Graduation Thesis

School of Engineering, Tohoku University

Self-Recognition-based of Modular Legged Robot

Department of Mechanical and Aerospace Engineering

Space Robotics Laboratory

C0TB1716 Tharit Sinsunthorn

(March, 2024)

Self-Recognition-based of Modular Legged Robot

Tharit Sinsunthorn

Abstract

In the expansive domain of lunar exploration, where the rugged lunar terrain and harsh environmental conditions present formidable challenges, the pursuit of innovative robotic solutions has intensified. Modular legged robots have emerged as a promising frontier, offering unparalleled adaptability and maneuverability in navigating the diverse lunar landscapes.

This thesis introduces "Moonbot", a modular legged robot to transcend the limitations of traditional lunar exploration missions. Developed as part of the prestigious Moonshot project led by the Space Robotics Laboratory at Tohoku University. Moonshot project has a strong aiming to develop a modular robot with the implementation of plug-and-play AI system for the propose of solar panels and lunar based construction. Henceforth, Moonbot stands as the pioneering prototype, embodying and extending the conceptual framework for robotic operations across lunar terrains. Its inception marks a pivotal moment in the evolution of robotic exploration, illuminating new pathways and possibilities for traversing and navigating the lunar landscape with unprecedented efficiency and adaptability.

Moonbot's innovative design is characterized by its self-recognition-based motion control capabilities, allowing dynamic adaptability to various leg configurations, from solitary to quadrupedal stances. Central to its design philosophy is a flexible structure, facilitated by magnetic connections between modular components, enabling rapid reconfiguration to meet evolving mission requirements.

Within the Moonshot project framework, Moonbot serves as a pioneering platform for testing and validating novel robotic technologies for lunar exploration and lunar base construction. Its modularity extends to independent leg controllers seamlessly integrated within the Robot Operating System 2 (ROS2) framework, ensuring seamless communication and precise control.

As Moonbot continues to undergo rigorous testing and refinement, it represents a significant leap forward in the quest for innovative robotic solutions for lunar exploration and beyond. With its adaptable design and robust capabilities, Moonbot embodies the spirit of exploration and innovation driving the Moonshot project, paving the way for future advancements in lunar robotics and space exploration.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the ambitious pursuit of lunar exploration, the traditional boundaries of robotics are being redefined. The quest for a robot capable of navigating the hazardous lunar environment has led to the development of Moonbot, a modular legged robot designed to transcend the limitations of conventional rover-based exploration. The essence of Moonbot lies in its ability to dynamically adapt and respond to the challenges presented by the lunar terrain.

1.1 Background

Lunar exploration has long captured the imagination of humanity. From the Apollo missions [1] of the 20th century to the more recent lunar rovers, our quest to uncover the secrets of the Moon has driven technological innovation and expanded our understanding of space.

As we look towards the future, the vision of establishing a permanent human presence on the Moon looms ever larger. Central to this vision is the construction of lunar bases, which will serve as hubs for scientific research, and resource utilization, and even as launch pads for missions deeper into space. However, the harsh and unforgiving lunar environment presents formidable challenges for such endeavors.

In the realm of space robotics research, modular robots, [3], [4] emerge as a transformative concept, reshaping our approach to space exploration. A modular robot is a versatile robotic system composed of interchangeable and reconfigurable modules that can be assembled in various configurations to adapt to different tasks and environments. Modular robots have been introduce to space robotics [5] marks a significant shift, providing unprecedented flexibility and adaptability for the challenges posed by extraterrestrial terrains.

Fig. 1.1: NASA' s Moon to Mars Autonomous Construction Technologies project to advance space-based construction capabilities for long-duration exploration missions on the Moon or Mars. [2]

In this thesis, we explore the development and capabilities of Moonbot, a modular legged robot designed for lunar exploration and base construction, following the target of Moonshot Project, Self-evolving AI Robot System for Lunar Exploration and Human Outpost Construction [6]. Through a combination of advanced control algorithms, Moonbot aims to push the boundaries of what is possible in extraterrestrial robotics. By harnessing the power of modular design and cutting-edge technology, we strive to pave the way for a new era of lunar exploration and human settlement.

1.2 Legged vs Wheeled

Researchers are increasingly considering a shift from wheeled to legged robotic system [7], particularly in environments where the terrain presents uncertainties. Although wheeled vehicles dominate exploration beyond Earth, this trend might be evolving. While wheels are effective on celestial bodies like the Moon and Mars, other challenging environments, such as smaller moons and asteroids, demand more adaptable solutions. In these low-gravity settings, robots navigating uneven terrain face the possibility of entering prolonged flight phases, a potentially hazardous situation for robots unprepared for such conditions. Below, we explore some of these advantages.

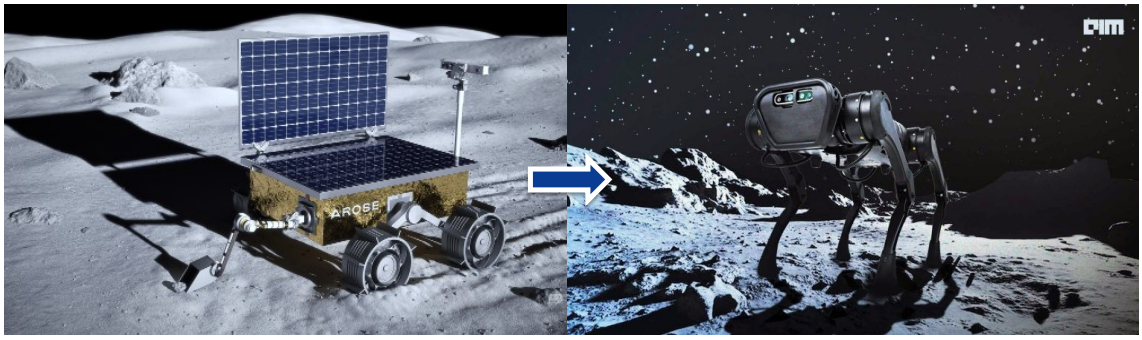


Fig. 1.2: Development of space rover and space legged robot.

1.2.1 Mobility

Legged robots offer superior mobility compared to wheeled robots due to their inherent omni-directional nature. This means that a legged robot can alter its direction independently of the main body axis simply by adjusting its footholds. In contrast, a traditional wheeled robot would need to perform specific maneuvers to change its direction.

Moreover, a legged robot can manipulate its body's movement and orientation while preserving footholds through adjustments in leg extension. This characteristic grants the robot an additional six degrees of freedom (DOF) for its body. It's important to note that while a wheeled robot with traction and directional motors can enhance its directionality, this often comes with the drawback of increased system complexity. Some robots also utilize specialized wheels, such as the ilonator wheel [8], to achieve omnidirectionality, but this is typically effective only on flat surfaces.

1.2.2 Overcoming Obstacles

A legged robot has the capability to surmount obstacles situated below its maximum ground clearance by stepping on them. In contrast, a wheeled robot can only navigate obstacles with heights up to half of its wheel radius [9]. Tracks, which are composed of a virtual wheel with a radius equal to half the track length, allow tracked vehicles to overcome higher obstacles compared to wheeled ones, albeit requiring substantial body motions.

1.2.3 Active Suspension

A legged robot inherently incorporates an active suspension system by adjusting the lengths of its legs in response to terrain irregularities. This capability enables a legged robot to traverse highly uneven terrain while maintaining a level body, ensuring smooth and comfortable motion for riders. In contrast, a wheeled robot keeps its body parallel to the terrain and experiences comparable tilts to the ground.

1.2.4 Natural Terrain

Wheeled vehicles demand costly, consistently paved surfaces for efficient movement. In contrast, legged systems, in principle, do not necessitate prepared terrain like their wheeled counterparts. They can navigate sandy, muddy, rigid, and soft terrains with comparable efficiency. Additionally, legged systems do not rely on continuous terrain for movement, providing another advantage.

1.2.5 Slippage and Jamming

Wheeled vehicles often encounter difficulties moving in soft terrain as their wheels tend to sink. In contrast, a leg, when placed vertically on the ground, only compresses the soft ground in a single direction. Vertical leg lifting avoids interference with the ground, and when the body is in motion, the feet rotate around their joints, preventing legs from interacting with the ground and causing jamming issues. This holds true for vehicle slippage during forward or backward propulsion.

1.2.6 Environment Damage

Legged robots necessitate specific points of contact with the ground, whereas wheeled or tracked vehicles rely on continuous paths along the ground. As a result, legged robots have reduced ground contact compared to conventional vehicles, leading to diminished environmental impact.

1.3 Legged Robots for Planetary Exploration

Legged robots' capability to navigate diverse and unknown terrains, surmount obstacles, and employ discrete ground contact points positions them as ideal candidates for planetary exploration. Specific robots have been purposefully designed and tested for such applications, including: (a) the AMBLER, created by Carnegie-Mellon University with NASA funding, serving as an experimental platform for technology development related to potential Mars missions [10].

