the foot to the shoulder.
3. In order to link the foot frame to the Centre of Mass (COM) of the body, we calculated the **offset vectors** of each shoulder from the body.
4. Using these **geometric parameters** and reference points we were able to formulate a series of linear equations using trigonometric manipulation to relate the XYZ coordinates of the feet to the respective joint angle.
5. However, there still remains the issue of multiple solutions for the same set of coordinates. In order to remove this ambiguity, we **limited the joint angles** to the prescribed workspace such that

$$-90 < \theta_h < 90$$

$$-90 < \theta_t < 90$$

$$0 < \theta_c < 180$$

6. The kinematics were implemented using the stick figure (Figure 7) modelled in SolidWorks to eliminate multiple solutions and cater for reachability and workspace constraints. The mathematical model was finalized via iterations based on this model to keep the signs and directions consistent with the reference frame selected.

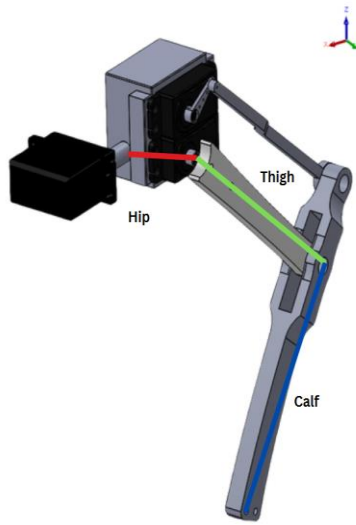


Figure 4: Reference Frame assignment and Link definition

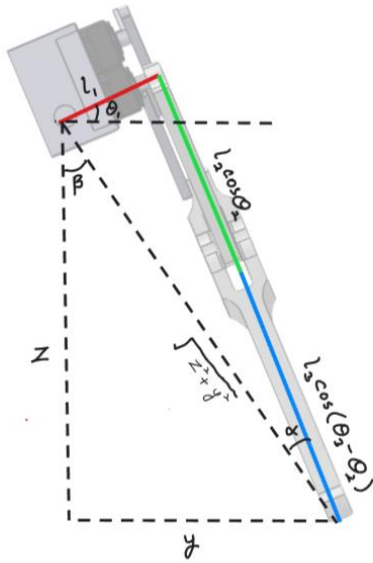


Figure 5: Inverse Kinematics - 1

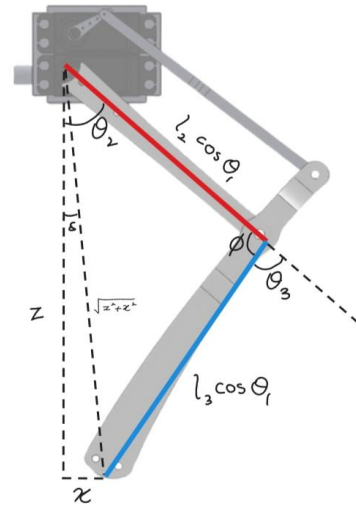


Figure 6: Inverse Kinematics - 2

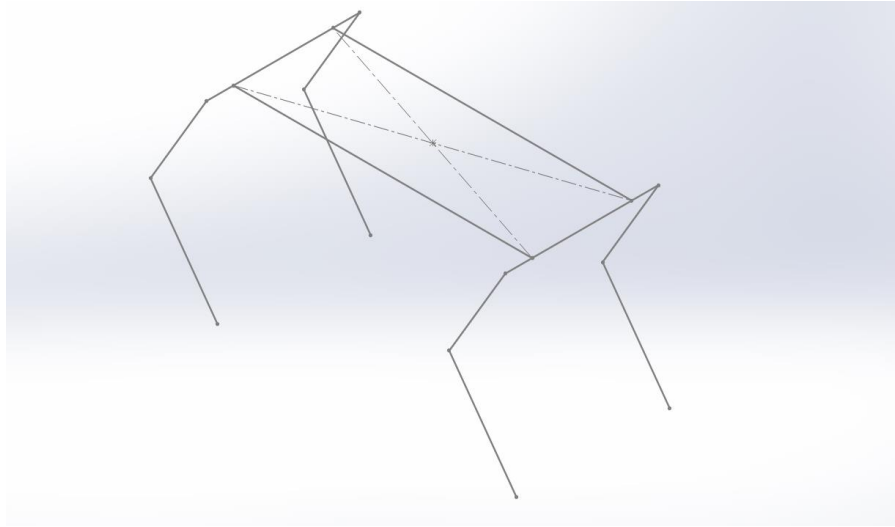


Figure 7: Stick Figure

Gaits and Algorithm

Gait and Stability

Currently our bot is being tested for trot gait. Trotting involves moving alternate legs in tandem to achieve forward motion. Raibert's inverted pendulum and virtual leg strategies are used to create this motion. The virtual leg analogy clumps the alternate legs together into an imaginary single leg connected to the COM of the quadruped such that two virtual legs need to be controlled in order to generate the gait. Hence the problem of quadrupedal control is reduced to controlling a biped robot.

1. **Balance and Stability:**

- The inverted pendulum model is used to maintain the robot's balance by dynamically adjusting the position of the legs to keep the robot's centre of mass (CoM) over its base of support. This involves calculating the necessary adjustments in the leg's position and force to counteract the gravitational force and any external disturbances. For any velocity there exists a neutral point at which, under ideal conditions, the velocity of the body when entering the stance phase is equal to the velocity when it leaves the stance phase. The positioning of the foot ahead

or behind this neutral point determines the acceleration or deceleration and is calculated by the error signal in velocity.

- The virtual leg strategy simplifies these calculations by providing a unified model for controlling the movement of all legs in coordination, effectively acting as a single point of control for balance.

2. Locomotion Control:

- The **virtual leg strategy** offers a method to control the robot's movement direction and speed by adjusting the length and angle of the virtual leg, which in turn influences the real legs' movements. This is particularly useful for varying the gait, stride length, and speed of the robot dynamically.
- The **inverted pendulum model** complements this by ensuring that, as the robot moves, it remains balanced. Adjustments made to the virtual leg for locomotion are continuously monitored and modified as necessary to prevent the robot from falling over, using the principles of the inverse pendulum to maintain stability.

3. Implementation:

- Sensors and feedback mechanisms are critical for the successful implementation of these combined strategies. Sensors provide real-time data on the robot's orientation, acceleration, and the position of its CoM, which are used to adjust the legs' actions continually. An **Inertial Measurement Unit (IMU)** is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the magnetic field surrounding the body, using a combination of accelerometers, gyroscopes, and magnetometers. An IMU is integral to maintaining and managing the stability of a quadrupedal robot by providing vital data on its orientation and movement. The IMU, with its accelerometers and gyroscopes, feeds real-time information about linear accelerations and angular rates to the robot's control system. This enables the robot to make instantaneous adjustments to its leg positions and body posture to balance actively, adapt its gait to different terrains, navigate with precision, and initiate fall recovery processes. Essentially, the IMU's inputs are critical for the robot to understand and react to its physical state and the dynamics of its

environment, ensuring stable and efficient movement across various activities and conditions.

- Control algorithms calculate the necessary adjustments to the virtual leg's parameters and translate these adjustments into movements of the actual legs, ensuring that the robot remains stable and moves according to the desired trajectory.

Matlab Simulation

Simulink Model

- **Solver:** The model uses time equation formulation and default solver type (variable-step). Additionally, the linear algebra type is set to sparse.
- **Environment:** The ground sub-system contains the blocks to initialise the environment of the simulation. This includes a world frame connected to the ground block and a solver configuration block. It also includes a mechanical configuration block to introduce gravity.
- **6 DOF Joint and Body:** In order to make the bot freely mobile in the environment a 6 DOF joint is used. All the components of the bot are connected to this 6 DOF Joint via Rigid Transform blocks. These **Rigid Transform blocks** introduce appropriate translations and rotations from one reference frame to another. The 6 DOF joint is connected to the body of the block which is a solid rectangular block.
- **Leg Subsystems:** The leg subsystems contain 3 Solid blocks representing the hip, thigh and calf limbs respectively. They are connected by Revolute Joint blocks. The revolute joints are set to rotate about the Z axis (default). The appropriate direction of the Z axis is ensured by adding reference frames within each solid block.
- **Foot:** The foot is represented by a Solid Sphere block connected to the final reference frame of the calf. A spherical foot profile is selected to simulate ideal point contact and use the Hertzian contact theory (Sphere and Plane Surface) to estimate normal force values.
- **Joint Control:** The joints are torque actuated via a feedback mechanism. The Feedback block takes the target angle, current angle and the current angular velocity and using a cascade control system obtains a torque signal that is fed into the joint.

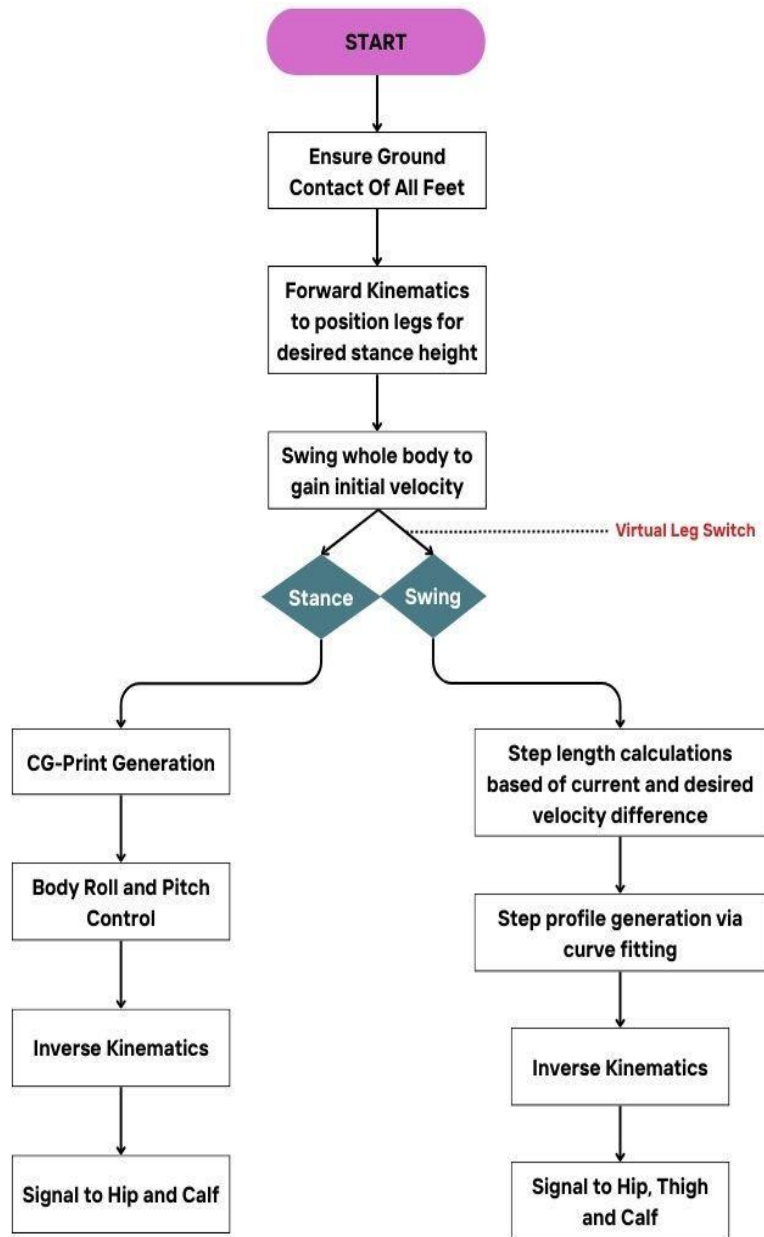


Figure 8: Control Algorithm

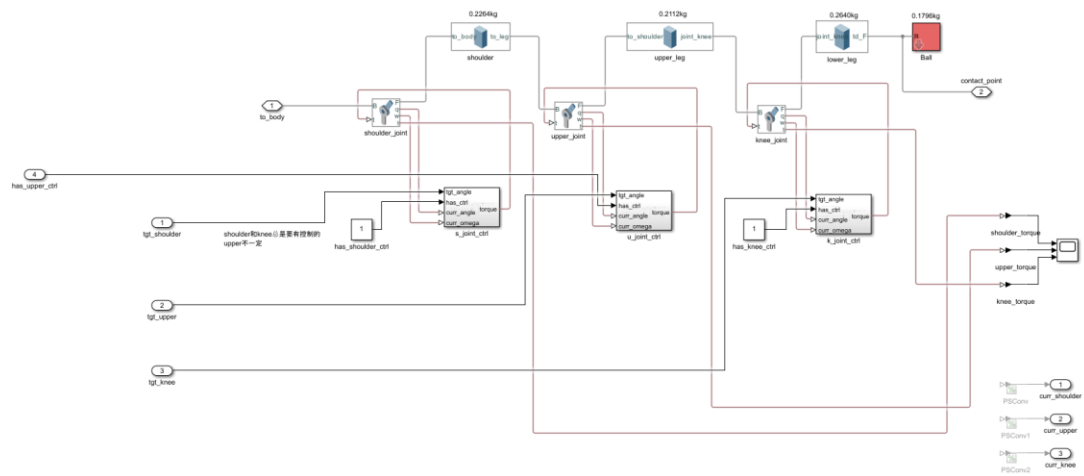


Figure 9: Leg Modeling in Simulink

Control System

The implemented code in the control system of the model can be explained in a series of steps.

1. **Initialise variables for body velocity, leg angles, end-effector positions, state and timer.**
2. **Calculate Leg Angles:**
 - For each leg (front-left, front-right, rear-left, rear-right), unpack the angles for the shoulder, upper limb, and knee joints.
3. **Perform Forward Kinematics:**
 - Calculate the position of each leg's end-effector using forward kinematics functions.
4. **Compute Shoulder Offsets:**
 - Determine the distance from the COM of the body to the shoulders of each leg.
5. **Virtual Leg Positions:**
 - Calculate the current positions of the virtual legs, which are the averages of the diagonal pairs of real legs.

6. State Machine Logic:

- **Standby State (State 0):**
 - Check if all feet are on the ground and if the body's velocity is below a certain threshold.
 - Transition to State 1 if conditions are met for a specified duration.
- **Initial Velocity Swing (State 1):**
 - Swing the whole body to achieve initial velocity.
 - Transition to State 2 if the body's velocity exceeds a certain threshold for a specific time.
- **Stance/Swing Phase (State 2):**
 - For the stance phase, disable upper limb control and balance the body.
 - For the swing phase, consider the body's angle and move the legs accordingly.
 - Transition to State 3 based on time and contact states.
- **Swing/Stance Phase (State 3):**
 - Same as State 2 but with roles of virtual legs swapped.
 - Transition back to State 2 based on time and contact states.
- **Loop:** Based on timer and contact state sensing, the roles of virtual legs are swapped alternatively between stance and swing. During the state 2 virtual leg 1 (forward left and rear right legs) are in stance phase and virtual leg 2 (forward right and rear left legs) are in swing phase. When the virtual leg 2 reaches stance phase, the controller switches to state 3 where the virtual leg 1 swings. This alternate action of virtual leg continuously causes the bot to trot.

7. Inverse Kinematics:

- For each leg, calculate the target joint angles from the desired end-effector positions.

8. Return Values:

- Output the target control angles for each leg, the control state, and debugging information.

9. PID Controller:

- The target control angles are fed into the respective leg subsystems. Here, these signals are unpacked into three separate signals i.e Hip, Thigh and

Calf angles fed into their respective PID subsystems.

The PID subsystem uses a cascade control system involving two PIDs in series to generate a torque signal. The first PID inputs a differential between the current and the target angle. This controller is configured to minimise the error in the angle effectively acting as a position controller. The output of this PID is differentiated with the current angular velocity of the joint. This differentiation helps to take into account the rate of change of the joint's angle, adding a derivative control aspect that anticipates future error based on current motion.

The result from the differentiation step is used as the input for the second PID controller. This PID controller is tuned to control the velocity of the joint, aiming to match the desired velocity as determined by the first PID controller's output and the current motion of the joint.

Benefit of Cascade Control System

- 1. Precision in Tracking Target Position:** The cascaded arrangement allows for precise control of the joint's position by first addressing the position error and then fine-tuning the velocity to reach and maintain that position.
- 2. Improved Response and Stability:** The differentiation step provides predictive control, which helps in improving the response time and stability of the system by countering the oscillations that might occur due to sudden changes in the target position.
- 3. Decoupling of Position and Velocity Control:** By using two separate PID controllers, the control of position and velocity are decoupled. The first controller focuses on achieving the desired position, while the second controller ensures that the velocity is controlled during this process. This can lead to smoother motion and better handling of dynamic changes.