1.4 Modular Robots: Versatile Solutions for Robotics

Modular robots represent a fascinating approach to robotics design, offering versatility, adaptability, and scalability in various applications. These robots consist of individual modules or units that can be reconfigured and assembled into different shapes and configurations to suit specific tasks and environments. This modular design al-

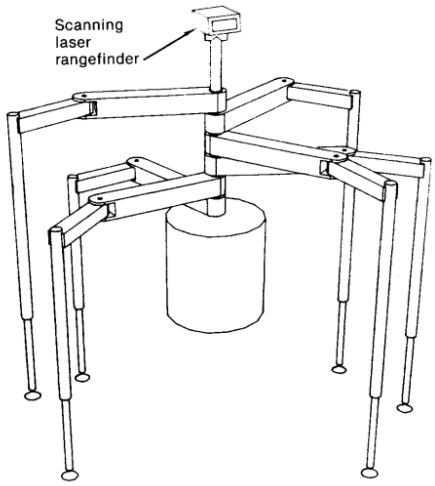


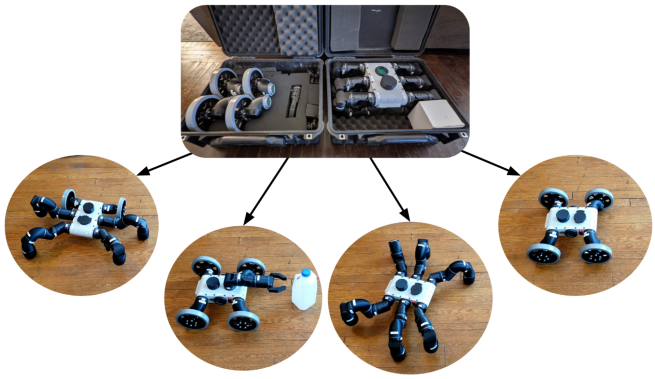
Fig. 1.3: Sketch of the AMBLER robot. [10]

lows for flexibility in robot morphology, enabling robots to adapt to diverse operating conditions and perform a wide range of tasks.

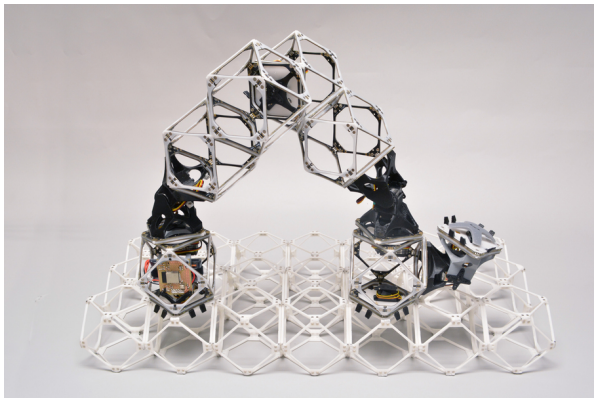
- **Versatility**: Modular robots can be reconfigured into various shapes and configurations, allowing them to adapt to different tasks and environments.
- **Adaptability**: The ability to dynamically adjust robot morphology enables modular robots to traverse diverse terrain, navigate obstacles, and access hard-to-reach areas.
- **Scalability**: Modular design facilitates scalability, allowing robots to be easily expanded or downsized by adding or removing modules as needed.
- **Redundancy and Fault Tolerance**: Modular robots inherently offer redundancy and fault tolerance, as module failure or damage can be mitigated by redistributing or replacing modules.
- **Resilience**: The ability to reconfigure and adapt to changing conditions enhances the resilience of modular robots, making them suitable for operation in remote and hazardous environments.

Moonbot, our modular legged robot designed for space exploration, exemplifies the synergy between modular robotics and lunar exploration. Equipped with a modular architecture, Moonbot is a significant prototype for future lunar exploration mission.

In the event of module failure or damage, the robot can dynamically reconfigure itself by redistributing or replacing modules, ensuring continuity of operation and mission success.



(a) Superbot' s modular units self-assemble. [11]



(b) Assembler bot from MIT. [12]

(c) Snapbot the modular legged robot. [13]

Fig. 1.4: Examples of modular robots.

In summary, modular robots represent a paradigm shift in robotics design, offering unparalleled versatility, adaptability, and resilience for lunar exploration and beyond. With their ability to navigate challenging terrain and withstand harsh environmental conditions, these robots hold immense potential to advance our understanding of the Moon and pave the way for future human exploration and habitation.

1.5 Moonshot Project

The Moonshot project of the Space Robotics Laboratory, led by Prof. Kazuya Yoshida at Tohoku University, aims to accomplish Goal 3: "Realization of AI robots that autonomously learn, adapt to their environment, evolve in intelligence, and act alongside human beings by 2050" [14]. In alignment with this goal, the project envisions the development of a "Self-evolving AI Robot System for Lunar Exploration and Human Outpost Construction" [6]. The project has identified several key objectives to achieve this vision:

1. Design modular and reconfigurable heterogeneous robotic systems.

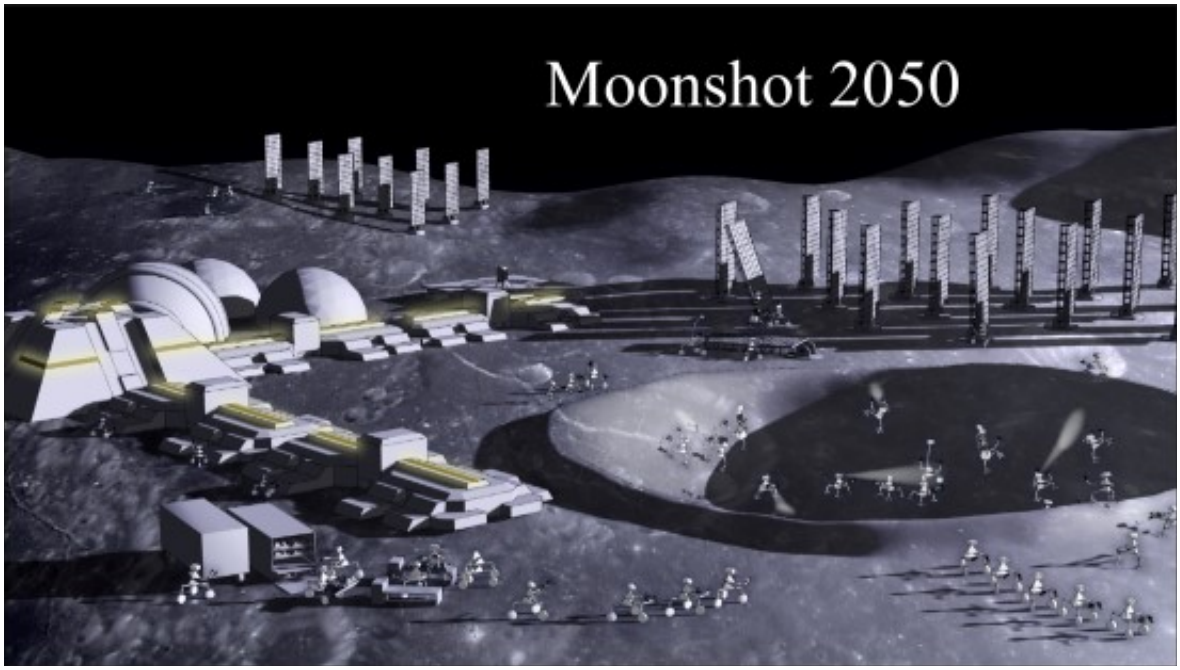


Fig. 1.5: lunar base with AI robots in 2050. [6]

2. Develop a transferable AI system for plug-and-play implementation.
3. Enable on-demand robot design and on-site assembling capabilities.

As part of the Moonshot project, our team has developed "Moonbot", a modular robot designed to address the challenges of lunar exploration. Moonbot inherits the vision of the Moonshot project, integrating autonomous learning, adaptability, and intelligence. This modular robot represents a pioneering step towards the project's overarching goal.

Moonbot draws inspiration from innovative modularity concepts in legged robotics, including the Snapbot project [13] [15], as well as general modular robotics advancements [16]. These projects laid the foundation for Moonbot's approach to modularity.

The advancement of Moonbot's design is the integration of modularity concept with a higher-level legged robot control system [17]. This combination results in a robotic platform capable of self-recognition, adaptability, and dynamic selection of control motions. Moonbot inherits the ability to configure its legs dynamically, ranging from one to four, thus enhancing its versatility and locomotion capabilities.

1.6 Research Objectives

This thesis aims to develop the first prototype of modular control system for legged robots. By using the implemented internal sensors, the robot can recognize the configuration and select the appropriate strategy of motion control, referred as self-recognition ability.

Secondly, the research aims to realize sophisticated software control mechanisms and high-level controllers for the quadruped modular robot. By designing and implementing robust software architectures, the robot can execute complex locomotion patterns. Through these objectives, the research endeavors to enhance the autonomy, adaptability, and overall performance of modular legged robots, paving the way for advancements in robotic exploration and deployment scenarios.

1.7 Thesis Overview

The structure of this thesis is outlined below.

- **Chapter 1: Introduction**:
  This chapter serves as an introduction to the thesis, providing background information and the purpose behind the research.
- **Chapter 2: Modular Legged Robot: "Moonbot"**:
  This chapter offers an in-depth exploration of Moonbot's structural composition, focusing on its hardware overview, modularity, and self-recognition capabilities.
- **Chapter 3: Software and Self-Recognition Algorithms**:
  Here, the thesis delves into the software architecture and logic of Moonbot's self-recognition algorithms, detailing the tools and methodologies used in their development.
- **Chapter 4: Motion Control**:
  This chapter examines Moonbot's motion control mechanisms, including its kinematics and locomotion strategies. It provides insights into how the robot navigates and moves within its environment.
- **Chapter 5: Conclusions**:
  The final chapter concludes the thesis with a summary of the achieved results, reflections on the challenges encountered, and proposals for future advancements in lunar exploration robotics.

Chapter 2

Modular Legged Robot:“Moonbot”

Moonbot is the first version of a modular robot model for the Moonshot project, ”Self Evolving AI Robot System for Lunar Exploration and Human Outpost Construction”. The aimed technologies of this robot are self-reconfigurable and self-assembly abilities.

Using the internal sensors between main body and modular legs, Moonbot autonomously performs a real-time self-recognition system to identify the locomotion style fitting with the configuration.

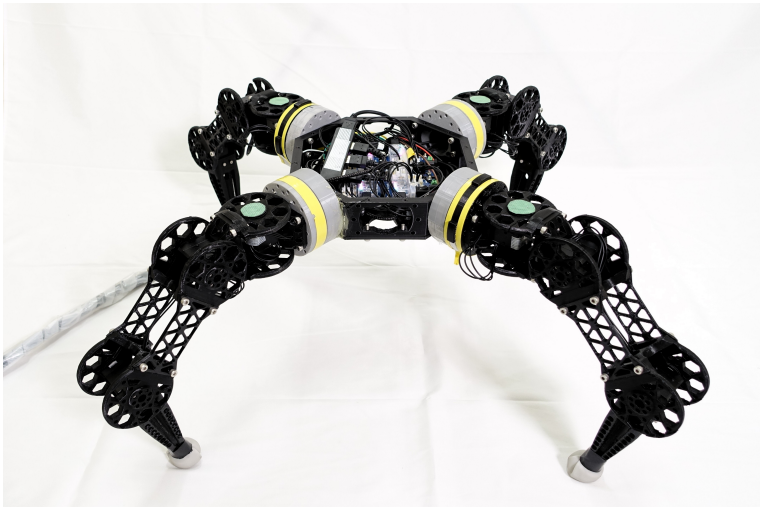


Fig. 2.1: Moonbot with full configuration on the sand box.

2.1 Hardware Design

The Moonbot’s parts are mainly made of PLA (Ploylactic Acid) [18], 3D printed. The Moonbot’s components consist of central body and four leg modules. The legs are connected via magnetic connection. The communications of each leg and the computer transfer via pogo pin connector. In the this section, the mechanical design and electronic of Moonbot are explained

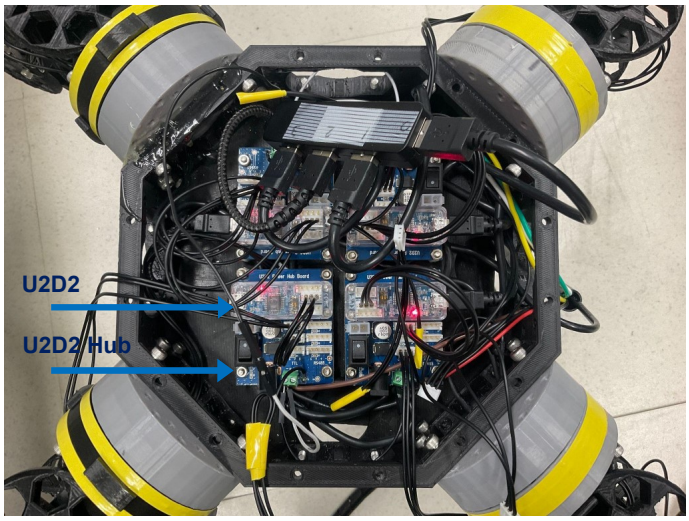


Fig. 2.2: Electronics components on the Moonbot center hub.

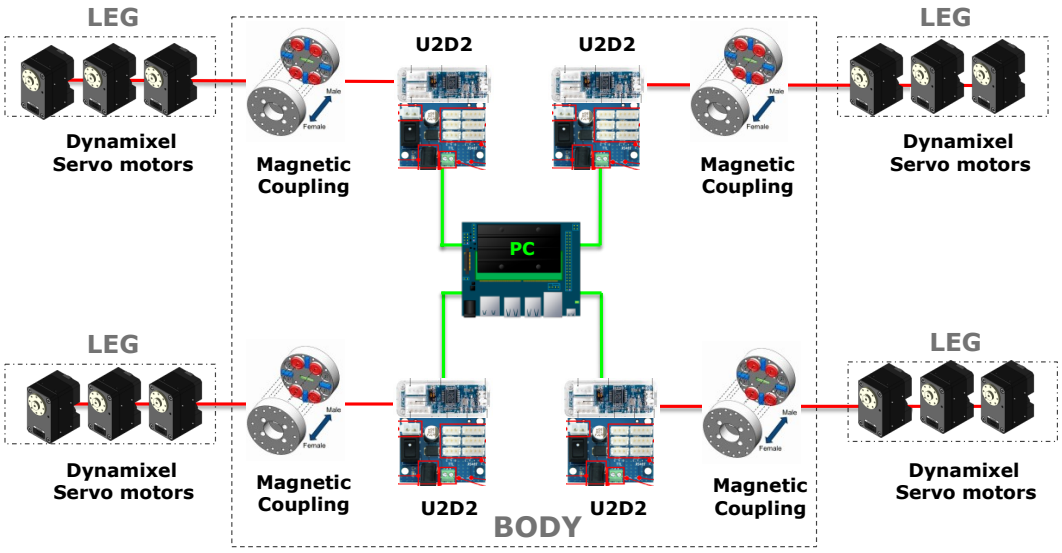


Fig. 2.3: Electronics components on the Moonbot center hub.

2.1.1 Central Body

The Moonbot’s central body is 20x20 cm square shape. The total weight, including all electronic components, is 1.02 kilograms. On the body, there are four U2D2 and U2D2 hub, a USB communication converter for connecting with servo motors. This converter is available for various communication protocol, including 4-pin UART, 3-pin TTL, and 4-pin RS-485. In this time, 3-pin TTL is selected. At the bottom of the body, four ball casters are attached to support the locomotion style where the body of the robot lies on the floor. At each corner of the shape, four ports of magnetic connection are anchored.

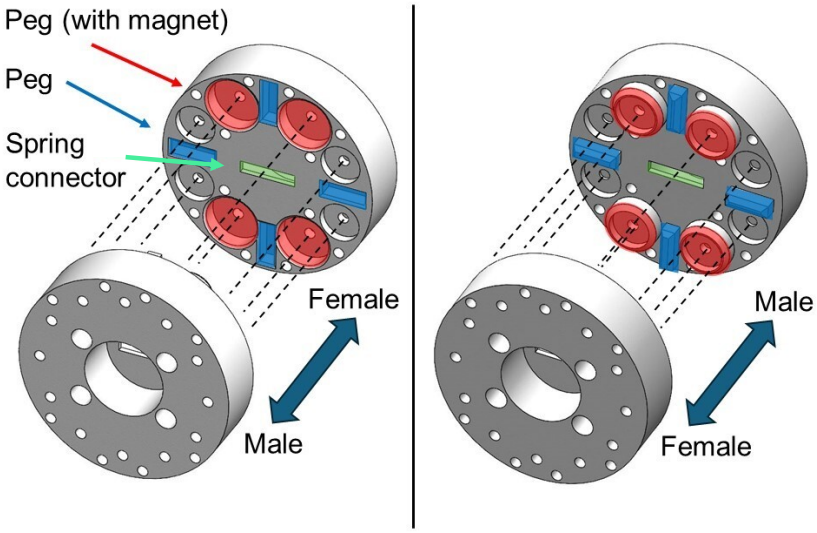


Fig. 2.4: Magnetic connector.

2.1.2 Magnetic Coupling

Each of the legs is connected to the body via 8 pairs of circular Neodynium magnets, arranged in circular pattern as shown in as shown in Fig. 2.4. A piese of magnet has 180 grams of weight, 3 mm in height and 150 mm of diameter.

In addition to the magnetic coupling, the electronics connections between the body and the legs are facilitated by pogo pins. Pogo pins, also known as spring-loaded pins, provide a reliable electrical connection between two components while allowing for movement and flexibility. This spring loaded connector is attached at the center of the magnetics connector of each module as shown in Fig. 2.5

2.1.3 Modular Leg Modules

Moonbot consists of four modular legs, utilizing a yaw-pitch-pitch configuration. Each of the leg has 530 grams with a small cylindrical end effector.The lengths of each link of the leg are shown in Table 2.1

Table 2.1: Moonbot's leg parameters.

Link's name	Length [mm]
Coxa	64.55
Femur	129.0
Tibia	156.0

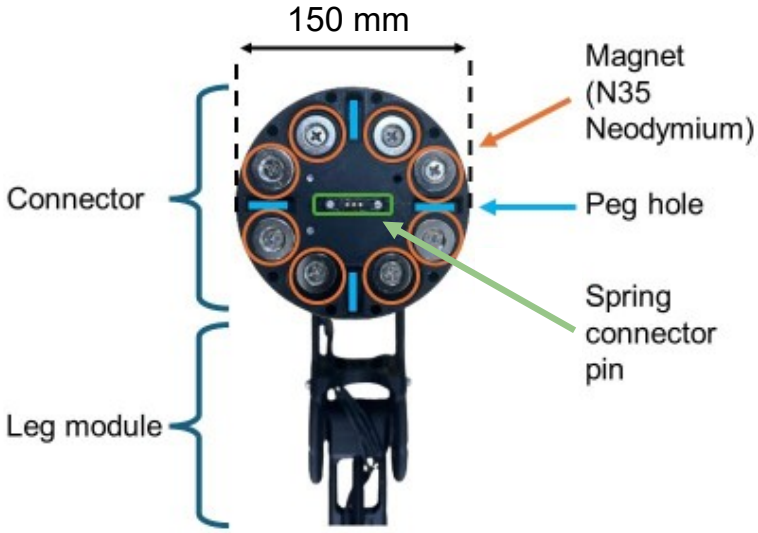


Fig. 2.5: Pogo pin connector located at the middle of magnetic coupling.