Finite Element analysis:

As our quadrupedal bot was designed with a low cost in mind we had to choose an appropriate material for 3D printing that is cheap and offers adequate strength simultaneously. Another parameter that was kept in mind was the availability of the selected material in the local market. Therefore, we selected Polylactic Acid (PLA) as our material of choice.

Table 1: Properties of Material

Property	Value	Unit
Young's Modulus	3.5	GPa
Poisson's Ratio	0.36	N/A
Density	1240	kg/m ³

To validate our material selection and leg design we performed finite element analysis on COMSOL Multiphysics. This was performed for two different load conditions on each limb i.e.

1. The limb is vertical and maximum compressive stress acts on it.
2. The limb is horizontal and a maximum shear stress acts on it.

Model parameters:

- **Study Type:** Solid Mechanics
- **Solver Type:** Stationary
- **Geometry:** 3D
- **Mesh Size:** Extra Fine
- **Physics:** The servo ends of the thigh and calf were treated as **fixed constraints** and **boundary loads** were applied to the lower ends of both the limbs.
- **Load:** FEA was conducted on the estimated maximum load that the legs had to bear. During trotting, the entire weight of the body is rested on two legs at a time. Hence each leg bears half the weight of the entire quadruped (**1.65 kg**). Upon applying the concepts of statics each limb had to bear a maximum of **~8.2 N**.

The servo housing contains the thigh and calf actuation servos so that all the servos are situated on the body of the quadruped. This eliminates unwanted inertias in the design. The housing is externally connected to the shoulder servo.

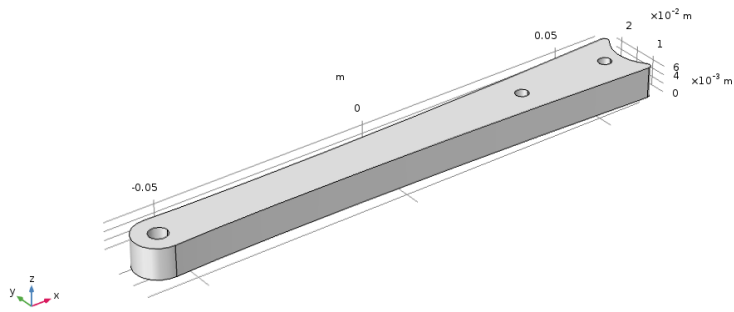


Figure 10: Thigh Link model in COMSOL

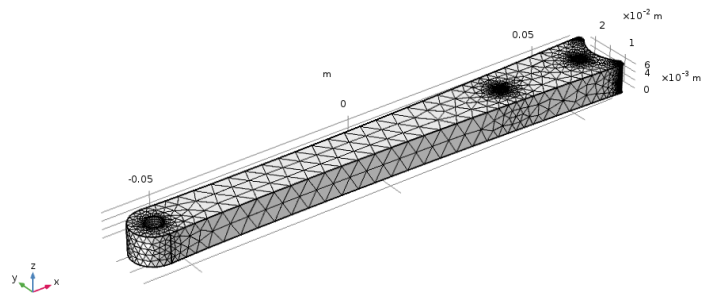


Figure 11: Thigh link Meshing in COMSOL

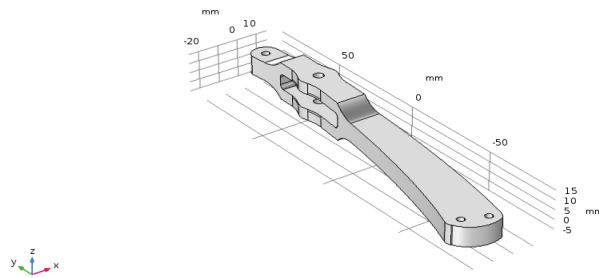


Figure 12: Calf Link Model in COMSOL

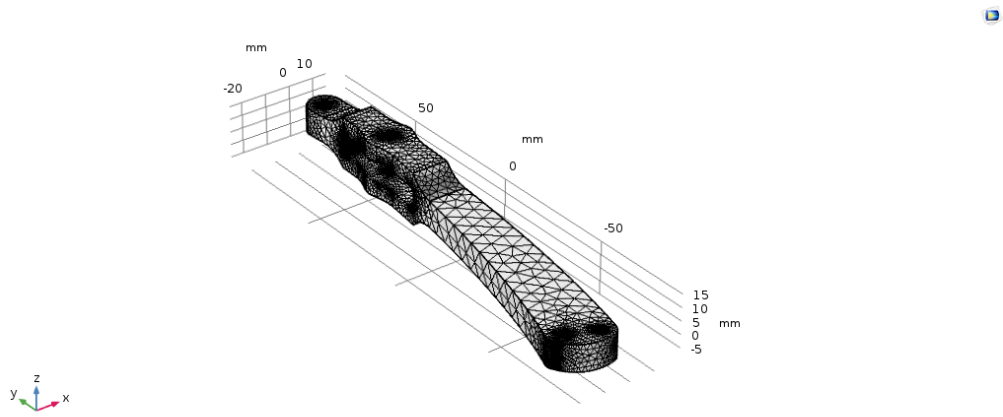


Figure 13: Calf Link Meshing in COMSOL

CAD:

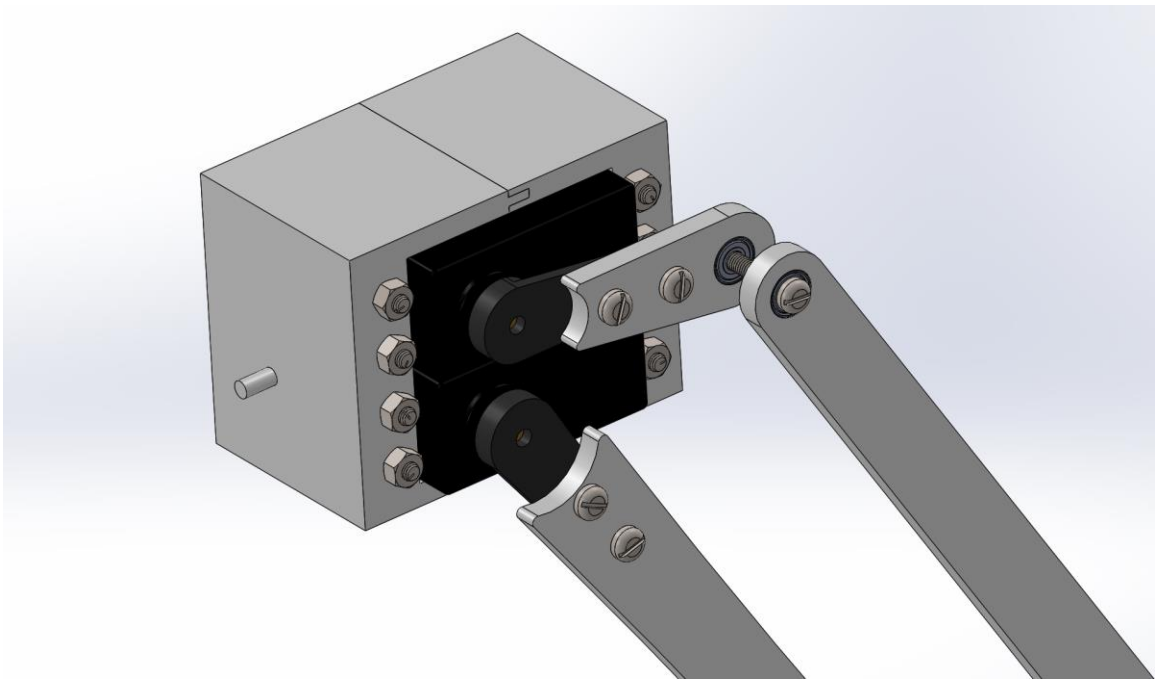


Figure 14: Hip Assembly

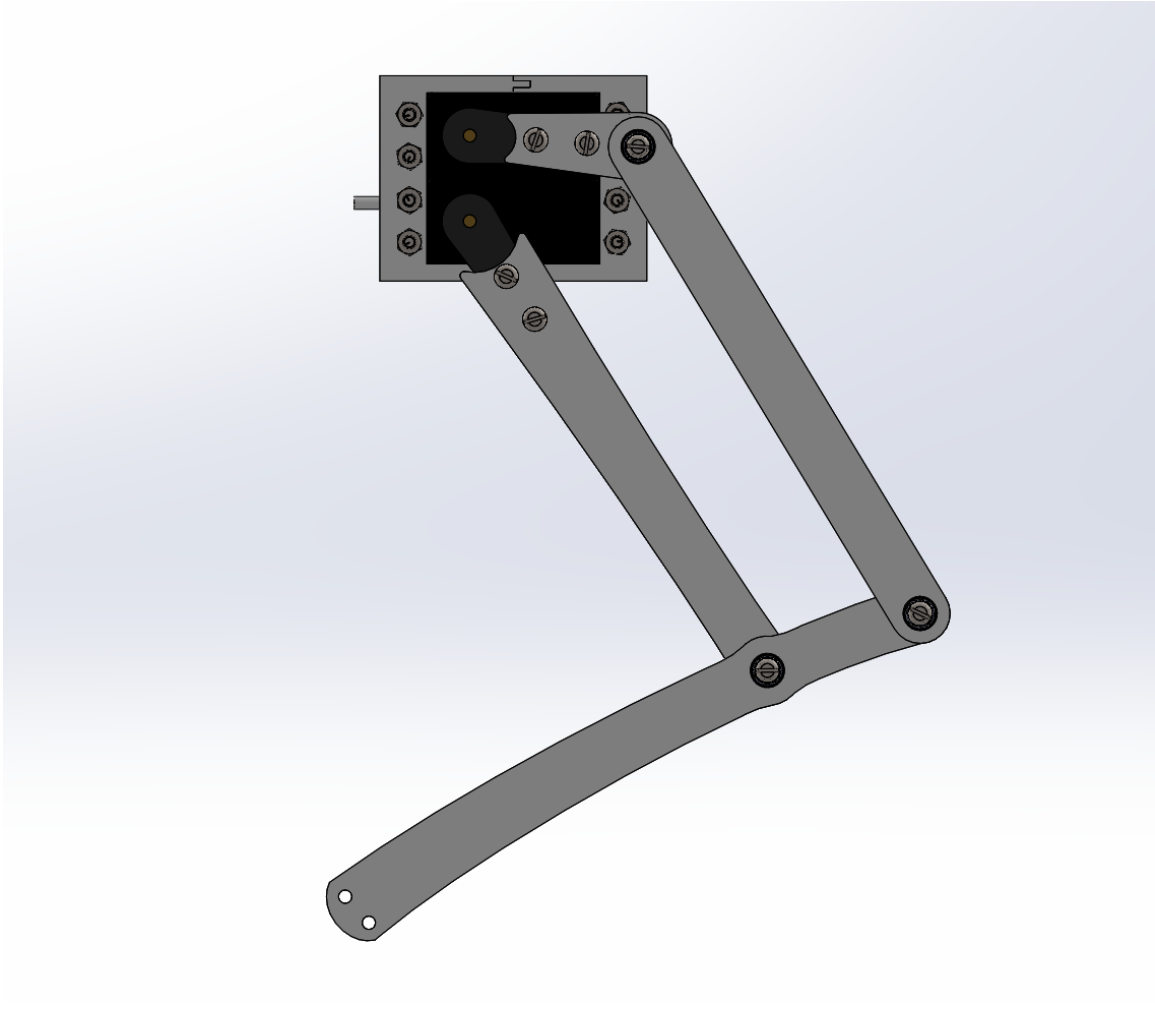


Figure 15: Leg Assembly

The thigh is actuated directly by the thigh servo. However, due to the positioning of the calf servo, the lower limb has to be actuated via a **4-bar kinematic linkage**. The calf acts as a rocker, the servo acts as the crank and a connecting rod is designed to act as a coupler. Designing a 4 bar kinematic linkage was also an iterative process. Several industrial designs were studied including 5-bar linkages and 6-bar linkages. However, such a complex system added weight to the bot and a 4-bar mechanism sufficed for the mobility requirements of our leg hence it was opted.

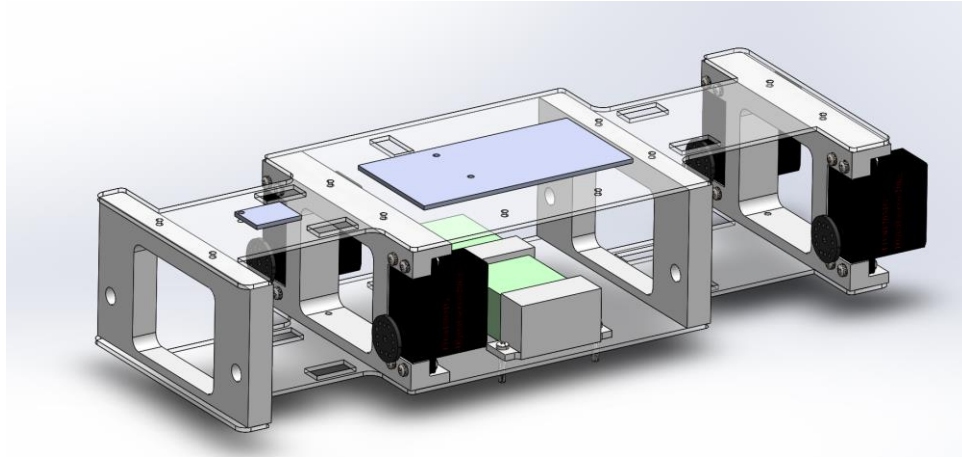


Figure 16: Robot Body

The body of the robot is designed to house the shoulder servos and provide support to the servo housing. In addition to this, the body will provide a platform to mount the microcontroller, IMU and the battery.

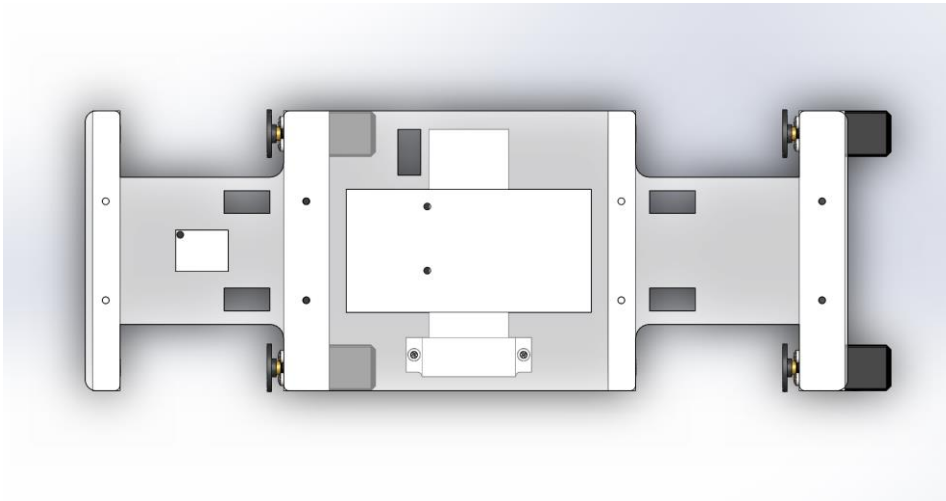


Figure 17: Robot Body (Top View)

The upper and lower plates of the body are made of acrylic sheets. The supports between the plates are of 3D printed PLA. The supports are designed specially to house the shoulder servos.