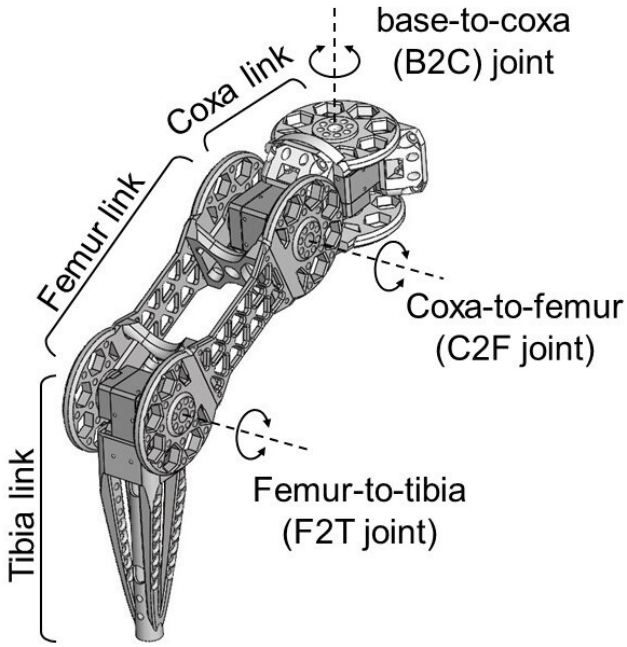


Fig. 2.6: Moonbot's modular legs.

2.1.4 Joints' Actuator: Dynamixel

Dynamixel is a brand of high-performance smart servos developed by ROBOTIS [19]. These servos are widely used in robotics applications due to their precision, durability, and advanced features. Dynamixel servos are known for their ability to provide

Fig. 2.7: Dynamixel servo motor, model XM430-W350-T.

Table 2.2: Dynamixel XM430-W350-T Specifications.

Specifications	
Parameter	**Value**
Speed	46 RPM
Torque	4.1 N·m
Size	33.5×46.5×34.0 mm
Mass	82 g

position, velocity, and torque control, as well as various feedback mechanisms.

The joints of the Moonbot is moved by Dynamixel XM430-350-T, utilizing a 3-pin TTL communication protocol, Moonbot can generate trajectories with precise position control. The specification of this servo is shown below

Chapter 3

Software and Self-Recognition Algorithms

Moonbot, designed for lunar surface operations, relies on a sophisticated software foundation. This chapter describe the structure of Moonbot's software architecture and the used tools.

3.1 Software Tools

3.1.1 ROS and ROS2

ROS (Robot Operating System): [20] is a flexible framework for writing robot software. ROS provides services for hardware abstraction, device drivers, communication between processes, package management, and more. ROS has been widely adopted in the robotics community, fostering collaboration and the sharing of libraries and tools.

ROS2 (Robot Operating System 2): The next generation of ROS, ROS2 [21] is designed to address limitations and capitalize on the lessons learned from the widespread use of its predecessor. One of the significant improvements in ROS2 is its enhanced support for the Data Distribution Service (DDS) middleware [22]. DDS is a standardized communication middleware that facilitates efficient and reliable data exchange between distributed systems.

By the above information, Moonbot utilizes ROS2, Foxy distribution, as the primary tool for organizing communication among its modular components. The decision to use ROS2 is motivated by its enhanced features, including:

- **Real-Time Capabilities:** ROS2 provides improved support for real-time systems, crucial for dynamic and responsive control.
- **Communication Middleware:** ROS2 offers a communication middleware options, allowing components to exchange information in a distributed system.

- **Modularity and Extensibility:** The modular and extendable structure allow developer to debug or extend the feature of the program easily.

3.1.2 ROS2 Core Concepts and Elements

ROS2 Graph

In ROS. it provide a communication network called "ROS graph". ROS2 process all data and connect them simultaneously via ROS2 graph. Develop utilize this concept to map the ROS2 elements to execute the program.

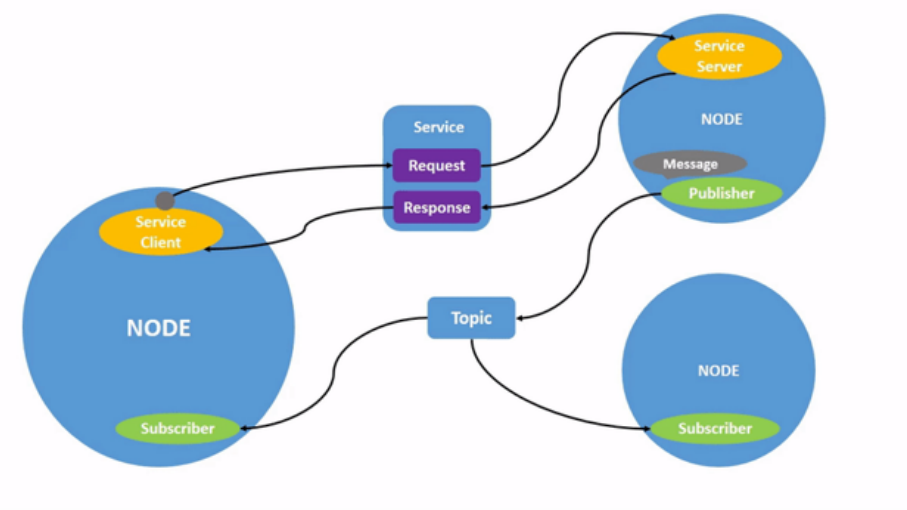


Fig. 3.1: Visualization of ROS2 graph. [21]

ROS2 Nodes

Nodes are individual processes in a ROS 2 system. They communicate with each other by publishing and subscribing to topics, providing and using services, and executing actions.

ROS2 Topics

Topics are named buses over which nodes can exchange messages. A node can publish messages to a topic, and any other node can subscribe to that topic to receive those messages. Topics facilitate asynchronous communication between nodes.

ROS2 Services

Services allow request-response communication between nodes. A node provides a service by specifying a request message type and a response message type. Other nodes can then call that service by sending a request message and receiving a response message.

ROS2 Actions

Actions provide a way for nodes to execute long-running tasks in a goal-oriented manner. Actions consist of a goal, feedback, and result, allowing for more complex interaction patterns than services. Nodes can send action goals, receive feedback on the progress of those goals, and eventually receive a result once the goal is completed.

3.1.3 ROS2 Control

Moonbot's control architecture is developed with ROS2, and within this framework, the ROS2 control package [23] plays a key role. In this section, we are going to delve into the specifics of ROS2 control, its integration with Moonbot's control system, and the key components that contribute to Moonbot's adaptability and precision.

ROS2 Control Overview

The `ros2_control` is a package providing a modular and configurable framework for controlling robotic systems. This package is a redevelopment of ros_control [24], used in ROS.

The ros_control framework relies on the `RobotHW` class as a structure to manage hardware interactions. While it provides a rigid structure for handling various hardware components like sensors and actuators, extending existing robots with additional hardware requires coding, limiting the flexibility to integrate new elements. The use of the `CombinedRobotHardware` class [25] attempts to address this limitation, yet it may not be optimal, especially when incorporating external sensors into the system.

In contrast, the ROS2 control framework introduces a more versatile approach by defining three types of hardware: `Actuator`, `Sensor`, and `System`. Through the composition of these basic components, any robotic cell, encompassing the robot and its surrounding environment, can be described. Unlike ros_control, ros2_control does not enforce a fixed set of interface types, allowing flexibility in defining interfaces. This