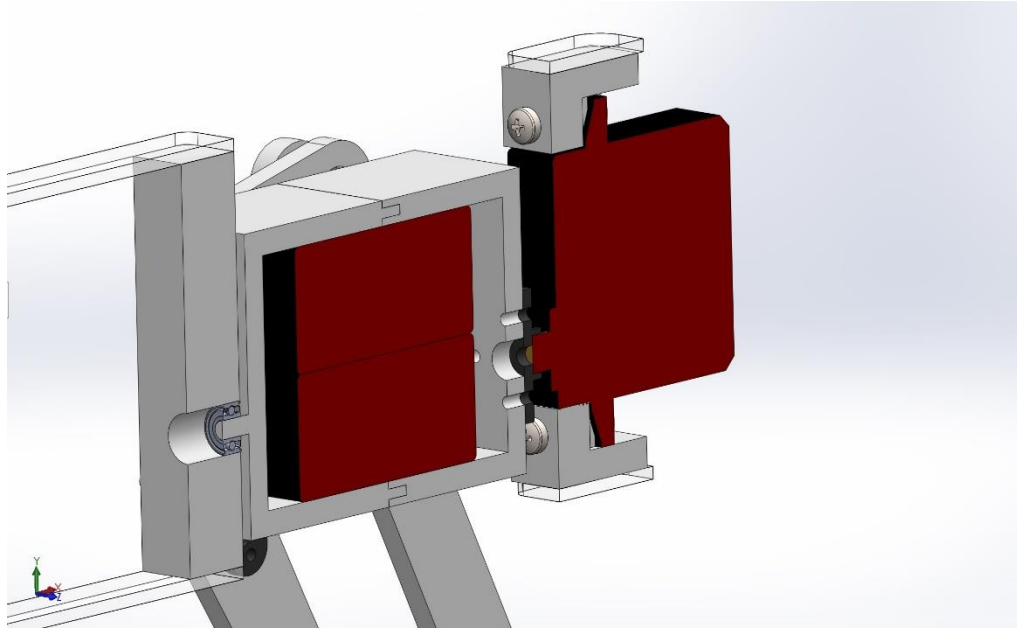


Figure 18: Hip Actuation Mechanism

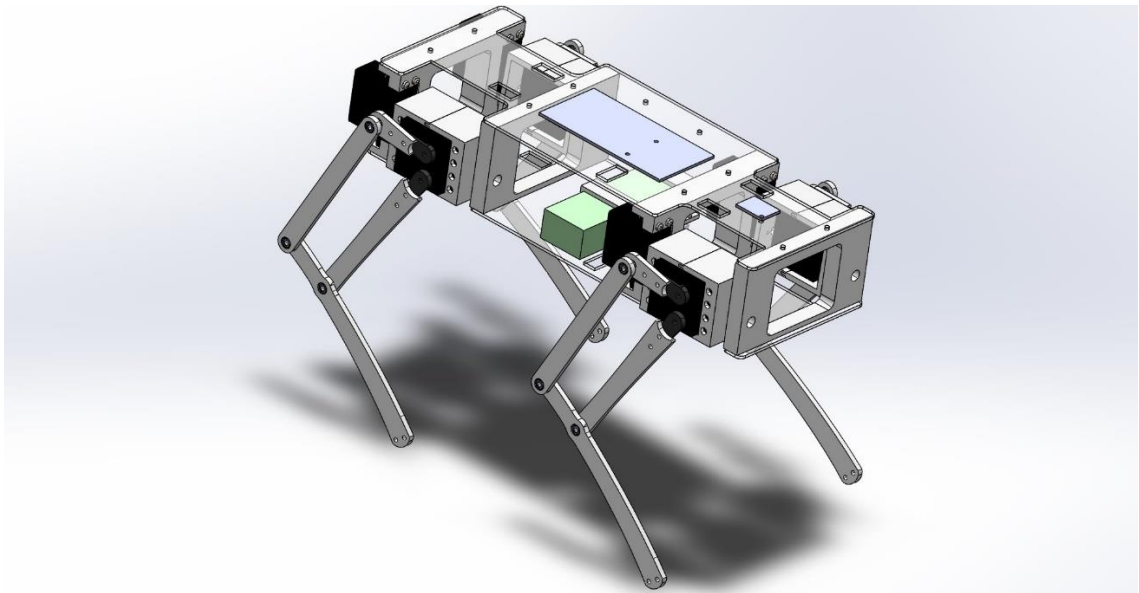


Figure 19: Complete Assembly

The final assembly showcases how all components including all four legs, servos, Arduino and battery are connected and mounted onto the body of the robot.

Assembly:

The robot is designed so that it assists in easy assembly and disassembly. We shall now provide a breakdown of the parts and the entire assembly process that was employed.

Part List:

- Servo Housing Front Bracket (4x)
- Servo Housing Rear Bracket (4x)
- Thigh (4x)
- Calf (4x)
- Knee Joint Servo Extension Link (4x)
- Coupler Link (4x)
- Hip Servo Support (2x)
- Base Plate Supports (2x)
- SKF 639 Bearing (28x)
- M3 Nuts and Bolts and Washers
- Base Plates (2x)
- DS3218 20kg Servos (12x)

Procedure:

1. Start off by mounting the hip servos on the dedicated supports. Each support holds two servos. All the fastening is done using M3 Nuts and Bolts.
2. Fasten the circular servo horns onto the rear housing bracket against the dedicated holes.
3. Attach the horn to the hip servo and screw it.
4. Next, place the other two servos (thigh and calf) inside the housing bracket and close them off by placing the front housing bracket. Fasten the motors and housing securely. The housing is designed in a way that the motors hold both the brackets together.
5. Press fit a bearing onto the shaft extending from the front housing bracket. We shall call these hip bearings.
6. Carry out these steps for all the 4 legs.
7. Now press fit the base plate supports on the hip bearings.

8. Finally adjust the position of the base plates and fasten them onto the supports. One base plate will be on the top of the bot and the other one on the bottom.
9. We will now move on to the assembly of the legs. First of all press fit bearings in all the dedicated holes on the limbs. There will be a total of 6 bearings per leg. These bearings ensure the smooth movement of pin joints.
10. Now form the knee joint by inserting an M3 bolt through the bearings making sure the holes of the thigh and calf limb align. Insert washers on both sides and between the bearings to ensure space and avoid unwanted friction. The washers also make sure that the nut and grip the inner ring of the bearing instead of the entire bearing in which case the purpose of the bearing will be lost.
11. In similar manner form pin joints between calf and coupler and coupler and Knee Joint Servo Extension Link. This forms one leg.
12. In a similar manner assemble all four legs.
13. Once all the legs are assembled, mount the thigh link on the lower servo in the housing and the Knee Joint Servo Extension Link on the upper servo in the housing using one arm servo horns. (3mm holes had to be drilled into the servo horns in order to fit M3 bolts in them).

Self-Balancing Algorithm:

The code for the self-balancing of the robot is divided into two main parts i.e. the `setup()` function where all the instances are initialised and the robot is brought to its stance position and the `loop()` function that runs continuously afterwards in order to ensure that the trunk of the robot remains in an upright position no matter the orientation of the ground. The working algorithm of the oecd is explained as follows:

1. Initialization (setup function):

- **Serial communication and I2C setup:**
 - Serial communication is initialised for debugging purposes.
 - I2C communication is started for interfacing with the MPU6050 sensor and the PWM driver for servos.
- **Body and Leg Initialization:**
 - Body and each leg of the quadruped are initialised.
 - Parameters like leg lengths, servo IDs, and other physical properties are set.
 - Initial servo angles are set using inverse kinematics.

2. Main Loop (loop function):

Sensor Reading and Orientation Calculation:

1. Sensor Reading (MPU6050):

- The loop continuously reads data from the MPU6050 sensor.
- The MPU6050 sensor provides raw data from its accelerometer and gyroscope.

2. Tilt Calculation:

- The gyroscope data is used to calculate the rate of change of the roll, pitch, and yaw angles.
- These rates are integrated over time to determine the current roll, pitch, and yaw angles.
- These angles represent the orientation of the robot with respect to its horizontal plane (roll and pitch) and its vertical axis (yaw).

Inverse Kinematics and Servo Control:

3. Inverse Kinematics (IK):

- For each leg of the quadruped, inverse kinematics calculations are performed.
- Inverse kinematics determines the joint angles required for each leg to achieve a desired orientation of the robot.
- These joint angles are calculated based on the desired roll, pitch, and yaw angles of the robot.

4. Servo Control:

- The joint angles obtained from inverse kinematics are converted to servo PWM values.
- These PWM values are sent to the servos.
- The servos adjust their positions based on these PWM values, thus adjusting the position of each leg.
- By controlling the servos, the robot maintains its upright orientation by adjusting the positions of its legs.

Timing Control:

5. Fixed Time Interval:

- The loop ensures a fixed time interval (**dt**) between iterations.
- This interval controls the frequency of updates and movements of the robot.
- By maintaining a constant time interval, the robot's movements are smooth and controlled.

Microcontroller, Actuator, Battery and IMU selection

Microcontroller: Arduino MEGA + WiFi R3 ATmega2560+ESP8266

- The Arduino MEGA 2560 provides plenty of processing power and memory so it can perform the computational tasks related to our quadruped robot's control system well.
- Featuring an expansive array of digital and analog I/O ports, as well as a multitude of communication interfaces, the board offers endless possibilities for the integration of sensors, actuators, and peripherals that the robot will need to operate.
- Compatibility with Arduino platform helps to integrate into a big community of developers where they have access to libraries and online resources to make development and troubleshooting processes simpler.
- In addition, the Arduino MEGA 2560 offers 12 PWM ports for controlling servos at the initial testing stage. Such a feature simplifies the initial setup and the calibration of the servo motors, especially important for the desired smooth and precise movement within the quadruped robot. In fact, thanks to the dedicated PWM ports it simplifies the servo signal managing process and makes the application more stable.
- While the MEGA 2560 has many powerful features, it has compact size so that it can fit within the chassis space constraints of the robot.
- Besides, it ensures that our project remains within budget without being required to compromise on performance or functionality.

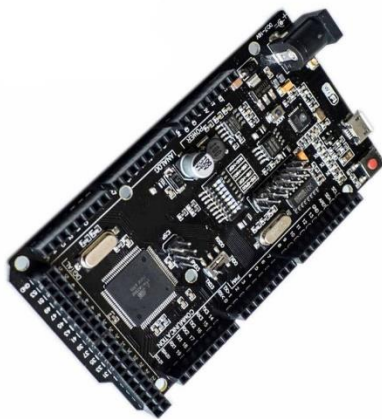


Figure 20: Arduino Mega

Actuator: DS3218 180-degree Position Control servo motor

- The DS3218 servo motor, with its torque rating of 20kgcm, fulfills my size, affordability, and torque requirements perfectly. Its durable metal gears ensure reliable operation, even under heavy loads or continuous use.
- With a wide operating voltage range, it offers flexibility in power supply options, accommodating various battery types or power sources.
- Position control capability is present in the DS3218 servo motor, making it ideal for easy control in quadruped robots. This feature allows precise manipulation of the servo's position, enabling smooth and accurate movement of the robot's limbs. By setting specific target positions, users can easily command the robot to perform desired actions such as walking, turning, or standing still. This simplifies the control process and enhances the overall user experience, especially for beginners or those seeking intuitive control solutions for their robotic projects.



Figure 21: DS3218 - 20KG Digital Servo

MPU – 6050



Figure 22: MPU-6050

- The MPU-6050 accelerometer and gyroscope module enables motion sensing, crucial for maintaining balance and stability in the quadruped robot. It provides Inertial Measurement Unit (IMU) functionality, allowing accurate tracking of the robot's movement in real-time.
- Advantages include comprehensive motion sensing, sensor fusion for accuracy, inertial navigation support, and dynamic stability control
- It is further advantageous due to the availability of Arduino libraries. These libraries simplify the integration of the MPU-6050 into your quadruped robot project, streamlining the development process and enabling quick and efficient access to its sensor data and functionalities.
- Compact integration ensures efficient use of space within the robot
- Availability and cost-effectiveness made it a convenient choice for timely implementation without compromising performance

Lipo Battery: 3S 2800mAh



Figure 23: 3S 2800 mAh LiPo Battery

- The 3S LiPo battery, with a lighter weight and advanced power, would be ideal to set up the agile and mobile robot. The battery has got 2800mAh capacity which

allows this device to operate for a while in normal usage. The average of 11.1 volts that we regulate for Arduino's 5 volts, and 6 volts for each servo without compromising on current distribution.

- The size and lightweight features of LiPo batteries make them easy to fit with almost any robotic design, hence they not only help with weight distribution but space utilization also.

Table 2: Parameters of Robot Dimensions

Parameter	Assigned Value (mm)
Hip length	27 mm
Calf Length	115 mm
Thigh Length	110.5 mm
Body Dimensions	306 x 122.3 x 225.5 mm
Hip-hip distance (x-axis)	136.5 mm
Foot radius	15 mm
Stance height	180 mm

CHAPTER 4: RESULTS AND DISCUSSIONS

Kinematics equations

Following are the forward and inverse kinematic equations:

Forward Kinematics

$$x = -l_2 \cos(\theta_1) \sin(\theta_2) - l_3 \cos(\theta_1) \sin(\theta_2 - \theta_3)$$

$$y = l_2 \sin(\theta_1) \cos(\theta_2) + l_3 \sin(\theta_1) \cos(\theta_3 - \theta_2)$$

$$z = -l_2 \cos(\theta_1) \cos(\theta_2) + l_3 \cos(\theta_1) \cos(\theta_3 - \theta_2)$$

Inverse Kinematics

$$\theta_1 = \tan^{-1}\left(\frac{y}{z}\right) - \sin^{-1}\left(\frac{l_1}{\sqrt{x^2 + y^2}}\right)$$

$$\theta_2 = \tan^{-1}\left(\frac{-x}{l_1 \sin(\theta_1) - (-z)}\right) - \sin^{-1}\left(\frac{l_3 \cos(\theta_1) \sin(\varphi)}{\sqrt{(l_1 \sin(\theta_1) - (-z))^2 + x^2}}\right)$$

$$\text{Given, } \varphi = \cos^{-1}\left(\frac{l_2^2 \cos^2(\theta_1) + l_3^2 \cos^2(\theta_1) - (l_1 \sin(\theta_1) - (-z))^2 + x^2}{2l_2 l_3 \cos^2(\theta_1)}\right)$$

$$\theta_3 = \pi - \cos^{-1}\left(\frac{l_2^2 \cos^2(\theta_1) + l_3^2 \cos^2(\theta_1) - (l_1 \sin(\theta_1) - (-z))^2 + x^2}{2l_2 l_3 \cos^2(\theta_1)}\right)$$

Dynamic Analysis

With the help of Lagrangian equations, we integrated it onto the Simulink model for our quadruped robot. The use of this method helped us establish the patterns in the function of mechanical components of the plant, and it helped us in the optimization of control algorithms. We achieved such insights as joint forces and motor operation by simply

working with scopes in Simulink to compute torques in MATLAB. This helped make informed choices for servo motors.

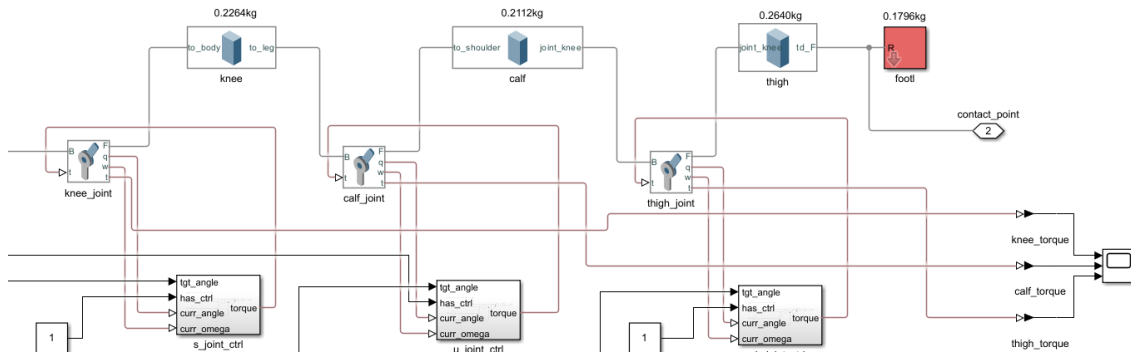


Figure 24: Simulink model of a single leg

Scripted motion planning and parameter finalisation

Scripted motion planning facilitated the meticulous optimization of parameters for our quadruped robot, focusing on body dimensions and leg lengths to determine the ideal centre of mass and stance positions. By systematically iterating through various configurations, we balanced stability and agility, ensuring efficient locomotion. This approach allowed us to explore different trajectories, refining our understanding of the robot's dynamics and interaction with its environment. Through simulation and testing, optimal configurations were identified, emphasising a balance between stability, manoeuvrability, and energy efficiency. The optimization outcomes that are calculated once the process is finalized signify the comprehensive optimization that was achieved, which can later be used to make necessary improvements. This calibrated approach enabled to obtain the value for conversion between PWM and robot motion angles in order to preserve accuracy in the final robot movement.