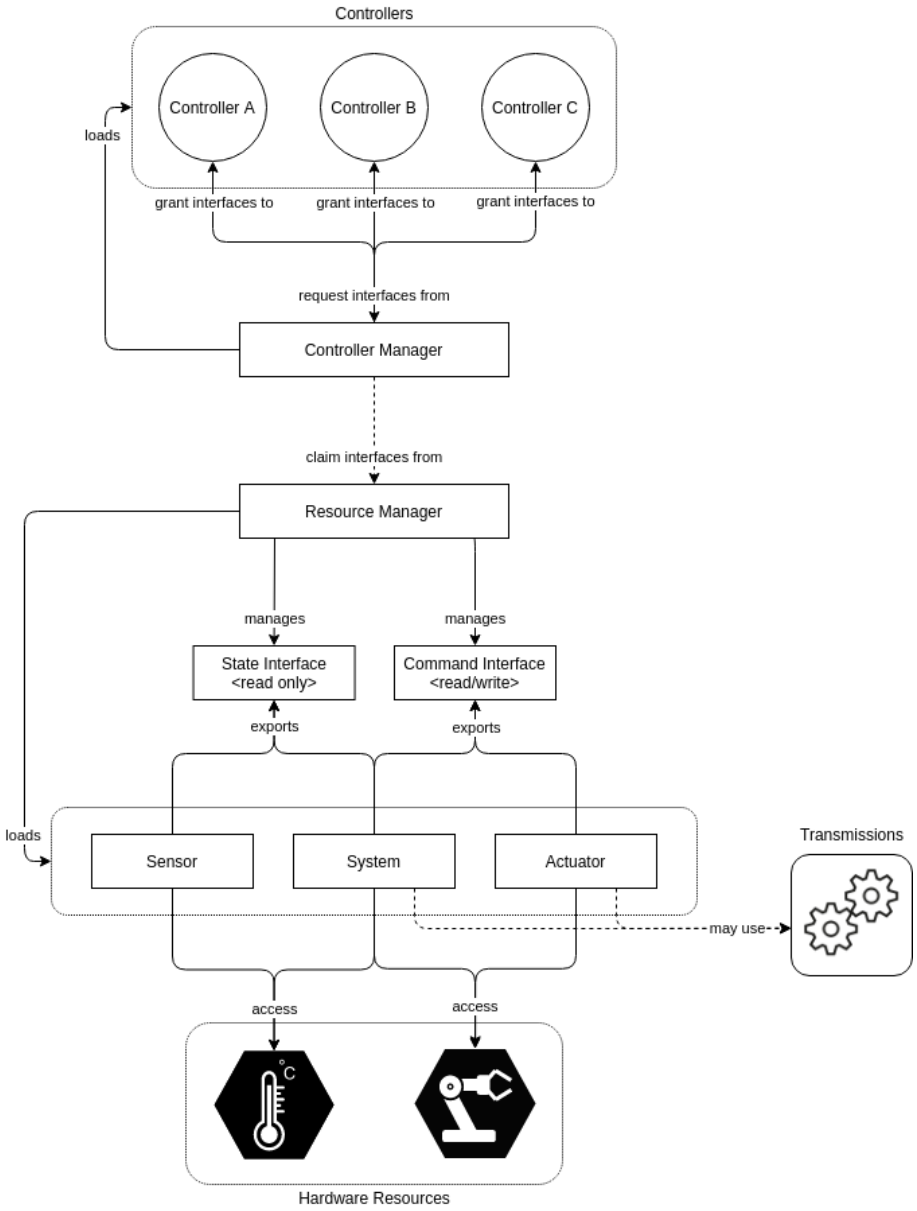


Fig. 3.2: ROS2 Control Framework. [23]

ensures compatibility with standard controllers, as standard interfaces are defined as constants in the `hardware_interface` package. Moreover, the `ControllerManager` in ros2_control facilitates controlled access to the hardware by managing the state of available interfaces through the `ResourceManager`. Unlike ros_control, controllers have no direct access to hardware, and they register their interfaces with the `ControllerManager`, enhancing modularity and resource conflict resolution.

Controller Manager

At the heart of the ROS2 control architecture lies the Controller Manager. This component is responsible for loading, managing, and switching between different controllers. In Moonbot, the Controller Manager coordinates the operation of controllers associated with each leg, allowing dynamic transitions between various locomotion styles based on the robot's configuration.

ROS2 Life Cycle

ROS2 Lifecycle [26] is utilized in controller manager of ROS2 control. The ROS 2 Node Lifecycle provides enhanced control over the state of the ROS system, as shown in Fig. 3.3, ensuring the proper initualization of all components before executing any behavior. This mechanism enables nodes to be restarted or replaced while the system is running.

A key aspect emphasized in this document is the concept of a managed node, which adheres to a predefined interface and operates within a well-defined life cycle state machine. By treating managed nodes as black boxes, developers have flexibility in implementing the life cycle functionality while ensuring compatibility with management tools across all compliant nodes.

3.1.4 Dynamixel SDK & Workbench

Dynamixel SDK

The Dynamixel SDK (Software Development Kit) [27] is a set of libraries and tools provided by ROBOTIS to interface with Dynamixel servos. It includes APIs (Application Programming Interfaces) for various programming languages such as C, C++, Python, and MATLAB, allowing developers to control Dynamixel servos easily from their preferred programming environment.

Key features of the Dynamixel SDK include:

- **High-level APIs:** The SDK provides high-level APIs that abstract the low-level communication protocols of Dynamixel servos, making it easier for developers to control and interact with the servos.
- **Advanced control functionalities:** It offers advanced control functionalities such as position control, velocity control, torque control, and compliance control.

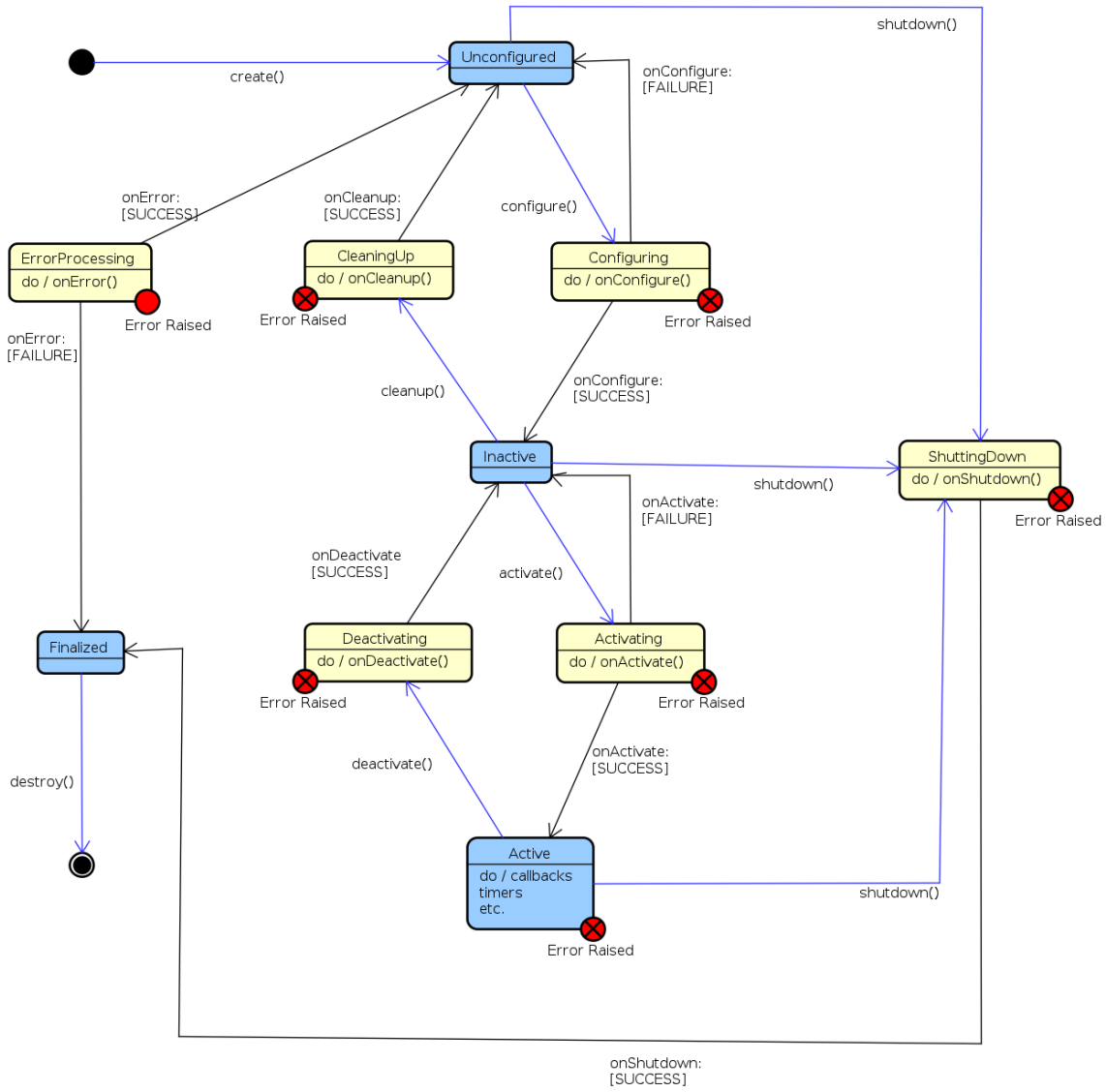


Fig. 3.3: ROS2 Node Lifecycle. [26]

- **Diagnostic tools:** The SDK includes diagnostic tools that allow developers to monitor the status of Dynamixel servos, diagnose issues, and perform troubleshooting.

- **Community support:** ROBOTIS provides extensive documentation, tutorials, and forums to support developers using the Dynamixel SDK, fostering a vibrant community of users who share knowledge and resources.

Dynamixel Workbench

In addition to the Dynamixel SDK, ROBOTIS offers the Dynamixel Workbench [28], a higher-level software framework built on top of the SDK. Dynamixel Workbench provides a comprehensive set of tools and utilities for managing Dynamixel-based robotic systems. It includes features such as:

- **Motion editing:** Dynamixel Workbench allows users to create and edit motion sequences for controlling Dynamixel servos in complex robotic behaviors.
- **Parameter tuning:** Users can adjust various parameters of Dynamixel servos, such as PID gains, compliance margins, and operational limits, to optimize performance and behavior.
- **Hardware interface:** Dynamixel Workbench includes APIs and drivers for interfacing with Dynamixel hardware, allowing users to communicate with servos, sensors, and other peripherals especially with ROS2 communication.

Together, the Dynamixel SDK and Dynamixel Workbench provide a powerful ecosystem for developing and controlling Dynamixel-based robotic systems, offering flexibility, scalability, and ease of use for researchers, educators, and hobbyists alike.

The Pinging function [29], within Dynamixel SDK, serves as a fundamental component crucial for self-recognition algorithms in Moonbot. This function plays a pivotal role in enabling the system to identify and communicate with connected specific ID Dynamixel servos. By sending a ping command to each servo, the system can receive a response indicating the presence and status of the servos. This information is essential for initializing, configuring, and monitoring the servos within the robotic framework.

3.2 Self-Recognition

In the context of our robotic system, self-recognition refers to the ability of the robot to autonomously assess the status and connectivity of its modules

Moonbot's adaptability is a cornerstone of its design, allowing for dynamic reconfiguration through the attachment and detachment of modules via magnetic connectors. To effectively control the robot, it is imperative for Moonbot to possess self-awareness regarding its current configuration and the type of connected module (e.g., yaw-pitch-pitch leg, gripper).

3.2.1 Problem Statements

1. The robot need to be able to activate and deactivate the controller for any kinds of the module.
2. With the different number of connected modules, the robot moves with different motion style.