FEA Results

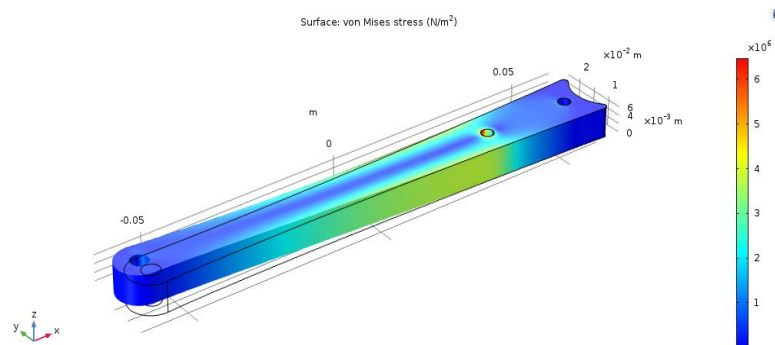


Figure 25: FEA Analysis – Thigh Link – Von Mises

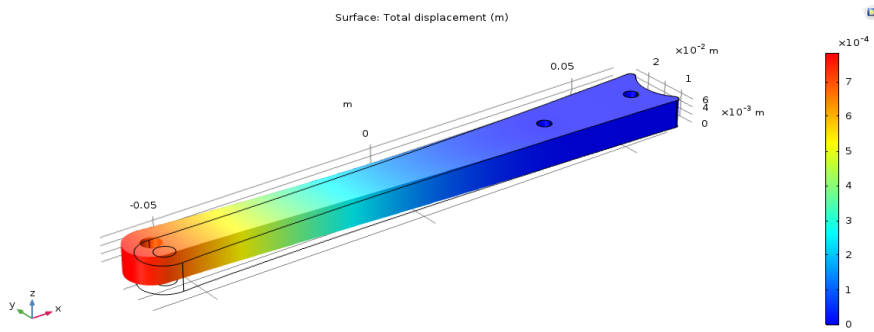


Figure 26: FEA Analysis – Thigh Link – Total Displacement

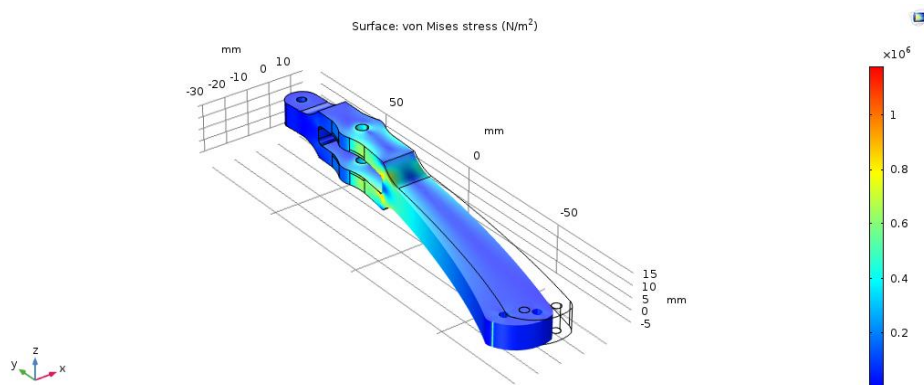


Figure 27: FEA Analysis – Calf link – Von Mises Stress

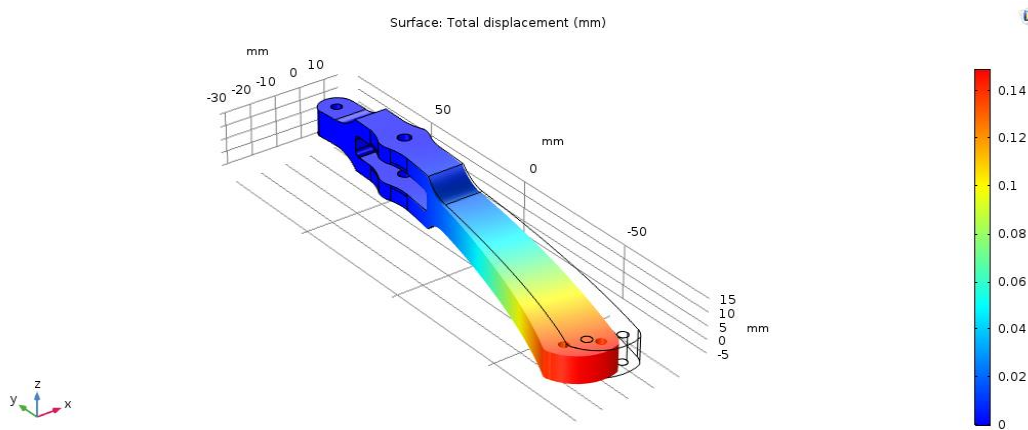


Figure 28: FEA Analysis - Calf link – Total Displacement

Table 3: FEA Results

Limb	Orientation (w.r.t Ground)	Maximum Von Mises Stress (MPa)	Maximum Deflection (mm)
Thigh	Vertical	0.3	30E-4
Thigh	Horizontal	6	0.7
Calf	Vertical	1	0.14
Calf	Horizontal	5	0.7

The leg design was an iterative process to compensate between a durable design that could bear high loads to the weight of the leg itself so that any unwanted inertias and overshoots can be eliminated. Finite element analysis helped us in optimising the leg design under these parameters.

The material opted for leg manufacture is Polylactic Acid (PLA) which is a high strength, low cost and easily available material. The tensile strength of PLA is 70 MPa. As the results of FEA show that the stresses are well within the tensile limit of the material.

A rather more interesting parameter is the deflection. Although the stresses are well within the tensile limit of the material, due to the high elasticity of the material, large deflection of the order $>1\text{mm}$ were seen in the design. This can cause serious computational errors in the control system of the robot. Hence a more stiff leg design was preferred and the final design caters all these needs.

Trajectory Generation

The foot trajectory is generated via a second order polynomial. The step height and the stride length are calculated based on the length of the virtual leg during the initial stance phase. The plot shows how the value of this coefficient varies with the time fraction of the swing leg. The trajectory formed by the foot is a quadratic curve.

A similar approach was used for the stance leg to generate the hip pattern for the inverted pendulum model. However, the magnitude of the coefficient for the stance leg is lower than the swing leg. This, too, was obtained using a second order polynomial.

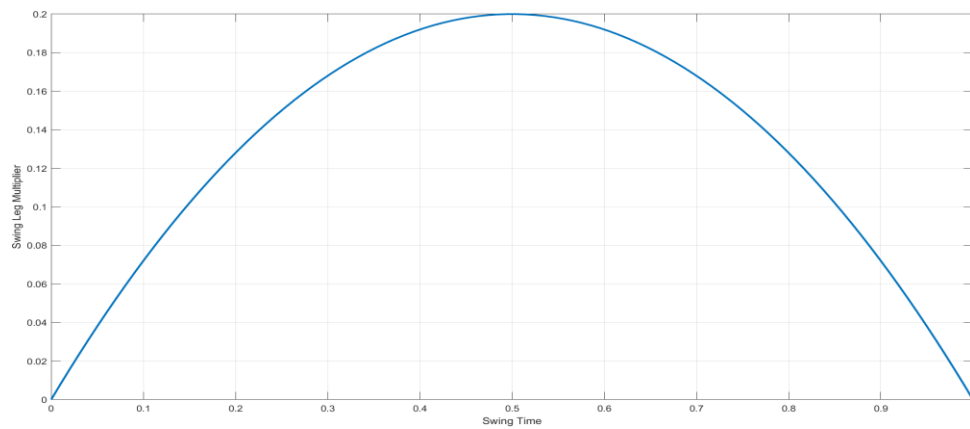


Figure 29: Foot Trajectory

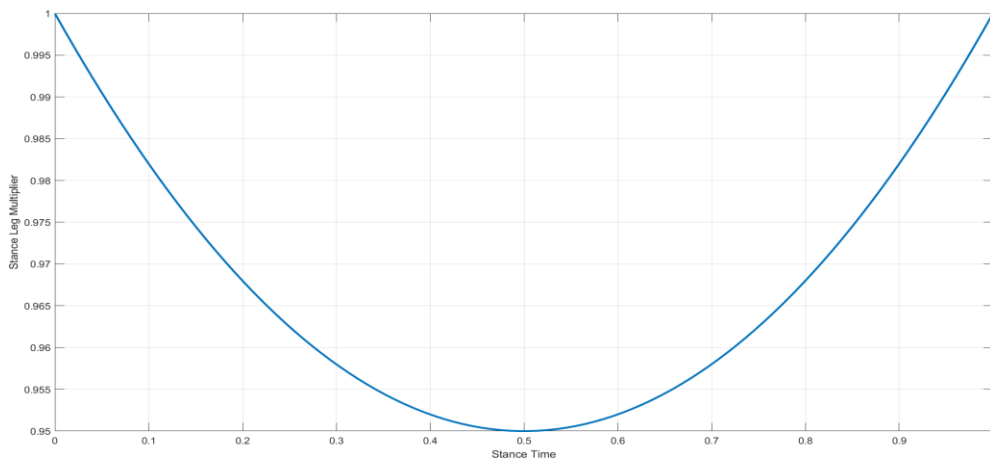


Figure 30: Stance Hip Trajectory

PID controller

The transition from a scripted motion planner to a PID controller marks a step towards sophistication of our quadruped robot's control system. With the realisation that the robot's base would not remain balanced during locomotion, necessitating continuous adjustments to maintain stability, the adoption of a PID controller offers a more dynamic and adaptive solution. Unlike the scripted motion planner, which pre-defines specific movements, the PID controller utilises feedback from sensors to continuously adjust motor commands, ensuring smoother and more stable motion. This shift represents a step forward in our quest to develop a sophisticated smart controller capable of responding to real-time

environmental changes and robot dynamics. By leveraging principles of feedback control, the PID controller enhances the robot's ability to navigate uneven terrain and adapt to unforeseen obstacles, ultimately improving its overall agility and performance. This transition underscores our commitment to innovation and continuous improvement, bringing us closer to achieving our goal of creating a highly capable and versatile quadruped robot.

Simulation: PID controller

The results of these simulations were particularly promising, as they demonstrated the successful implementation of a trot gait using a PID (Proportional-Integral-Derivative) controller. Through meticulous tuning of the PID controller parameters, we achieved smooth and stable motion during the trotting gait. The Simulink simulations allowed us to observe the robot's behavior in a virtual environment, providing valuable insights into its dynamics and performance. We iteratively adjusted the PID controller gains to optimize the gait for efficiency and stability. The successful trot gait demonstrated the effectiveness of our control algorithm in coordinating the movement of the robot's limbs. This achievement signifies a significant milestone in our project, indicating progress towards realizing a fully functional quadruped robot. Moving forward, we plan to further refine and validate our control strategies through experimentation and real-world testing. Overall, the MATLAB Simulink simulations played a crucial role in validating our approach and guiding future development efforts.

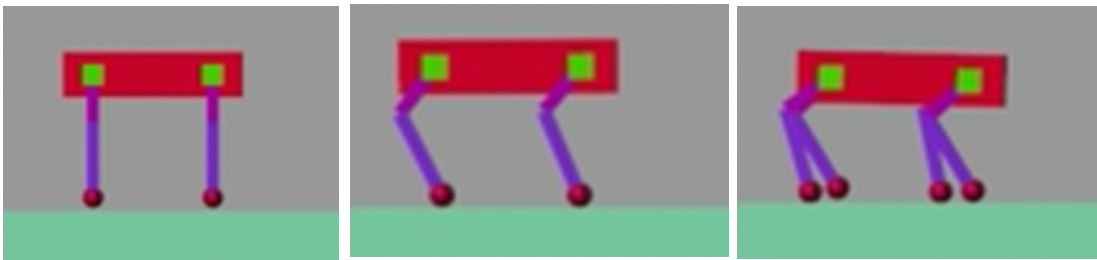


Figure 31: Simulink Multibody Simulation - Trot Gait (Sideview)

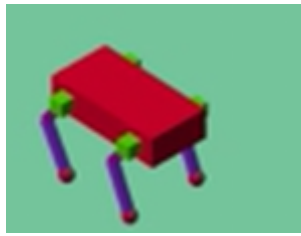


Figure 32: Simulink Multibody Simulation - Trot Gait (Isometric View)

Prototype Testing

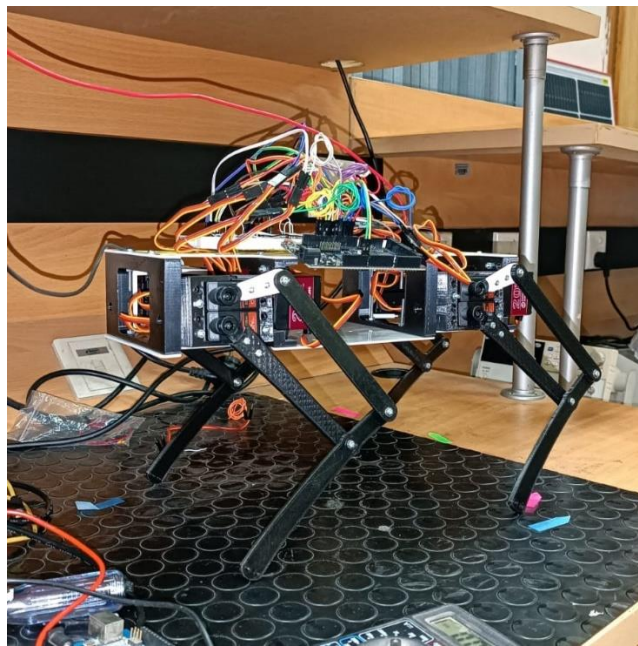


Figure 33: Stance Position under self-weight with no support

During prototype testing, the assembled quadruped robot demonstrated stability while standing on a table in a stance position, successfully supporting its weight. It also exhibited the capability to perform up and down motions when propped onto a raised surface. The calibration of leg servos ensured accurate movement control, facilitating tasks such as inverse kinematics testing. By providing foot positions in the code, the robot executed inverse kinematics and accurately placed its feet as intended, specifically the path of foot for step motion. Despite these achievements, stable walking is still under development, with ongoing testing focused on refining PID constants and IMU-PWM conversion constants for the physical model. These iterative tests are essential for enhancing the robot's

stability and overall performance, guiding further improvements in its locomotion capabilities.

A notable observation during testing was the haphazard motion exhibited by the robot when PWM signals were affected by noise. To address this issue, a PWM driver was integrated into the model, resulting in improved performance and stability. This integration effectively mitigated the impact of noise on the PWM signals, ensuring smoother and more consistent motion of the robot. By enhancing signal integrity, the PWM driver contributed to the overall reliability and accuracy of the control system, minimizing disruptions caused by external factors.



Figure 34: Step motion using IK (a) : Upward motion

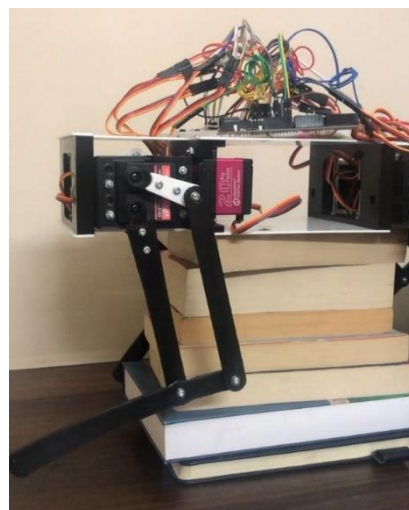


Figure 35: Step motion using IK (b) :Forward motion

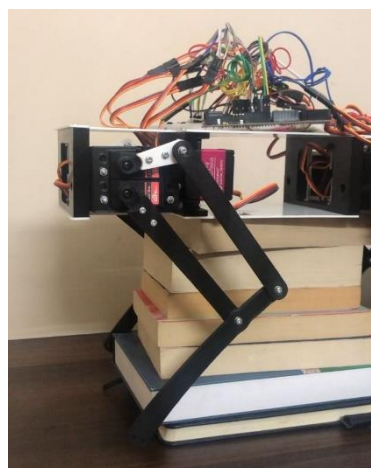


Figure 36: Step motion using IK (c) : Step complete.

CHAPTER 5: CONCLUSION AND RECOMMENDATION

The project on the design and development of a quadruped robot has been a comprehensive journey into the realms of robotics, combining mechanical engineering principles with innovative control strategies to create a robot capable of agile and dynamic movement. Throughout this endeavor, we have delved into the historical evolution of quadruped robotics, the intricacies of leg design, the critical aspects of kinematic and dynamic analysis, and the implementation of sophisticated control algorithms to achieve stability and efficient locomotion.