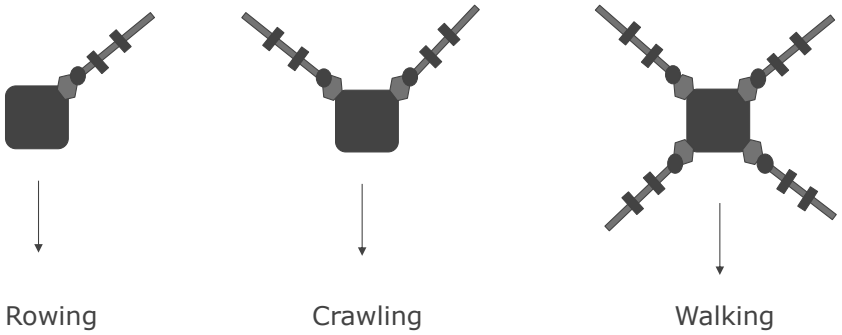


Fig. 3.4: Motion selection strategies for different number of module connections.

3. With the different kinds of module, the robot selects the proper motion for the specific module type.

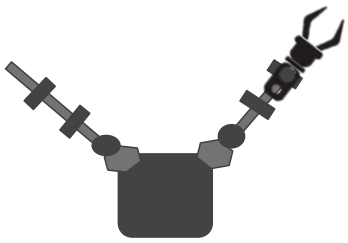


Fig. 3.5: Different kind of modules connected to the robot.

3.2.2 Algorithms Overview

Inspired by the design of Snapbot, a modular-legged robot developed by Disney Inc. [13], [15], Moonbot employs a self-recognition system. The connections are periodically verified by pinging the specific ID of Dynamixel motors. The flowchart in Fig. 3.6 and Algorithm 1 shows the modular connection algorithm for one connector. After the robot identifies the type of module, the module's controller at the specific port is independently launched, and activated. With the specific configuration identified, Moonbot then selects the appropriate locomotion style.

3.2.3 Node Structure and Components

The controller of the Moonbot are separated for each connector, including limbs— RF (Right Front), LF (Left Front), LR (Left Rear), and RR (Right Rear) as shown in Fig. 3.7. From this structure, to control the motion, four launching services are needed

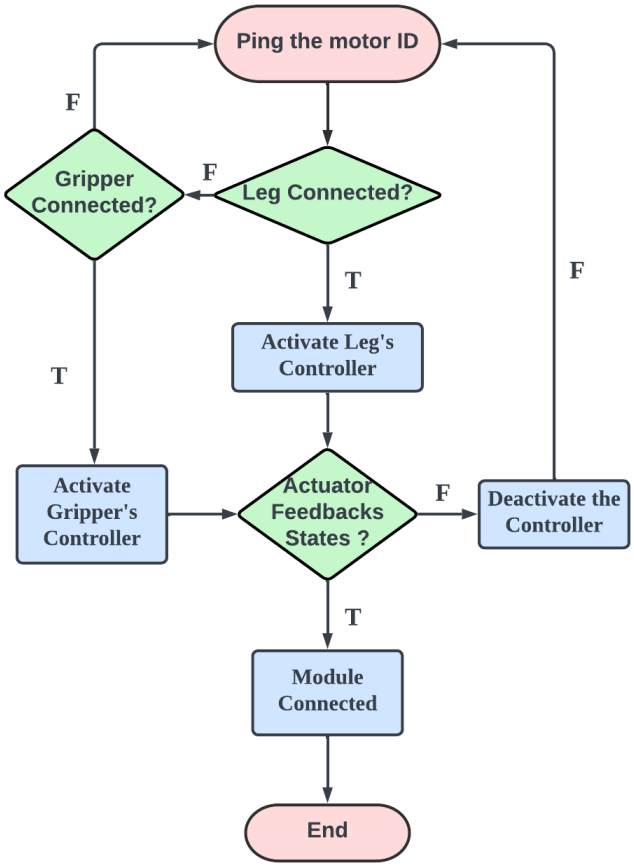


Fig. 3.6: Self-recognition algorithm for each connection.

to be implemented for each controller. With the framework of ROS2 control, we can handled the command and state of each joint independently. Consequently, the module detection node handle the joint states of four specific ports, separately.

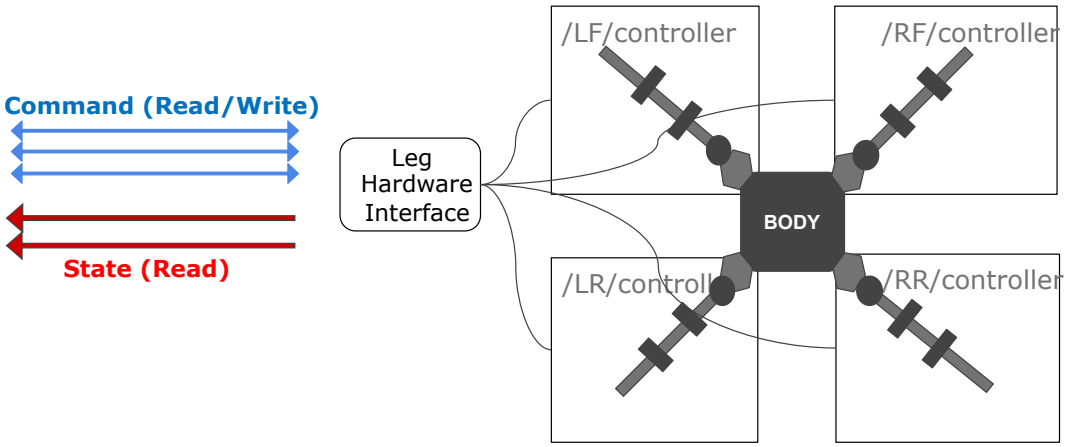


Fig. 3.7: Separated controller for each module.

Algorithm 1 Self-Recognition for module connection at right-rear port.

```
1: function RR_INITIALIZER_CALLBACK
2:     if no RR_module_detected then
3:         if Found leg module at RR port then
4:             Activate leg's controller at RR port
5:         else if Found gripper module at RR port then
6:             Activate gripper's controller at RR port
7:         end if
8:     else if RR_module_detected then
9:         if RR_leg_connected then
10:            if RR_state error then
11:                Deactivate leg's controller at RR
12:            end if
13:        else if RR_gripper_connected then
14:            if RR_state error then
15:                Deactivate gripper's controller at RR
16:            end if
17:        end if
18:    end if
19: end function
```

3.2.4 Initialization Process

The initialization process is involved with activation of the controller into the life cycle of control, as we have discussed in the section of controller manager.

Upon initiation, the nodes for module detection check the availability of the corresponding hardware by attempting to communicate with the actuators (pinging). The communication is established through Dynamixel motors with the specific ID number. After the success of this communication, the module detection node sends the request to the service server that handle the launch service of actuator activation.

3.2.5 Periodic Connection Status Monitoring

Following the initialization, the node continuously monitors the joint states of each module at regular intervals, providing real-time information on the current position of the limbs' joints. Timer callbacks dedicated to each limb facilitate this periodic monitoring.

Additionally, the node functions as an error detector by monitoring the status of the motors. If a joint position deviates beyond a predefined threshold, indicating potential issues like disconnection or misalignment, the node dynamically updates the connection status for that module.

However, it's worth noting that the current version of the *hardware_interface* in ROS2 control does not include a restart function for lost hardware. Therefore, the launch services incorporate a feature to automatically shut down the controller manager process in response to hardware disconnection. Upon reconnecting the hardware, the program reinitializes the ROS2 control activity, relaunching the controller manager process.

3.2.6 Adaptability to Grippers

The Dynamixel motors offer a unique ID identification feature, accessible through the Dynamixel Wizard GUI tool [30], which facilitates initial setup. In Moonbot's control system, the module detection node is designed to identify specialized components, such as grippers, by searching for specific motor IDs. For instance, a motor ID of 1 corresponds to the leg module, while an ID of 2 signifies the gripper module. By extending self-recognition capabilities to include end-effectors, we enhance the versatility of our robotic system.

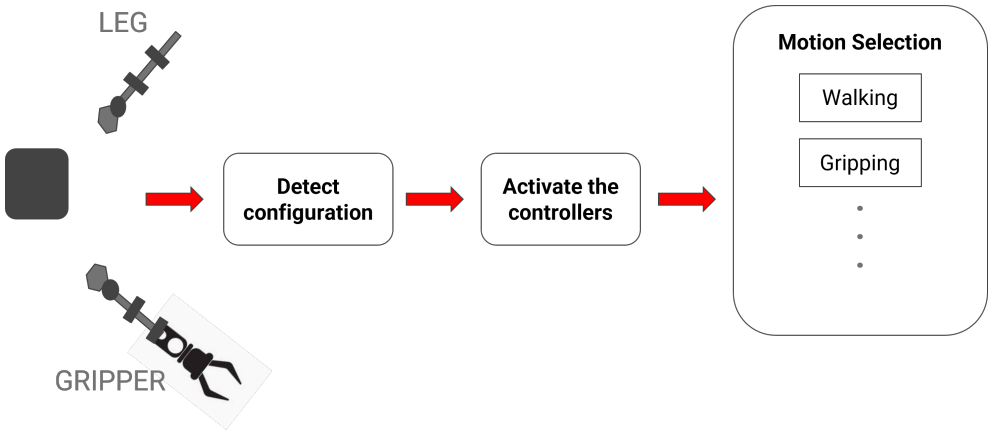


Fig. 3.8: Motion selection depending on module type.