Our exploration began with a detailed literature review, tracing the milestones in quadruped robotics, from early reflexive responses to advanced locomotion strategies. The leg design section highlighted the shift towards biomimetic approaches, emphasizing articulated joints, compliant mechanisms, and sensor integration for adaptive movement. Kinematic analysis provided the mathematical foundation for understanding the robot's movement, while dynamic analysis, through the Lagrangian and Newton-Euler methods, offered insights into managing forces and torques for stable motion.

The methodology focused on the kinematics of each leg, gait analysis, and the implementation of control strategies, including PID controllers and Model Predictive Control, to navigate various terrains. Our experiments with finite element analysis (FEA) ensured that our design choices—specifically, the use of Polylactic Acid (PLA) for leg construction—were validated against stress and deflection criteria, ensuring durability and performance.

In the results and discussions chapter, we shared the outcomes of our design choices, from the selection of the Arduino MEGA 2560 PRO Mini and TIANKONGRC servos to the integration of the GY-951 IMU for motion sensing. The kinematics equations laid out the mathematical groundwork for our robot's movement, and the motion planning and parameter finalization process underscored the importance of optimizing the robot's design for balance, agility, and energy efficiency.

Recommendations

- Explore composite materials or advanced polymers for better weight, strength, and flexibility balance, enhancing performance in rugged environments.
- Integrate adaptive and learning-based strategies like reinforcement learning for improved adaptability and efficiency.
- Incorporate diverse sensors (LiDAR, depth cameras, tactile sensors) to improve environment interaction and enable complex tasks.
- Research into energy storage and management, like regenerative braking and solar power, to extend the robot's operational life.

Future Work

- Extensive field testing across different terrains (e.g., sand, rocks, and vegetation) to evaluate the robot's performance and identify areas for improvement in locomotion strategies and terrain adaptability.
- Development of communication protocols and algorithms for multi-robot cooperation, enabling a team of quadruped robots to perform coordinated tasks or explore complex environments more efficiently.
- Creation of an intuitive user interface for non-expert users to interact with the robot, including task programming, status monitoring, and manual control options, to broaden the robot's applicability in educational, research, and industrial settings.
- Implementation of safety and compliance mechanisms to ensure that the robot can operate safely in environments shared with humans, adapting its behavior to prevent collisions and injuries.

REFERENCES

- [1] G. N. Boone and J. K. Hodgins, "Reflexive responses to slipping in bipedal running robots," in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, Pittsburgh, PA, USA, 1995, pp. 158-164 vol.3, doi: 10.1109/IROS.1995.525878. [Online]. Available: <https://ieeexplore.ieee.org/document/525878>
- [2] Md Hasibur Rahman, Saadia Alam, Trisha Das Mou, Faisal Uddin, and Mahady Hasan, "A Dynamic Approach to Low-Cost Design, Development, and Computational Simulation of a 12DoF Quadruped Robot," *Robotics*, vol. 12, p. 24, 2023, doi: 10.3390/robotics12010028. [Online]. Available: https://www.researchgate.net/publication/368636484_A_Dynamic_Approach_to_Low-Cost_Design_Development_and_Computational_Simulation_of_a_12DoF_Quadruped_Robot
- [3] M. H. Raibert, B. Brown Jr., M. Chepponia, J. Koechilig, J. K. Hodgins, D. Duatman, and W. K. Brennan, "Dynamically Stable Legged Locomotion," [Online]. Available: <https://apps.dtic.mil/sti/tr/pdf/ADA225713.pdf>
- [4] J. Furusho, A. Sano, M. Sakaguchi, and E. Koizumi, "Realization of bounce gait in a quadruped robot with articular-joint-type legs," in *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, Nagoya, Japan, 1995, pp. 697-702 vol.1, doi: 10.1109/ROBOT.1995.525365. [Online]. Available: <https://ieeexplore.ieee.org/document/525365>

- [5] K. Matsuoka, "Sustained oscillations generated by mutually inhibiting neurons with adaptation," *Biological Cybernetics*, vol. 52, pp. 367-376, 1985. [Online]. Available: https://www.researchgate.net/publication/20158913_Sustained_oscillations_generated_by_mutually_inhibiting_neurons_with_adaptation
- [6] Y. Fukuoka, H. Kimura, and A. H. Cohen, "Adaptive Dynamic Walking of a Quadruped Robot on Irregular Terrain Based on Biological Concepts," *The International Journal of Robotics Research*, vol. 22, no. 3-4, pp. 187-202, 2003, doi: 10.1177/0278364903022003004. [Online]. Available: <https://journals.sagepub.com/doi/abs/10.1177/0278364903022003004>
- [7] H. Kimura, Y. Fukuoka, and A. Cohen, "Adaptive Dynamic Walking of a Quadruped Robot on Natural Ground Based on Biological Concepts," *I. J. Robotic Res.*, vol. 26, pp. 475-490, 2007, doi: 10.1177/0278364907078089. [Online]. Available: https://www.researchgate.net/publication/220122045_Adaptive_Dynamic_Walking_of_a_Quadruped_Robot_on_Natural_Ground_Based_on_Biological_Concepts
- [8] J. Buchli and A. J. Ijspeert, "Self-organized adaptive legged locomotion in a compliant quadruped robot," *Autonomous Robots*, vol. 25, no. 3, pp. 331–347, 2008. [Online]. Available: <https://doi.org/10.1007/s10514-008-9099-2>
- [9] Y. Fukuoka, H. Kimura, Y. Hada, and K. Takase, "Adaptive dynamic walking of a quadruped robot 'Tekken' on irregular terrain using a neural system model," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2003, vol. 2, pp. 2037-2042, doi: 10.1109/ROBOT.2003.1241893. [Online]. Available:

https://www.researchgate.net/publication/4041451_Adaptive_dynamic_walking_of_a_quadraped_robot_'Tekken'_on_irregular_terrain_using_a_neural_system_model

[10] Y. Fukuoka and H. Kimura, "Dynamic Locomotion of a Biomorphic Quadruped 'Tekken' Robot Using Various Gaits: Walk, Trot, Free-Gait and Bound," *Applied Bionics and Biomechanics*, vol. 6, pp. 63-71, 2009, doi: 10.1080/11762320902734208. [Online]. Available:

https://www.researchgate.net/publication/232891655_Dynamic_Locomotion_of_a_Biomorphic_Quadruped_%27Tekken%27_Robot_Using_Various_Gaits_Walk_Trot_Free-Gait_and_Bound

[11] M. Raibert, K. Blankespoor, G. Nelson, and R. Playter, "BigDog the Rough-Terrain Quadruped Robot," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 10822-10825, 2008, doi: 10.3182/20080706-5-KR-1001.01833. [Online]. Available:

<https://doi.org/10.3182/20080706-5-KR-1001.01833>

[12] D. Wooden, M. Malchano, K. Blankespoor, A. Howardy, A. A. Rizzi, and M. Raibert, "Autonomous navigation for BigDog," in *2010 IEEE International Conference on Robotics and Automation*, Anchorage, AK, USA, 2010, pp. 4736-4741, doi: 10.1109/ROBOT.2010.5509226. [Online]. Available: [http://refhub.elsevier.com/S2405-](http://refhub.elsevier.com/S2405-8440(18)32693-8/sref8)

[8440\(18\)32693-8/sref8](http://refhub.elsevier.com/S2405-8440(18)32693-8/sref8)

[13] D. V. Lee and A. A. Biewener, "BigDog-inspired studies in the locomotion of goats and dogs," *Integrative and Comparative Biology*, vol. 51, no. 1, pp. 190-202, 2011, doi: 10.1093/icb/icr061. [Online]. Available: <https://doi.org/10.1093/icb/icr061>

- [14] M. P. Murphy, A. Saunders, C. Moreira, A. A. Rizzi, and M. Raibert, "The littledog robot," *The International Journal of Robotics Research*, vol. 30, no. 2, pp. 145-149, 2011. [Online]. Available: [http://refhub.elsevier.com/S2405-8440\(18\)32693-8/sref11](http://refhub.elsevier.com/S2405-8440(18)32693-8/sref11)
- [15] P. Filitchkin and K. Byl, "Feature-based terrain classification for littledog," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura-Algarve, Portugal, 2012, pp. 1387-1392, doi: 10.1109/IROS.2012.6386042. [Online]. Available: <https://ieeexplore.ieee.org/document/6386042>
- [16] A. Shkolnik, M. Levashov, I. R. Manchester, and R. Tedrake, "Bounding on rough terrain with the LittleDog robot," *The International Journal of Robotics Research*, vol. 30, no. 2, pp. 192-215, 2011, doi: 10.1177/0278364910388315. [Online]. Available: <https://journals.sagepub.com/doi/abs/10.1177/0278364910388315>
- [17] P. Filitchkin and K. Byl, "Feature-based terrain classification for LittleDog," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura-Algarve, Portugal, 2012, pp. 1387-1392, doi: 10.1109/IROS.2012.6386042. [Online]. Available: <https://ieeexplore.ieee.org/document/6386042>
- [18] C. Semini, N. G. Tsagarakis, E. Guglielmino, M. Focchi, F. Cannella, and D. G. Caldwell, "Design of HyQ—a hydraulically and electrically actuated quadruped robot," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 225, no. 6, pp. 831-849, 2011. [Online]. Available: https://www.researchgate.net/publication/262048609_Design_of_HyQ_-_A_hydraulically_and_electrically_actuated_quadruped_robot
- [19] G. Taga, Y. Yamaguchi, and H. Shimizu, "Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment," *Biological Cybernetics*,

vol. 65, no. 3, pp. 147–159, 1991, doi: 10.1007/BF00198086. [Online]. Available: <https://doi.org/10.1007/BF00198086>

[20] P. T. Doan, H. D. Vo, H. K. Kim, and S. B. Kim, "A new approach for development of quadruped robot based on biological concepts," *International Journal of Precision Engineering and Manufacturing*, vol. 11, pp. 559-568, 2010. [Online]. Available: https://www.researchgate.net/publication/227334049_A_New_Approach_for_Development_of_Quadruped_Robot_Based_on_Biological_Concepts

[21] X. Chen, K. Watanabe, K. Kiguchi, and K. Izumi, "An ART-based fuzzy controller for the adaptive navigation of a quadruped robot," *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 3, pp. 318-328, Sept. 2002, doi: 10.1109/TMECH.2002.802722. [Online]. Available: <https://doi.org/10.1109/TMECH.2002.802722>

[22] E. Jumantoro, A. H. Alasiry, and H. Hermawan, "Stability optimization on quadruped robot using trajectory algorithm," in *2017 International Electronics Symposium on Engineering Technology and Applications (IES-ETA)*, Surabaya, Indonesia, 2017, pp. 93-98, doi: 10.1109/ELECSYM.2017.8240385. [Online]. Available: <https://ieeexplore.ieee.org/document/8240385>

[23] K. Izumi, K. Watanabe, M. Shindo, and R. Sato, "Acquisition of Obstacle Avoidance Behaviors for a Quadruped Robot Using Visual and Ultrasonic Sensors," 2006, pp. 1-6, doi: 10.1109/ICARCV.2006.345078. [Online]. Available: https://www.researchgate.net/publication/221144151_Acquisition_of_Obstacle_Avoidance_Behaviors_for_a_Quadruped_Robot_Using_Visual_and_Ultrasonic_Sensors

[24] https://www.researchgate.net/publication/221786637_Selection_of_Obstacle_Avoidance_Behaviors_Based_on_Visual_and_Ultrasonic_Sensors_for_Quadruped_Robots

[25] M. H. Rahman, S. Alam, T. Das Mou, F. Uddin, and M. Hasan, "A Dynamic Approach to Low-Cost Design, Development, and Computational Simulation of a 12DoF Quadruped Robot," *Robotics*, vol. 12, p. 24, 2023, doi: 10.3390/robotics12010028.

[Online]. Available:

https://www.researchgate.net/publication/368636484_A_Dynamic_Approach_to_Low-Cost_Design_Development_and_Computational_Simulation_of_a_12DoF_Quadruped_Robot

[26] M. Hutter et al., "ANYmal - a highly mobile and dynamic quadrupedal robot," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, Korea (South), 2016, pp. 38-44, doi: 10.1109/IROS.2016.7758092. [Online]. Available: <https://ieeexplore.ieee.org/document/7758092>

[27] J. Gabriel, M. Khorram, and S. A. A. Moosavian, "Stable Stair-Climbing of a Quadruped Robot," 2018. [Online]. Available:

https://www.researchgate.net/publication/327571191_Stable_Stair-Climbing_of_a_Quadruped_Robot

[28] [28] *Novel Design of a Wheeled Robot with Double Swing Arms Capable of Autonomous Stair Climbing*, doi: 10.1145/3230876.3230899. [Online]. Available: <https://doi.org/10.1145/3230876.3230899>

[29] D. Papadopoulos and M. Buehler, "Stable running in a quadruped robot with compliant legs," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, San Francisco, CA, USA, 2000, pp. 444-449 vol.1, doi: 10.1109/ROBOT.2000.844095. [Online]. Available: <https://ieeexplore.ieee.org/document/844095>

[30] M. H. Rahman, M. M. Islam, M. F. Al Monir, S. B. Alam, M. M. Rahman, M. Shidujaman, and R. Islam, "Kinematics analysis of a quadruped robot: Simulation and Evaluation," in *2022 2nd International Conference on Image Processing and Robotics (ICIPRob)*, March 2022, pp. 1-6, doi: 10.1109/ICIPRob55092.2022.9776498. [Online]. Available:

https://www.researchgate.net/publication/361466674_Kinematics_analysis_of_a_quadruped_robot_Simulation_and_Evaluation

[31] J. A. Buford, R. F. Zernicke, and J. L. Smith, "Adaptive control for backward quadrupedal walking. I. Posture and hindlimb kinematics," *Journal of Neurophysiology*, vol. 64, no. 3, pp. 745-755, 1990, doi: 10.1152/jn.1990.64.3.745. [Online]. Available:

<https://doi.org/10.1152/jn.1990.64.3.745>

[32] R. Rodrigo and R. Murillo Aranda, "CONTROL OF A QUADRUPED LEG USING AN ANALYTICAL APPROACH," Thesis for obtaining the degree of Master in Optomechatronics, 2022, doi: 10.13140/RG.2.2.29418.24006. [Online]. Available: https://www.researchgate.net/publication/357791580_CONTROL_OF_A_QUADRUPED_LEG_USING_AN_ANALYTICAL_APPROACH_Thesis_for_obtaining_the_degree_of_Master_in_Optomechatronics

[33] M. M. U. Atique and M. A. R. Ahad, "Inverse Kinematics solution for a 3DOF robotic structure using Denavit-Hartenberg Convention," in *2014 International Conference on Informatics, Electronics & Vision (ICIEV)*, Dhaka, Bangladesh, 2014, pp. 1-5, doi: 10.1109/ICIEV.2014.6850854. [Online]. Available:

<https://ieeexplore.ieee.org/abstract/document/6850854>

- [34] J. Carlo, "Software and control design for the MIT Cheetah quadruped robots," 2020. [Online]. Available: https://www.researchgate.net/publication/349476726_Software_and_control_design_for_the_MIT_Cheetah_quadruped_robots
- [35] C. Scharzenberger, J. Mendoza, and A. Hunt, "Design of a Canine Inspired Quadruped Robot as a Platform for Synthetic Neural Network Control," in *Martinez-Hernandez, U., et al. Biomimetic and Biohybrid Systems. Living Machines 2019*, Lecture Notes in Computer Science, vol. 11556, Springer, Cham, 2019, doi: 10.1007/978-3-030-24741-6_20. [Online]. Available: https://doi.org/10.1007/978-3-030-24741-6_20
- [36] K. Abdel-Malek and S. Othman, "Multiple sweeping using the Denavit–Hartenberg representation method," **Computer-Aided Design**, vol. 31, no. 9, pp. 1999, 1999, doi: 10.1016/S0010-4485(99)00053-6. [Online]. Available: [https://doi.org/10.1016/S0010-4485\(99\)00053-6](https://doi.org/10.1016/S0010-4485(99)00053-6)
- [37] M. W. Spong, S. Hutchinson, and M. Vidyasagar, **Robot Modeling and Control**. John Wiley & Sons, Inc., 2006. [Online]. Available: https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://www.researchgate.net/profile/Mohamed_Mourad_Lafifi/post/How_to_avoid_singular_configurations/attachment/59d6361b79197b807799389a/AS%253A386996594855942%25401469278586939/download/Spong%2B-%2BRobot%2Bmodeling%2Band%2BControl.pdf&ved=2ahUKEwikwOnT2eyFAxXvREDHczCBB0QFnoECBEQAQ&usq=A0vVaw0SmJvmoTWd1k0O3PyRfDSIhttps://w

www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://www.researchgate.net/profile/Mohamed_Mourad_Lafifi/post/How_to_avoid_singular_configurations/attachment/59d6361b79197b807799389a/AS%253A386996594855942%25401469278586939/download/Spong%2B-%2BRobot%2Bmodeling%2Band%2BControl.pdf&ved=2ahUKEwikwOnT2eyFAxXvR_EDHczCBB0QFnoECBEQAQ&usq=A0vVaw0SmJvmoTWd1k0O3PyRfDSI

[38] M. Z. Huang, S.-H. Ling, and Y. Sheng, "A study of velocity kinematics for hybrid manipulators with parallel-series configurations," in *[1993] Proceedings IEEE International Conference on Robotics and Automation*, Atlanta, GA, USA, 1993, pp. 456-461 vol.1, doi: 10.1109/ROBOT.1993.292022. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/292022>

[39] A. Kelly and N. Seegmiller, "A Vector Algebra Formulation of Mobile Robot Velocity Kinematics," in *Springer Tracts in Advanced Robotics*, vol. 92, pp. 613-627, 2014, doi: 10.1007/978-3-642-40686-7_41. [Online]. Available: https://www.researchgate.net/publication/288437144_A_Vector_Algebra_Formulation_of_Mobile_Robot_Velocity_Kinematics

[40] E. Taşkıran, M. Yılmaz, Ö. Koca, U. Seven, and K. Erbatur, "Trajectory generation with natural ZMP references for the biped walking robot SURALP," in *2010 IEEE International Conference on Robotics and Automation*, Anchorage, AK, USA, 2010, pp. 4237-4242, doi: 10.1109/ROBOT.2010.5509792. [Online]. Available: <https://ieeexplore.ieee.org/document/5509792>

- [41] Z. Li, S. S. Ge, and S. Liu, "Contact-Force Distribution Optimization and Control for Quadruped Robots Using Both Gradient and Adaptive Neural Networks," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 8, pp. 1460-1473, Aug. 2014, doi: 10.1109/TNNLS.2013.2293500. [Online]. Available: <https://ieeexplore.ieee.org/document/6687263>
- [42] M. Hutter et al., "ANYmal - a highly mobile and dynamic quadrupedal robot," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, Korea (South), 2016, pp. 38-44, doi: 10.1109/IROS.2016.7758092. [Online]. Available: <https://ieeexplore.ieee.org/document/7758092>
- [43] G. Bledt, M. J. Powell, B. Katz, J. Di Carlo, P. M. Wensing, and S. Kim, "MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadruped Robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, 2018, pp. 2245-2252, doi: 10.1109/IROS.2018.8593885. [Online]. Available: <https://ieeexplore.ieee.org/document/8593885>
- [44] D. Kim, J. D. Carlo, B. Katz, G. Bledt, and S. Kim, "Highly Dynamic Quadruped Locomotion via Whole-Body Impulse Control and Model Predictive Control," *ArXiv*, abs/1909.06586, 2019. [Online]. Available: <https://www.semanticscholar.org/paper/Highly-Dynamic-Quadruped-Locomotion-via-Whole-Body-Kim-Carlo/90e8e65532a647074fa705e8c84d70fa0e1af266>
- [45] S. Fahmi, C. Mastalli, M. Focchi, and C. Semini, "Passive Whole-Body Control for Quadruped Robots: Experimental Validation Over Challenging Terrain," *IEEE Robotics*

and Automation Letters, vol. 4, no. 3, pp. 2553-2560, July 2019. [Online]. Available: <https://ieeexplore.ieee.org/document/8678400>. doi: 10.1109/LRA.2019.2908502.

[46] Y. Ogura and T. Komuro, "The Simplest Balance Controller for Dynamic Walking," 2022. [Online]. Available: https://www.researchgate.net/publication/365349123_The_Simplest_Balance_Controller_for_Dynamic_Walking.

[47] T. Joseph, A. Shaikh, M. Sarode, and Y. Srinivasa Rao, "Quadruped Robots: Gait Analysis and Control," *2020 IEEE 17th India Council International Conference (INDICON)*, 2020. [Online]. Available: <https://ieeexplore.ieee.org/document/9342521>. doi: 10.1109/INDICON49873.2020.9342521.

[48] A. Tuleu, "Hardware, software and control design considerations towards low-cost compliant quadruped robots," 2016. [Online]. Available: <https://www.semanticscholar.org/paper/Hardware%2C-software-and-control-design-towards-Tuleu/d142aaae1f17de6a6cb2a511e37df6d2b5e3ebcf>.

[49] M. M. U. Atique, M. R. I. Sarker, and M. A. R. Ahad, "Development of an 8DOF quadruped robot and implementation of Inverse Kinematics using Denavit-Hartenberg convention," *Heliyon*, vol. 4, no. 12, p. e01053, 2018. [Online]. Available: <https://doi.org/10.1016/j.heliyon.2018.e01053>.

[50] Y. Gao, "Towards the Design and Evaluation of Robotic Legs of Quadruped Robots," 2018. [Online]. Available: <https://www.semanticscholar.org/paper/Towards-the-Design-and-Evaluation-of-Robotic-Legs-Gao/e55b9f9bb6590b4909f94e098266afc454b31194>

- [51] E. Yu, "Design and Prototyping of a Transformable Quadruped Robot,"
Undergraduate Honors Thesis, Dept. of Mech. Eng., Ohio State Univ., Columbus, OH,
USA. [Online]. Available: <http://kb.osu.edu/handle/1811/84714>.
- [52] Z. Li, "Mechanical Design, Analysis and Simulation of a Quadruped Robot,"
Undergraduate Honors Thesis, Dept. of Mech. Eng., Ohio State Univ., Columbus, OH,
USA, 2019. [Online]. Available: <https://core.ac.uk/download/196228673.pdf>.

APPENDICES

Appendix 1: MATLAB Controller Code (Trot Gait)

```
function [tgt_ctrl_fl,tgt_ctrl_fr,tgt_ctrl_rl,tgt_ctrl_rr, ctrl_state, debug]
= ...
    controller(fl_ang, fr_ang, rl_ang, rr_ang, ...
        contact_state, body_R, sensing_info, t, body, planner)

% the sample time of the block is set to be 0.0025 (400Hz);
dt = 0.0025;
%unpack sensing_info
body_vx = sensing_info(1);
body_vy = sensing_info(2);

fl_ang_s = fl_ang(1); fl_ang_u = fl_ang(2); fl_ang_k = fl_ang(3);
fr_ang_s = fr_ang(1); fr_ang_u = fr_ang(2); fr_ang_k = fr_ang(3);
rl_ang_s = rl_ang(1); rl_ang_u = rl_ang(2); rl_ang_k = rl_ang(3);
rr_ang_s = rr_ang(1); rr_ang_u = rr_ang(2); rr_ang_k = rr_ang(3);
% forward kinematics
fl_end_effector = fk_left(fl_ang_s, fl_ang_u, fl_ang_k, body);
rl_end_effector = fk_left(rl_ang_s, rl_ang_u, rl_ang_k, body);
fr_end_effector = fk_right(fr_ang_s, fr_ang_u, fr_ang_k, body);
rr_end_effector = fk_right(rr_ang_s, rr_ang_u, rr_ang_k, body);

% distance from center of mass to shoulder
fl_offset = [body.shoulder_distance;body.y_length/2+body.shoulder_size/2;0];
fr_offset = [body.shoulder_distance;-body.y_length/2-body.shoulder_size/2;0];
rl_offset = [-body.shoulder_distance;body.y_length/2+body.shoulder_size/2;0];
rr_offset = [-body.shoulder_distance;-body.y_length/2-body.shoulder_size/2;0];

%virtual leg1 fl (front left) rr(rear right) body frame
curr_virtual_leg1 = (fl_end_effector + fl_offset + rr_end_effector +
rr_offset)*0.5;
%virtual leg2 fr (front right) rl(rear left) body frame
curr_virtual_leg2 = (fr_end_effector + fr_offset + rl_end_effector +
rl_offset)*0.5;

%decide state and state transition helpers
persistent state;
persistent timer;
persistent timer2;
if isempty(state)
    state = 0;
end
if isempty(timer)
    timer = 0;
end
if isempty(timer2)
    timer2 = 0;
```

```

end

% init variables
    % fl fr r1 rr
has_u_ctrl = [1 1 1 1];
normal_virtual_leg1 = [0;0;planner.stand_height];
normal_virtual_leg2 = [0;0;planner.stand_height];
move_fl_end_effector = normal_virtual_leg1;
move_rr_end_effector = normal_virtual_leg1;
move_fr_end_effector = normal_virtual_leg2;
move_rl_end_effector = normal_virtual_leg2;

% state machine

if state == 0 % standby state
    % in this state, just do nothing, but use time to make sure that all feet
    stands on the ground
    if contact_state(1) == 1 && contact_state(2) == 1 && contact_state(3) == 1
    && contact_state(4) == 1
        %all foot on ground
        if abs(body_vx) < planner.state0_vel_thres
            timer = timer + 1;
        else
            timer = 0;
        end
        if timer > planner.state0_trans_thres
            state = 1;
            timer = 0;
        end
    end
elseif state == 1 % swing whole body to get a inital velocity
    timer2 = timer2 + 1;
    move_theta =
planner.state0_swing_ang*sin(2*pi/planner.state0_swing_T*timer2);
    move_leg_Rx = [cos(move_theta) 0 sin(move_theta);
        0 1 0;
        -sin(move_theta) 0 cos(move_theta)];
    tgt_virtual_leg1_b = move_leg_Rx*normal_virtual_leg1;
    tgt_virtual_leg2_b = move_leg_Rx*normal_virtual_leg2;
    move_fl_end_effector = tgt_virtual_leg1_b;
    move_rr_end_effector = tgt_virtual_leg1_b;
    move_fr_end_effector = tgt_virtual_leg2_b;
    move_rl_end_effector = tgt_virtual_leg2_b;

    if body_vx > planner.state12_trans_speed
        timer = timer + 1;
    else
        timer = 0;
    end
    if timer > 40 %(keep a velocity larger than 0.2 for 0.1s)
        state = 2;
    end
end

```

```

        timer = 0;
        timer2 = 0;
    end

elseif state == 2 % stance virtual leg1, swing virtual leg2
    timer2 = timer2 + 1;
    if timer2 > planner.leg_swing_time %(0.5s finish movement)
        timer2 = planner.leg_swing_time;
    end
    % stance leg no u control but need to balance the body
    % fl fr rl rr
    has_u_ctrl = [0 1 1 0];
    t = timer2/planner.leg_swing_time;
    % stance_length_coeff = 0.95+0.05*(timer2/planner.leg_swing_time);
    stance_length_coeff = 0.2*t^2 - 0.2*t + 1;
    tgt_virtual_leg1_b = stance_length_coeff*normal_virtual_leg1;
    move_fl_end_effector = tgt_virtual_leg1_b;
    move_rr_end_effector = tgt_virtual_leg1_b;

    % swing leg must consider body angle
    length_coeff = 0.8*t^2-0.8*t+1;
    % move leg in x direction
    move_theta = get_move_angle(length_coeff*normal_virtual_leg2,body_vx,
planner.tgt_body_vx, planner);
    move_leg_Rx = [cos(move_theta) 0 sin(move_theta);
        0 1 0;
        -sin(move_theta) 0 cos(move_theta)];
    tgt_virtual_leg2_b = length_coeff*move_leg_Rx*normal_virtual_leg2;
    % move leg in y direction
    sagittal_angle = get_sagittal_angle(tgt_virtual_leg2_b,body_R, body_vy,
planner);
    move_leg_Ry =[1 0 0;
        0 cos(sagittal_angle) -sin(sagittal_angle);
        0 sin(sagittal_angle) cos(sagittal_angle)];
    tgt_virtual_leg2_b = move_leg_Ry*tgt_virtual_leg2_b;

    move_fr_end_effector = body_R'*(tgt_virtual_leg2_b+fr_offset)-fr_offset;
    if norm(move_fr_end_effector) > body.max_stretch
        move_fr_end_effector =
move_fr_end_effector/norm(move_fr_end_effector)*body.max_stretch;
    end
    move_rl_end_effector = body_R'*(tgt_virtual_leg2_b+r1_offset)-r1_offset;
    if norm(move_rl_end_effector) > body.max_stretch
        move_rl_end_effector =
move_rl_end_effector/norm(move_rl_end_effector)*body.max_stretch;
    end

    % state transition

```

```

    if timer2 > 0.5*planner.leg_swing_time && contact_state(2) == 1 &&
contact_state(3) == 1
        timer = timer + 1;
    else
        timer = 0;
    end
    if timer > 5 % to be determined
        state = 3;
        timer = 0;
        timer2 = 0;
    end

elseif state == 3 % swing virtual leg1, stance virtual leg2
    timer2 = timer2 + 1;
    if timer2 > planner.leg_swing_time %(0.5s finish movement)
        timer2 = planner.leg_swing_time;
    end
    % stance leg no u control but need to balance the body
    % fl fr rl rr
    has_u_ctrl = [1 0 0 1];
    t = timer2/planner.leg_swing_time;
    stance_length_coeff = 0.2*t^2 - 0.2*t + 1;
    tgt_virtual_leg2_b = stance_length_coeff*normal_virtual_leg2;
    move_fr_end_effector = tgt_virtual_leg2_b;
    move_rl_end_effector = tgt_virtual_leg2_b;

    % swing leg must consider body angle
    length_coeff = 0.8*t^2-0.8*t+1;
    move_theta = get_move_angle(length_coeff*normal_virtual_leg1,body_vx,
planner.tgt_body_vx, planner);
    move_leg_Rx = [cos(move_theta) 0 sin(move_theta);
0 1 0;
-sin(move_theta) 0 cos(move_theta)];
    tgt_virtual_leg1_b = length_coeff*move_leg_Rx*normal_virtual_leg1;
    % move leg in y direction
    sagittal_angle = get_sagittal_angle(tgt_virtual_leg1_b,body_R, body_vy,
planner);
    move_leg_Ry =[1 0 0;
0 cos(sagittal_angle) -sin(sagittal_angle);
0 sin(sagittal_angle) cos(sagittal_angle)];
    tgt_virtual_leg1_b = move_leg_Ry*tgt_virtual_leg1_b;

    move_fl_end_effector = body_R'*(tgt_virtual_leg1_b+fl_offset)-fl_offset;
    if norm(move_fl_end_effector) > body.max_stretch
        move_fl_end_effector =
move_fl_end_effector/norm(move_fl_end_effector)*body.max_stretch;
    end
    move_rr_end_effector = body_R'*(tgt_virtual_leg1_b+rr_offset)-rr_offset;
    if norm(move_rr_end_effector) > body.max_stretch
        move_rr_end_effector =
move_rr_end_effector/norm(move_rr_end_effector)*body.max_stretch;

```

```

    end

    % state transition
    if timer2 > 0.5*planner.leg_swing_time && contact_state(1) == 1 &&
contact_state(4) == 1
        timer = timer + 1;
    else
        timer = 0;
    end
    if timer > 5 % to be determined
        state = 2;
        timer = 0;
        timer2 = 0;
    end
end

%% fl to joint
[inv_s,inv_u,inv_k] = IK_left(move_fl_end_effector, body);
fl_tgt_ang_s = inv_s;
fl_tgt_ang_u = inv_u;
fl_tgt_ang_k = inv_k;
tgt_ctrl_fl = [fl_tgt_ang_s;fl_tgt_ang_u;fl_tgt_ang_k;has_u_ctrl(1)];

%% fr to joint
[inv_s,inv_u,inv_k] = IK_right(move_fr_end_effector, body);
fr_tgt_ang_s = inv_s;
fr_tgt_ang_u = inv_u;
fr_tgt_ang_k = inv_k;
tgt_ctrl_fr = [fr_tgt_ang_s;fr_tgt_ang_u;fr_tgt_ang_k;has_u_ctrl(2)];

%% rl to joint
[inv_s,inv_u,inv_k] = IK_left(move_rl_end_effector, body);
rl_tgt_ang_s = inv_s;
rl_tgt_ang_u = inv_u;
rl_tgt_ang_k = inv_k;
tgt_ctrl_rl = [rl_tgt_ang_s;rl_tgt_ang_u;rl_tgt_ang_k;has_u_ctrl(3)];

%% rr to joint
[inv_s,inv_u,inv_k] = IK_right(move_rr_end_effector, body);
rr_tgt_ang_s = inv_s;
rr_tgt_ang_u = inv_u;
rr_tgt_ang_k = inv_k;
tgt_ctrl_rr = [rr_tgt_ang_s;rr_tgt_ang_u;rr_tgt_ang_k;has_u_ctrl(4)];

ctrl_state = state;
debug = timer;

end

function move_angle = get_move_angle(virtual_leg,body_vx, tgt_body_vx,
planner)