3.2.7 Motion Selection

After the implementation of connection recognition of the module is success, the self-recognition algorithm can report the connection status of each connector in real-time, periodically. From this information, we can perform an experiment for motion selection depending on Moonbot configuration's which are mentioned in the problem statements.

To present the adaptability of the modular legged robot, Moonbot's locomotion styles including crawling motion and walking motion. The motion style is selected by the number of leg module connection.

1. *Crawling Motion:* Moonbot moves by crawling motion when operating with a single-leg configuration towards the direction of connected module. Extending its capabilities, two-legged and three-legged configuration, Moonbot move with the same style as single configuration, but higher power by more legs applied.

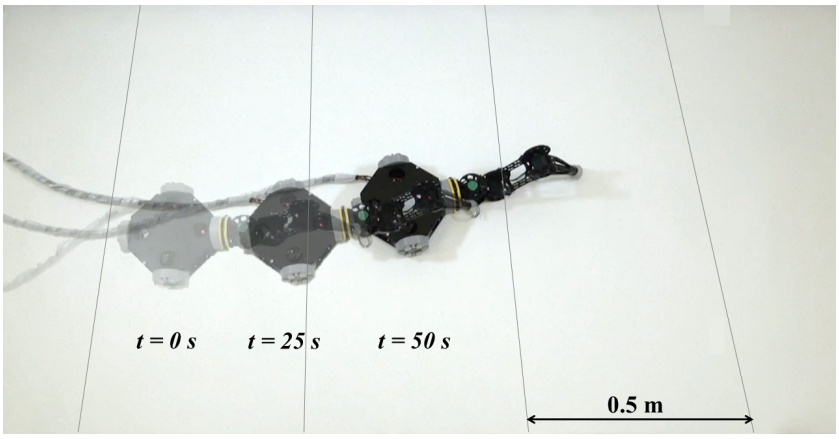
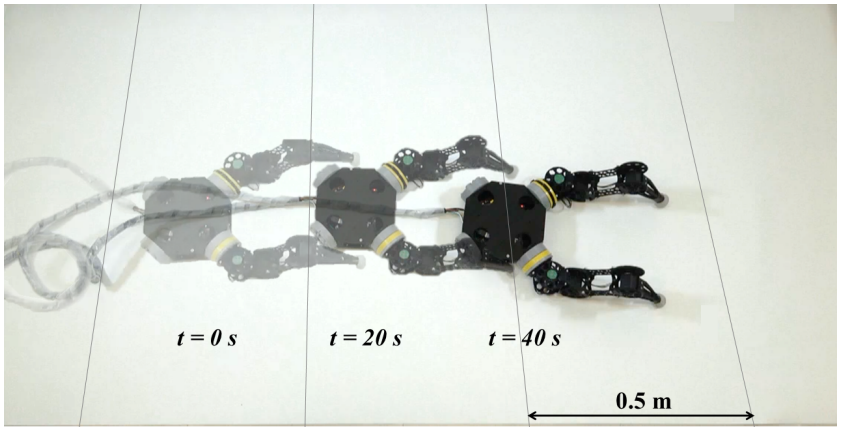
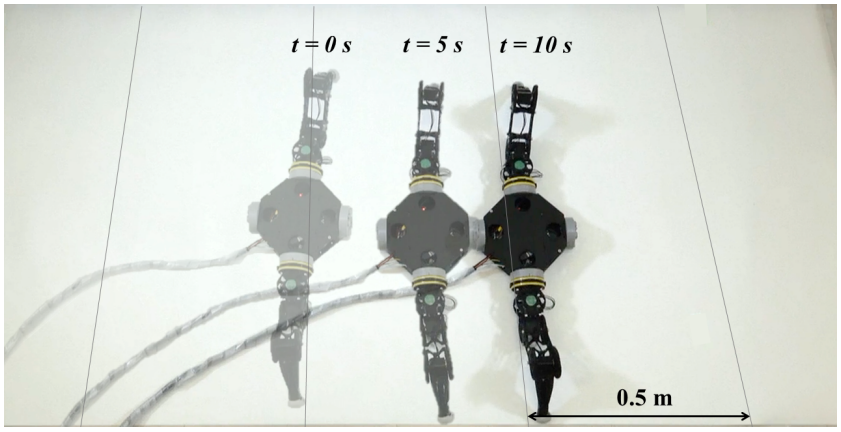


Fig. 3.9: Snapshots of crawling motion for single-leg configuration.



(a) Crawling motion for double-leg configuration.



(b) Crawling motion for double-leg at opposite ports configuration.

Fig. 3.10: Snapshots of crawling motion for double-leg configuration.

2. *Walking motion:* In full configuration, employing all four legs, is designed for versatile locomotion. Moonbot can perform both a crawl gait for static stability and a trot gait for a more dynamic walking motion.

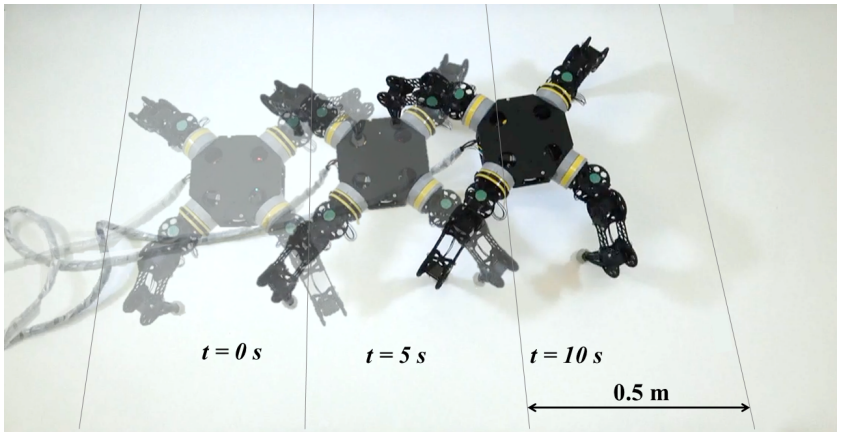


Fig. 3.11: Snapshots of walking motion in self-recognition test.

In addition to the number of module, the type of module also a parameter for motion selection. Moonbot's module contain both leg type and gripper type. By finding the ID dynamixel motor labeled as a gripper type, the robot can recognize the connection of gripper and perform a motion task such as waving and gripping the gripper.

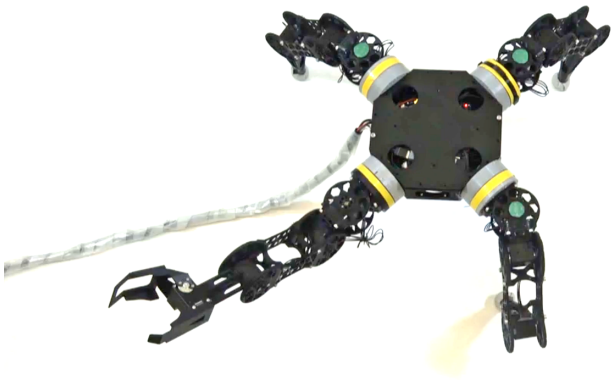


Fig. 3.12: Gripper module connect with the Moonbot.

Fig. 3.13: Motion selection depending on module type.

Chapter 4

Motion Control

This section drives into the locomotion strategy for Moonbot with full configuration operation. The motion control is orchestrated through the integration of ROS2 control framework. The joint trajectory controller is used to operate the multiple joints.

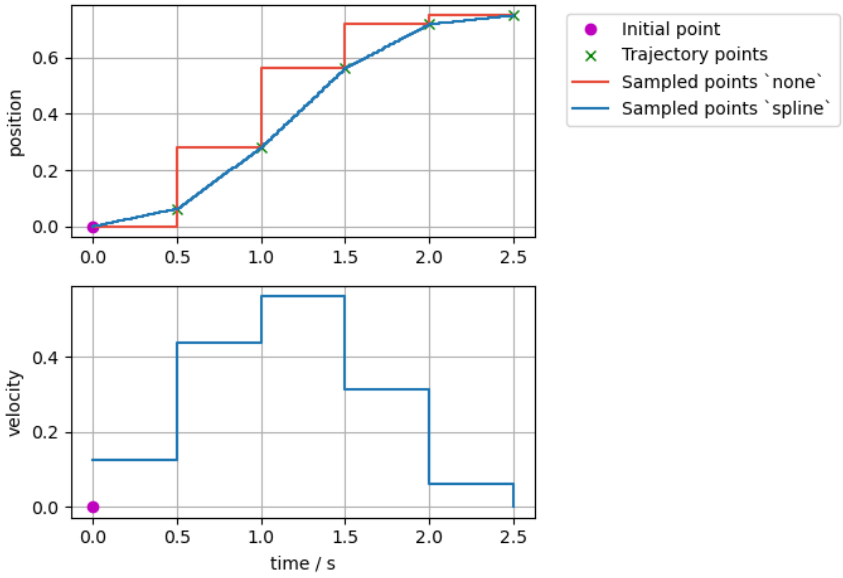
4.1 Joint Trajectory Controller

ROS2 control offers a versatile tool called `ros2_controllers` [31], designed as a controller interface accommodating various robot specifications. For example, the `position_controller` can accept position inputs and generate corresponding outputs to a position interface. Similarly, the `velocity_controller` receives input and produces velocity output. Additionally, specialized controllers like the `diff_drive_controller` provide advanced control interfaces tailored for mobile robots equipped with wheeled locomotion systems.