```

```

distance = body_vx* planner.Ts/2 + planner.Kv*(tgt_body_vx-body_vx);

if distance > 0.5*norm(virtual_leg)
    distance = 0.5*norm(virtual_leg);
elseif distance < -0.5*norm(virtual_leg)
    distance = -0.5*norm(virtual_leg);
end

move_angle = asin(distance/norm(virtual_leg));
if move_angle > 20/180*pi
    move_angle = 20/180*pi;
elseif move_angle < -20/180*pi
    move_angle = -20/180*pi;
end

end

function move_angle = get_sagittal_angle(virtual_leg,body_R, body_vy, planner)

distance = body_vy* planner.y_Ts/2 + planner.y_Kv*(0-body_vy);

if distance > 0.5*norm(virtual_leg)
    distance = 0.5*norm(virtual_leg);
elseif distance < -0.5*norm(virtual_leg)
    distance = -0.5*norm(virtual_leg);
end

move_angle = asin(distance/norm(virtual_leg));
if move_angle > 20/180*pi
    move_angle = 20/180*pi;
elseif move_angle < -20/180*pi
    move_angle = -20/180*pi;
end

end

```


Appendix 2: MATLAB Variable Initialization:

```
% Add library path for robot control utilities
addpath(genpath(fullfile(pwd, 'contact_library')));

% Define ground plane properties for simulation
groundPlane.stiffness = 50000; % N/m
groundPlane.damping = 1000; % Ns/m
groundPlane.surfaceHeight = 0.4; % meters

% Set the physical dimensions of the robot's body
robotBody.length = 0.3; % meters along the x-axis
robotBody.width = 0.1; % meters along the y-axis
robotBody.height = 0.027; % meters along the z-axis
robotBody.shoulderDiameter = 0.027;
robotBody.upperArmLength = 0.1105;
robotBody.forearmLength = 0.115;
robotBody.footDiameter = 0.03; % radius of foot in meters
robotBody.distanceBetweenShoulders = 0.1365;
robotBody.maximumLimbExtension = robotBody.upperArmLength +
robotBody.forearmLength;
robotBody.jointDamping = 0.1; % damping coefficient at the knee joint

% Parameters for leg motion control using PID regulators
motionControl.positionP = 8; % Proportional gain for position
motionControl.positionI = 0; % Integral gain for position
motionControl.positionD = 0; % Derivative gain for position
motionControl.velocityP = 1.2; % Proportional gain for velocity
motionControl.velocityI = 0; % Integral gain for velocity
motionControl.velocityD = 0; % Derivative gain for velocity

% High-level planning parameters for leg movement
movementPlanner.touchdownHeight = robotBody.footDiameter / 2; % Distance from
foot center to ground
movementPlanner.initialShoulderAngle = 0;
movementPlanner.upperArmAngle = 37.91 * pi / 180; % Radians
movementPlanner.forearmAngle = -74.1 * pi / 180; % Radians
currentPosition = forwardKinematics_left(movementPlanner.initialShoulderAngle,
movementPlanner.upperArmAngle, movementPlanner.forearmAngle, robotBody);
movementPlanner.standingHeight = currentPosition(3);
movementPlanner.flyingHeight = 1.1 * movementPlanner.standingHeight; %
Elevated height during leg flight

% Gait control parameters for locomotion
gaitControl.cycleDuration = 3; % Time for one complete gait cycle in seconds
gaitControl.swingAngle = 12 * pi / 180; % Radians
gaitControl.initialShakeAngle = 3 * pi / 180; % Radians

% Target orientations and velocities for gait control
gaitControl.targetBodyAngle = 0; % Radians
gaitControl.targetBodyVelocityX = 0.4; % Velocity in m/s
```

```
gaitControl.legPlacementTimeX = 0.1; % Time to place leg in the x-direction
gaitControl.xDirectionVelocityCoefficient = 0.5; % Coefficient for x-
direction leg placement
gaitControl.legPlacementTimeY = 0.21; % Time to place leg in the y-direction
gaitControl.yDirectionVelocityCoefficient = -0.34; % Coefficient for y-
direction leg placement

% Define state transition thresholds for control logic
stateTransitions.velocityThreshold = 0.025; % m/s
stateTransitions.transitionTime = 300; % Time threshold for transitioning
states
stateTransitions.swingAngleThreshold = 10 * pi / 180; % Radians
stateTransitions.swingTime = 1200; % Time to complete a leg swing
stateTransitions.transitionSpeedThreshold = 0.05; % Speed threshold for state
transitions
stateTransitions.legSwingDuration = 70; % Time to generate leg motion in
milliseconds
```

Appendix 3: Arduino Code (Balancing)

3.1 Maincode

```
#include <Arduino.h>
#include <Servo.h>
#include <Wire.h>
#include "Structures.h"
#include "functions.h"
#include "classes.h"
#include <Adafruit_PWMServoDriver.h>
#include <math.h>
#include "SimpleMPU6050.h"

#define MPU6050_DEFAULT_ADDRESS 0x68

SimpleMPU6050 mpu;

unsigned long timer = 0;
float delT = 50;

CONSTRUCTBODY Body;
CONSTRUSTLEG LegFR;
CONSTRUSTLEG LegFL;
CONSTRUSTLEG LegBR;
CONSTRUSTLEG LegBL;
Adafruit_PWMServoDriver Servodriver = Adafruit_PWMServoDriver(0x40);

PID_controller myPIDController(0.35, 0.2, 0);

tilt_Vector tilt;

void MPUgetAngle(int16_t *gyro, int16_t *accel, int32_t *quat) {
    Quaternion q;
    VectorFloat gravity;
    float ypr[3] = {0, 0, 0};
    float xyz[3] = {0, 0, 0};
    mpu.GetQuaternion(&q, quat);
    mpu.GetGravity(&gravity, &q);
    mpu.GetYawPitchRoll(ypr, &q, &gravity);
    mpu.ConvertToDegrees(ypr, xyz);

    tilt.roll = xyz[1];
}
```

```

tilt.pitch = xyz[2];
tilt.yaw = xyz[0]; // yaw has a lot of drift
tilt.yaw = 0;
}

xyz angleToPWM(CONSTRUSTLEG *Leg, xyz ServoAngles) {
    xyz PWM_value;
    PWM_value.x = Leg->conversion[0] * ServoAngles.x;
    PWM_value.y = Leg->conversion[1] * ServoAngles.y;
    PWM_value.z = Leg->conversion[2] * ServoAngles.z;

    return PWM_value;
}

xyz setServoAng(CONSTRUSTLEG *Leg, xyz Ang_theta, Adafruit_PWMServoDriver
Servodriver) {
    bool RightLeg = (Leg->legID == 2) || (Leg->legID == 4);
    bool FrontLeg = (Leg->legID == 1) || (Leg->legID == 2);
    xyz ServoAngles;

    if (!RightLeg) {
        if (Leg->theta1OverYaxis) {
            if (FrontLeg) ServoAngles.x = 90 - (Ang_theta.x), ServoAngles.y =
300 - Ang_theta.y, ServoAngles.z = 180 - Ang_theta.z;
            else ServoAngles.x = 90 - (-Ang_theta.x), ServoAngles.y = 300 -
Ang_theta.y, ServoAngles.z = 180 - Ang_theta.z;
        }
        else {
            if (FrontLeg) ServoAngles.x = 90 + (Ang_theta.x), ServoAngles.y =
300 - Ang_theta.y, ServoAngles.z = 180 - Ang_theta.z;
            else ServoAngles.x = 90 + (-Ang_theta.x), ServoAngles.y = 300 -
Ang_theta.y, ServoAngles.z = 180 - Ang_theta.z;
        }
    }
    else if (RightLeg) {
        if (Leg->theta1OverYaxis) {
            if (FrontLeg) ServoAngles.x = 90 + (Ang_theta.x);
            else ServoAngles.x = 90 + (-Ang_theta.x);
        }
        else {
            if (FrontLeg) ServoAngles.x = 90 - (Ang_theta.x);
            else ServoAngles.x = 90 - (-Ang_theta.x);
        }
    }
}

```

```

    }
    ServoAngles.y = Ang_theta.y - 120;
    ServoAngles.z = Ang_theta.z;
}

xyz servoAnglesPWM = angleToPWM(Leg, ServoAngles);

xyz PWM;
PWM.x = Leg->Servo_MIN_PWM[0] + servoAnglesPWM.x;
PWM.y = Leg->Servo_MIN_PWM[1] + servoAnglesPWM.y;
PWM.z = Leg->Servo_MIN_PWM[2] + servoAnglesPWM.z;

if (PWM.x < Leg->safe_PWM_MAX[0] && PWM.x > Leg->safe_PWM_MIN[0]) {
    return PWM.x = PWM.x;
}
if (PWM.y < Leg->safe_PWM_MAX[1] && PWM.y > Leg->safe_PWM_MIN[1]) {
    return PWM.y = PWM.y;
}
if (PWM.z < Leg->safe_PWM_MAX[2] && PWM.z > Leg->safe_PWM_MIN[2]) {
    return PWM.z = PWM.z;
}

return PWM;
}

void setup() {
    Serial.begin(115200);
    Wire.begin();
    Servodriver.begin();
    Servodriver.setPWMFreq(60);

    Body.final_rotation.pitch = 0;
    Body.final_rotation.roll = 0;
    Body.final_rotation.yaw = 0;

    tilt.roll = 0;
    tilt.pitch = 0;
    tilt.yaw = 0;

    Serial.println("Startup delay 2 seconds: ");
    delay(2000);
}

```

```

constructBody(&Body, 0.2075, 0.13, 0.15, 0.03, 0.105, 0.14);
constructLeg(&LegFR, &Body, 2);
constructLeg(&LegFL, &Body, 1);
constructLeg(&LegBR, &Body, 4);
constructLeg(&LegBL, &Body, 3);

xyz InputPWM[4];
InputPWM[0] = setServoAng(&LegFL, legIK(&LegFL, &Body, tilt),
&Servodriver);
InputPWM[1] = setServoAng(&LegFR, legIK(&LegFR, &Body, tilt),
&Servodriver);
InputPWM[2] = setServoAng(&LegBL, legIK(&LegBL, &Body, tilt),
&Servodriver);
InputPWM[3] = setServoAng(&LegBR, legIK(&LegBR, &Body, tilt),
&Servodriver);

Servodriver.setPWM(LegBL.ServoID[0], 0, InputPWM[2].x);
Servodriver.setPWM(LegBL.ServoID[1], 0, InputPWM[2].y);
Servodriver.setPWM(LegBL.ServoID[2], 0, InputPWM[2].z);

Servodriver.setPWM(LegBR.ServoID[0], 0, InputPWM[3].x);
Servodriver.setPWM(LegBR.ServoID[1], 0, InputPWM[3].y);
Servodriver.setPWM(LegBR.ServoID[2], 0, InputPWM[3].z);

Servodriver.setPWM(LegFL.ServoID[0], 0, InputPWM[0].x);
Servodriver.setPWM(LegFL.ServoID[1], 0, InputPWM[0].y);
Servodriver.setPWM(LegFL.ServoID[2], 0, InputPWM[0].z);

Servodriver

.setPWM(LegFR.ServoID[0], 0, InputPWM[1].x);
Servodriver.setPWM(LegFR.ServoID[1], 0, InputPWM[1].y);
Servodriver.setPWM(LegFR.ServoID[2], 0, InputPWM[1].z);
}

void loop() {
  unsigned long now = millis();
  if ((now - timer) > delT) {
    int16_t accel[3], gyro[3];
    int32_t quat[4];
    float angle[3];

```

```

MPUgetAngle(gyro, accel, quat);

tilt.roll = (tilt.roll + gyro[0] * delT / 1000);
tilt.pitch = (tilt.pitch + gyro[1] * delT / 1000);
tilt.yaw = (tilt.yaw + gyro[2] * delT / 1000);

xyz InputPWM[4];
InputPWM[0] = setServoAng(&LegFL, legIK(&LegFL, &Body, tilt),
&Servodriver);
InputPWM[1] = setServoAng(&LegFR, legIK(&LegFR, &Body, tilt),
&Servodriver);
InputPWM[2] = setServoAng(&LegBL, legIK(&LegBL, &Body, tilt),
&Servodriver);
InputPWM[3] = setServoAng(&LegBR, legIK(&LegBR, &Body, tilt),
&Servodriver);

Servodriver.setPWM(LegBL.ServoID[0], 0, InputPWM[2].x);
Servodriver.setPWM(LegBL.ServoID[1], 0, InputPWM[2].y);
Servodriver.setPWM(LegBL.ServoID[2], 0, InputPWM[2].z);

Servodriver.setPWM(LegBR.ServoID[0], 0, InputPWM[3].x);
Servodriver.setPWM(LegBR.ServoID[1], 0, InputPWM[3].y);
Servodriver.setPWM(LegBR.ServoID[2], 0, InputPWM[3].z);

Servodriver.setPWM(LegFL.ServoID[0], 0, InputPWM[0].x);
Servodriver.setPWM(LegFL.ServoID[1], 0, InputPWM[0].y);
Servodriver.setPWM(LegFL.ServoID[2], 0, InputPWM[0].z);

Servodriver.setPWM(LegFR.ServoID[0], 0, InputPWM[1].x);
Servodriver.setPWM(LegFR.ServoID[1], 0, InputPWM[1].y);
Servodriver.setPWM(LegFR.ServoID[2], 0, InputPWM[1].z);

FIFO_Delay_timer = millis();
}
}

```

3.2 Structures.h

```

#ifndef STRUCTURES_H
#define STRUCTURES_H

```

```

struct tilt_Vector {
    float roll;
    float pitch;
    float yaw;
};

struct xyz {
    float x;
    float y;
    float z;
};

struct CONSTRUCTBODY {
    float bodywidth;
    float bodylength;
    float bodyheight;

    float Link1;
    float Link2;
    float Link3;

    tilt_Vector final_rotation;
};

struct CONSTRUCTLEG {
    xyz leg_Origin;
    xyz foot_Pos;

    short LegID;

    int ServoID[3];

    float Angles[3];

    float Conversion[3];
    float Servo_MIN_PWM[3];
    float Neutral_Zero_PWM[3];

    bool theta1OverYaxis;
    float safe_PWM_MAX[3];
    float safe_PWM_MIN[3];
};

```



```
};  
  
#endif
```

3.3 functions.h

```
#ifndef FUNCTIONS_H  
#define FUNCTIONS_H  
  
#include "Structures.h"  
  
void LegGeometry(CONSTRUCTLEG *Leg, CONSTRUCTBODY *Body, int LegID);  
  
void BodyGeometry(CONSTRUCTBODY *Body, float bodylength, float bodywidth,  
float bodyheight, float Link1, float Link2, float Link3);  
  
xyz x_rotate(xyz PointCoord, tilt_Vector Tilt);  
  
xyz y_rotate(xyz PointCoord, tilt_Vector Tilt);  
  
xyz z_rotate(xyz PointCoord, tilt_Vector Tilt);  
  
xyz R_Matrix(xyz PointCoord, tilt_Vector Tilt);  
  
xyz T_Matrix(xyz PointCoord, float theta1);  
  
xyz IK(CONSTRUSTLEG *Leg, CONSTRUCTBODY *Body, tilt_Vector Tilt);  
  
#endif
```

3.4 Classes.h

```
#ifndef CLASSES_H  
#define CLASSES_H  
  
#include "Structures.h"
```

```

class PID_controller {
public:
    PID_controller(double kp, double ki, double kd);
    float calc(float ref, float read_value, float delT );

private:
    double kp, ki, kd;
    float previousError, SUM_error;
};

#endif // CLASSES_H

```

3.5 PID.cpp

```

#ifndef CLASSES_H
#define CLASSES_H

class PID_controller {
private:
    double kp;
    double ki;
    double kd;
    double previousError;
    double SUM_error;

public:
    PIDController_c(double kp, double ki, double kd) : kp(kp), ki(ki),
kd(kd), previousError(0), SUM_error(0) {}

    float calc(float ref, float read_value, float delT) {
        float error = ref - read_value;
        float derivativeError = (error - previousError) / delT;
        SUM_error += error * delT;
        float output = kp * error + ki * SUM_error + kd * derivativeError;
        previousError = error;
        return output;
    }
};

#endif // CLASSES_H

```

3.6 IK.cpp

```
//IK.cpp
#include "Structures.h"
#include <math.h>
#include "functions.h"
#include <Arduino.h>

// Calculates angle B1 based on dx and dz' considering the quadrant
float B1Calculation(float dx, float dzPrim) {
    if (dx > 0 && dzPrim >= 0) {
        return atan(dzPrim / dx);
    } else if (dx == 0 && dzPrim >= 0) {
        return M_PI / 2;
    } else if (dx < 0) {
        return atan(dzPrim / dx) + (dzPrim >= 0 ? M_PI : 2 * M_PI);
    } else if (dx > 0 && dzPrim < 0) {
        return atan(dzPrim / dx) + 2 * M_PI;
    } else {
        return 3 * M_PI / 2;
    }
}

// Converts radians to degrees
float radToDegree(float Ang_radian) {
    return Ang_radian * 180 / M_PI;
}

// Rotation around the x-axis
xyz x_rotate(xyz PointCoord, tilt_Vector Tilt) {
    float x = PointCoord.x;
    float y = PointCoord.y * cos(Tilt.roll) - PointCoord.z * sin(Tilt.roll);
    float z = PointCoord.y * sin(Tilt.roll) + PointCoord.z * cos(Tilt.roll);
    return xyz{x, y, z};
}

// Rotation around the y-axis
xyz y_rotate(xyz PointCoord, tilt_Vector Tilt) {
```

```

    float x = PointCoord.x * cos(Tilt.pitch) + PointCoord.z *
sin(Tilt.pitch);
    float y = PointCoord.y;
    float z = -PointCoord.x * sin(Tilt.pitch) + PointCoord.z *
cos(Tilt.pitch);
    return xyz{x, y, z};
}

// Rotation around the z-axis
xyz z_rotate(xyz PointCoord, tilt_Vector Tilt) {
    float x = PointCoord.x * cos(Tilt.yaw) - PointCoord.y * sin(Tilt.yaw);
    float y = PointCoord.x * sin(Tilt.yaw) + PointCoord.y * cos(Tilt.yaw);
    float z = PointCoord.z;
    return xyz{x, y, z};
}

// Applies rotations around all axes to a PointCoord
xyz R_Matrix(xyz PointCoord, tilt_Vector Tilt) {
    // Convert Tilt angles from degrees to radians
    Tilt.roll *= M_PI / 180;
    Tilt.pitch *= M_PI / 180;
    Tilt.yaw *= M_PI / 180;

    // Apply rotations
    PointCoord = x_rotate(PointCoord, Tilt);
    PointCoord = y_rotate(PointCoord, Tilt);
    PointCoord = z_rotate(PointCoord, Tilt);

    return PointCoord;
}

// Transformation matrix given a rotation around the z-axis
xyz T_Matrix(xyz PointCoord, float theta1) {
    xyz result;
    result.x = PointCoord.x;
    result.y = PointCoord.y * cos(theta1) - PointCoord.z * sin(theta1);
    result.z = PointCoord.y * sin(theta1) + PointCoord.z * cos(theta1);
    return result;
}

// Inverse kinematics for a Leg
xyz IK(CONSTRUCTLEG *Leg, CONSTRUCTBODY *Body, tilt_Vector Tilt) {

```

```

// Check if the Leg is front or back and if it is right or left (viewed
from the back).
bool theta1OverYaxis;
bool RightLeg = (Leg->LegID == 2) || (Leg->LegID == 4);
bool FrontLeg = (Leg->LegID == 1) || (Leg->LegID == 2);

// Transform foot position based on Body Tilt
xyz foot_Pos = R_Matrix(Leg->foot_Pos, Tilt);

// Calculate deltas between foot and Leg origin
xyz Delta;
Delta.x = foot_Pos.x - Leg->legOrigin.x;
Delta.y = foot_Pos.y - Leg->legOrigin.y;
Delta.z = foot_Pos.z - Leg->legOrigin.z;

// Calculate DX, DY, DZ
float DX = fabs(Delta.x);
float DY = fabs(Delta.y);
float DZ = fabs(Delta.z);

// Theta1 calculation
float A = sqrt(pow(DY, 2) + pow(DZ, 2));
float beta1 = atan2(DY, DZ);
float alpha2 = asin(Body->link1 / A);
float alpha3 = M_PI / 2 - alpha2;
float theta1;

// Determine if theta1 is above or below the Y-axis
if (RightLeg) {
    if (Delta.y <= Body->link1) {
        theta1 = M_PI / 2 - alpha3 - beta1;
        theta1OverYaxis = false;
    } else {
        theta1 = alpha3 + beta1 - M_PI / 2;
        theta1OverYaxis = true;
    }
} else { // if it's left Leg
    if (Delta.y >= -Body->link1) {
        theta1 = M_PI / 2 - alpha3 - beta1;
        theta1OverYaxis = false;
    } else {
        theta1 = alpha3 + beta1 - M_PI / 2;

```

```

        theta1OverYaxis = true;
    }
}

// Find J2 coordinate with regard to the Leg origin
xyz J2;
J2.x = 0;
J2.y = Body->link1 * cos(theta1) * (RightLeg ? 1 : -1);
J2.z = Body->link1 * sin(theta1) * (theta1OverYaxis ? 1 : -1);

// Calculate J4J2 Delta vector
xyz J4J2;
J4J2.x = Delta.x - J2.x;
J4J2.y = Delta.y - J2.y;
J4J2.z = Delta.z - J2.z;

// Transform the J4J2 vector to the coordinate system xz' tilted by
theta1
xyz latestJ4J2 = T_Matrix(J4J2, theta1OverYaxis ? theta1 : -theta1);

// Theta2 and Theta3 calculation
float B = sqrt(pow(latestJ4J2.z, 2) + pow(latestJ4J2.x, 2));
float b1 = B1Calculation(latestJ4J2.x, latestJ4J2.z);
float b2 = acos((pow(Body->link2, 2) + pow(B, 2) - pow(Body->link3, 2))
/ (2 * Body->link2 * B));
float b3 = acos((pow(Body->link2, 2) + pow(Body->link3, 2) - pow(B, 2))
/ (2 * Body->link2 * Body->link3));

//4 bar linkage transformation
float thetaFollower = b1 - b2;
float gamma = theta2 - M_PI/2 //gamma is the angle in the triangle
used to calculate ground
float l4 = sqrt(a*a + b*b - 2*a*b*cos(gamma)); //l4 is ground length
//l1 = , l2= , l3=
float l1_3 = sqrt(l3*l3 + l4*l4 - 2*l3*l4*cos(thetaFollower));
float cosBeta = (l2*l2 + l4*l4 - l3*l3) / (2 * l2 * l4);
float beta = acos(cosBeta);
float cosThetaCrank = (l1*l1 + l4*l4 - l1_3*l1_3) / (2 * l1 * l4);
float theta2 = acos(cosThetaCrank);

float theta3 = M_PI - b3;

```

```

    // Convert calculated radians to degrees
    xyz Ang_theta;
    Ang_theta.x = radToDegree(theta1);
    Ang_theta.y = radToDegree(theta2);
    Ang_theta.z = radToDegree(theta3);

    return Ang_theta;
}

```

3.7 constructBody.cpp

```

//constructBody.cpp

#include "Structures.h"

void constructBody(CONSTRUCTBODY *Body, float bodylength, float bodywidth,
float bodyheight, float Link1, float Link2, float Link3) {
    Body->bodylength = bodylength;
    Body->bodywidth = bodywidth;
    Body->bodyheight = bodyheight;

    Body->Link1 = Link1;
    Body->Link2 = Link2;
    Body->Link3 = Link3;
}

```

3.8 constructLeg

```

//constructLeg.cpp

#include "Structures.h"
#include "functions.h"
#include <Arduino.h>

void constructLeg(CONSTRUCTLeg *Leg, CONSTRUCTBODY *Body, int LegID) {

```

```

float footOffset = 0.03;

Leg->LegID = LegID;

if (LegID == 1) { // left front Leg
    Leg->leg_Origin.x = Body->bodylength / 2;
    Leg->leg_Origin.y = Body->bodywidth / 2 * (-1);
    Leg->leg_Origin.z = 0;

    Leg->foot_Pos.x = Body->bodylength / 2;
    Leg->foot_Pos.y = Body->bodywidth / 2 * (-1) - footOffset;
    Leg->foot_Pos.z = Body->bodyheight * (-1);

    Leg->ServoID[0] = 0;
    Leg->ServoID[1] = 1;
    Leg->ServoID[2] = 2;

    Leg->Servo_MIN_PWM[0] = 84;
    Leg->Servo_MIN_PWM[1] = 84;
    Leg->Servo_MIN_PWM[2] = 84;

    Leg->Conversion[0] = 2.619;
    Leg->Conversion[1] = 2.619;
    Leg->Conversion[2] = 2.610;

    Leg->Neutral_Zero_PWM[0] = Leg->Servo_MIN_PWM[0] + 90 * Leg-
>Conversion[0];
    Leg->Neutral_Zero_PWM[1] = Leg->Servo_MIN_PWM[1] + 130 * Leg-
>Conversion[1];
    Leg->Neutral_Zero_PWM[2] = Leg->Servo_MIN_PWM[2] + 0 * Leg-
>Conversion[2];

    Leg->safe_PWM_MAX[0] = Leg->Servo_MIN_PWM[0] + 125 * Leg-
>Conversion[0];
    Leg->safe_PWM_MAX[1] = Leg->Servo_MIN_PWM[1] + 135 * Leg-
>Conversion[1];
    Leg->safe_PWM_MAX[2] = Leg->Servo_MIN_PWM[2] + 180 * Leg-
>Conversion[2];

    Leg->safe_PWM_MIN[0] = Leg->Servo_MIN_PWM[0] + 55 * Leg-
>Conversion[0];

```



```

    Leg->safe_PWM_MIN[1] = Leg->Servo_MIN_PWM[1] + 30 * Leg-
>Conversion[1];
    Leg->safe_PWM_MIN[2] = Leg->Servo_MIN_PWM[2];
}
// Continue the same pattern for other Leg IDs...
if (LegID == 2) { // right front Leg
    Leg->leg_Origin.x = Body->bodylength / 2;
    Leg->leg_Origin.y = Body->bodywidth / 2;
    Leg->leg_Origin.z = 0;

    Leg->foot_Pos.x = Body->bodylength / 2;
    Leg->foot_Pos.y = Body->bodywidth / 2 + footOffset;
    Leg->foot_Pos.z = Body->bodyheight * (-1);

    Leg->ServoID[0] = 3;
    Leg->ServoID[1] = 4;
    Leg->ServoID[2] = 5;

    Leg->Servo_MIN_PWM[0] = 84;
    Leg->Servo_MIN_PWM[1] = 84;
    Leg->Servo_MIN_PWM[2] = 84;

    Leg->Conversion[0] = 2.605;
    Leg->Conversion[1] = 2.610;
    Leg->Conversion[2] = 2.576;

    Leg->Neutral_Zero_PWM[0] = Leg->Servo_MIN_PWM[0] + 90 * Leg-
>Conversion[0];
    Leg->Neutral_Zero_PWM[1] = Leg->Servo_MIN_PWM[1] + 60 * Leg-
>Conversion[1];
    Leg->Neutral_Zero_PWM[2] = Leg->Servo_MIN_PWM[2] + 180 * Leg-
>Conversion[2];

    Leg->safe_PWM_MAX[0] = Leg->Servo_MIN_PWM[0] + 125 * Leg-
>Conversion[0];
    Leg->safe_PWM_MAX[1] = Leg->Servo_MIN_PWM[1] + 135 * Leg-
>Conversion[1];
    Leg->safe_PWM_MAX[2] = Leg->Servo_MIN_PWM[2] + 180 * Leg-
>Conversion[2];

    Leg->safe_PWM_MIN[0] = Leg->Servo_MIN_PWM[0] + 55 * Leg-
>Conversion[0];

```

```

    Leg->safe_PWM_MIN[1] = Leg->Servo_MIN_PWM[1] + 30 * Leg-
>Conversion[1];
    Leg->safe_PWM_MIN[2] = Leg->Servo_MIN_PWM[2];
}

if (LegID == 3) { // left back Leg
    Leg->leg_Origin.x = Body->bodylength / 2 * (-1);
    Leg->leg_Origin.y = Body->bodywidth / 2 * (-1);
    Leg->leg_Origin.z = 0;

    Leg->foot_Pos.x = Body->bodylength / 2 * (-1);
    Leg->foot_Pos.y = Body->bodywidth / 2 * (-1) - footOffset;
    Leg->foot_Pos.z = Body->bodyheight * (-1);

    Leg->ServoID[0] = 6;
    Leg->ServoID[1] = 7;
    Leg->ServoID[2] = 8;

    Leg->Servo_MIN_PWM[0] = 84;
    Leg->Servo_MIN_PWM[1] = 84;
    Leg->Servo_MIN_PWM[2] = 84;

    Leg->Conversion[0] = 2.595;
    Leg->Conversion[1] = 2.619;
    Leg->Conversion[2] = 2.614;

    Leg->Neutral_Zero_PWM[0] = Leg->Servo_MIN_PWM[0] + 90 * Leg-
>Conversion[0];
    Leg->Neutral_Zero_PWM[1] = Leg->Servo_MIN_PWM[1] + 120 * Leg-
>Conversion[1];
    Leg->Neutral_Zero_PWM[2] = Leg->Servo_MIN_PWM[2] + 0 * Leg-
>Conversion[2];

    Leg->safe_PWM_MAX[0] = Leg->Servo_MIN_PWM[0] + 125 * Leg-
>Conversion[0];
    Leg->safe_PWM_MAX[1] = Leg->Servo_MIN_PWM[1] + 150 * Leg-
>Conversion[1];
    Leg->safe_PWM_MAX[2] = Leg->Servo_MIN_PWM[2] + 180 * Leg-
>Conversion[2];

    Leg->safe_PWM_MIN[0] = Leg->Servo_MIN_PWM[0] + 55 * Leg-
>Conversion[0];

```

```

    Leg->safe_PWM_MIN[1] = Leg->Servo_MIN_PWM[1] + 30 * Leg-
>Conversion[1];
    Leg->safe_PWM_MIN[2] = Leg->Servo_MIN_PWM[2];
}

if (LegID == 4) { // right back Leg
    Leg->leg_Origin.x = Body->bodylength / 2 * (-1);
    Leg->leg_Origin.y = Body->bodywidth / 2;
    Leg->leg_Origin.z = 0;

    Leg->foot_Pos.x = Body->bodylength / 2 * (-1);
    Leg->foot_Pos.y = Body->bodywidth / 2 + footOffset;
    Leg->foot_Pos.z = Body->bodyheight * (-1);

    Leg->ServoID[0] = 9;
    Leg->ServoID[1] = 10;
    Leg->ServoID[2] = 11;

    Leg->Servo_MIN_PWM[0] = 85;
    Leg->Servo_MIN_PWM[1] = 84;
    Leg->Servo_MIN_PWM[2] = 85;

    Leg->Conversion[0] = 2.610;
    Leg->Conversion[1] = 2.725;
    Leg->Conversion[2] = 2.500;

    Leg->Neutral_Zero_PWM[0] = Leg->Servo_MIN_PWM[0] + (90 * Leg-
>Conversion[0]);
    Leg->Neutral_Zero_PWM[1] = Leg->Servo_MIN_PWM[1] + (60 * Leg-
>Conversion[1]);
    Leg->Neutral_Zero_PWM[2] = Leg->Servo_MIN_PWM[2] + (180 * Leg-
>Conversion[2]);

    Leg->safe_PWM_MAX[0] = Leg->Servo_MIN_PWM[0] + 125 * Leg-
>Conversion[0];
    Leg->safe_PWM_MAX[1] = Leg->Servo_MIN_PWM[1] + 150 * Leg-
>Conversion[1];
    Leg->safe_PWM_MAX[2] = Leg->Servo_MIN_PWM[2] + 180 * Leg-
>Conversion[2];

    Leg->safe_PWM_MIN[0] = Leg->Servo_MIN_PWM[0] + 55 * Leg-
>Conversion[0];

```

```
    Leg->safe_PWM_MIN[1] = Leg->Servo_MIN_PWM[1] + 30 * Leg-  
>Conversion[1];  
    Leg->safe_PWM_MIN[2] = Leg->Servo_MIN_PWM[2];  
  }  
}
```