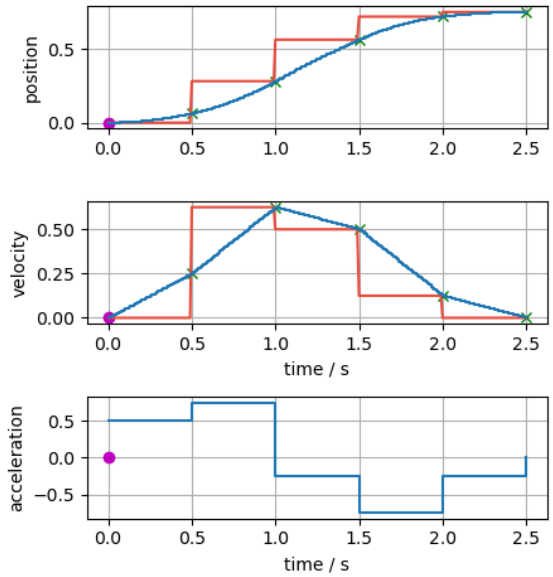
For precise control of Moonbot's dynamic locomotion, the `joint_trajectory_controller` from ROS2 is utilized. This controller executes joint-space trajectories on multiple joints, enabling smooth and dynamic locomotion by interpolating the waypoints to desired target at specific time.

There are three strategies for spline interpolation, depending on specification of those waypoints.

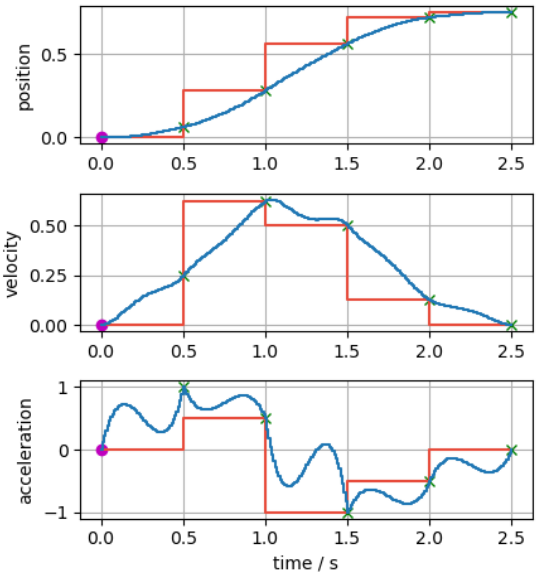
- Linear: Only position is specified so it provide the continuity of joint trajectory only at the position level.
- Cubic: Position and velocity are specified so the continuity is confirmed at the velocity level.
- Quintic: Position, velocity and acceleration are specified: Guarantees continuity at the acceleration level.



(a) Joint-space trajectory for linear spline.



(b) Joint-space trajectory for linear spline.



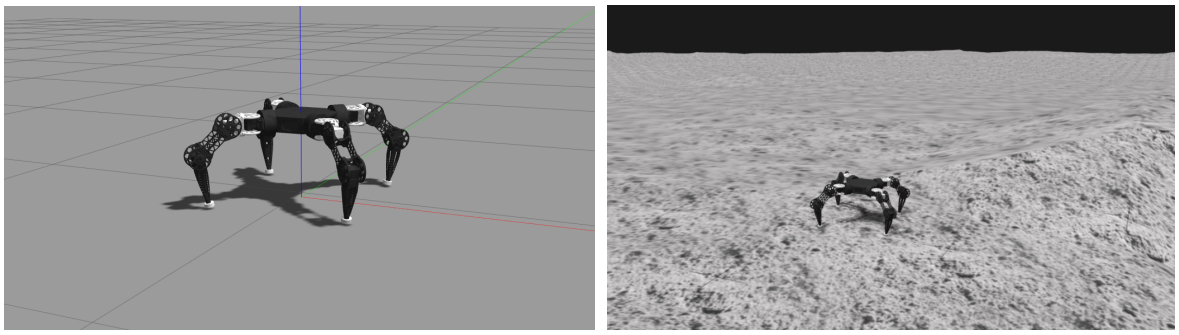
(c) Joint-space trajectory for linear spline.

Fig. 4.1: Spline interpolation Strategies of joint trajectory controller.

The joint-space trajectories for each case are illustrated in Fig. 4.1. In this time, Moonbot is using only position linear interpolation for easier prototyping the motion control and designing gait motion.

4.2 Gazebo Simulation

Gazebo [32] stands as a robust open-source 3D simulation environment widely used in the robotics community. Its physics engine, sensor simulation, and realistic rendering



(a) Earth ground environment. (b) Lunar surface environment.

Fig. 4.2: Moonbot in Gazebo simulation.

capabilities make it a choice for testing Moonbot's motion and control strategies in both earth-based environment and lunar landscape environment.

4.3 Kinematics Model of Legs Motion

The inverse kinematics (IK) model is used as a function for calculate the joint positions, based on the given coordinates of the end effector's position in the leg's coordinate system.

Let (x, y, z) represent the desired coordinates of the end effector position in the leg's coordinate system, as shown in Fig. 4.3. The IK algorithm calculates the joint angles $(\theta_1, \theta_2, \theta_3)$ corresponding to the coxa, femur, and tibia joints, respectively.

The result of coxa joint (θ_1), responsible for lateral movement, is limited within a span of -90 to 90 degrees. Similarly, the femur joint (θ_2), facilitating forward and backward motion, adheres to the same range of -90 to 90 degrees. Finally the tibia joint (θ_3) spans from -100 to 100 degrees. These specified limits ensure that the leg articulates within safe and mechanically feasible parameters.

First, we initialize the parameters including the lengths of the leg segments (l_1, l_2, l_3) for coxa, femur, and tibia, respectively.

Inverse Kinematics Calculation:

- First, we calculate the distance from the coxa joint to the EE position in xy plane, l_{xy}, and the distance from femur's joint to the end effector, D_{j2E}:

$$D_{xy} = \sqrt{x^2 + y^2} \tag{4.1}$$

$$D_{j2E} = \sqrt{z^2 + (D_{xy} - l_1)^2} \tag{4.2}$$

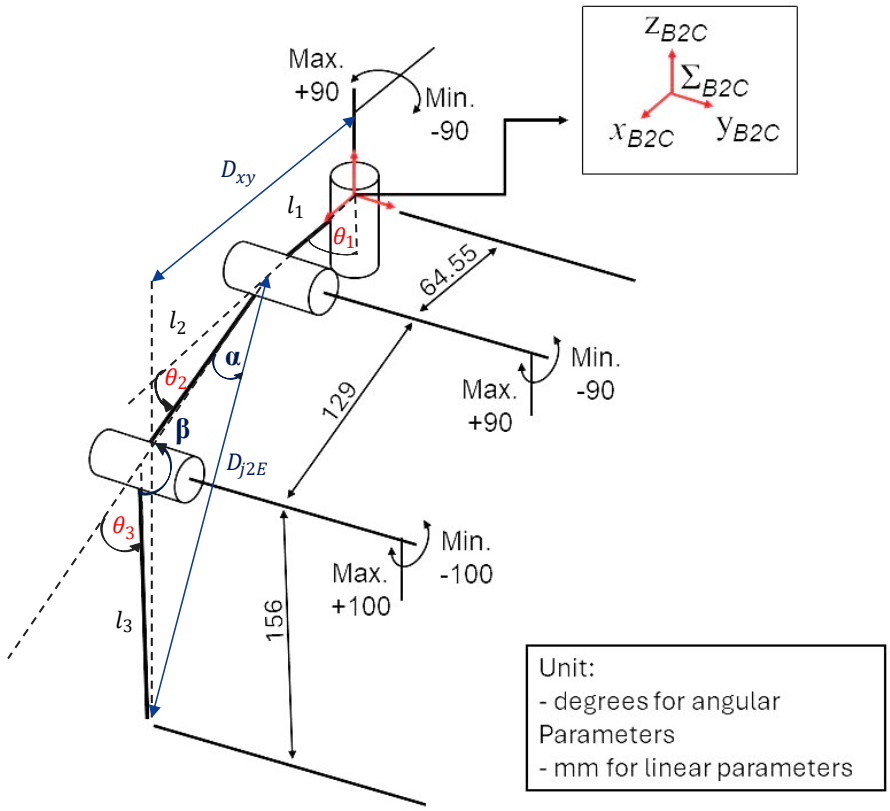


Fig. 4.3: Configuration of leg's module of Moonbot.

- Calculate the angle (α) between the femur segment and the line connecting the coxa and EE position:

$$\alpha = \arccos\left(\frac{l_3^2 - l_2^2 - D_{j2E}^2}{-2 \cdot l_2 \cdot D_{j2E}}\right) \tag{4.3}$$

- Calculate the joint angle (θ_1) of the coxa joint:

$$\theta_1 = \arctan(y, x) \tag{4.4}$$

- Calculate the joint angle (θ_2) of the femur joint:

$$\theta_2 = \arctan\left(\frac{z}{D_{xy} - l_1}\right) - \alpha \tag{4.5}$$

- Calculate the angle (β) between the tibia segment and the femur segment:

$$\beta = \arccos\left(\frac{D_{j2E} - l_2^2 - l_3^2}{-2 \cdot l_2 \cdot l_3}\right) \tag{4.6}$$

- Calculate the joint angle (θ_3) of the tibia joint:

$$\theta_3 = \pi - \beta \tag{4.7}$$

Return the calculated joint angles $(\theta_1, \theta_2, \theta_3)$ for the coxa, femur, and tibia joints, respectively.

With the given leg segment lengths:

$$l_1 = 64.55 \text{ mm} \tag{4.8}$$

$$l_2 = 129 \text{ mm} \tag{4.9}$$

$$l_3 = 156 \text{ mm} \tag{4.10}$$

the IK algorithm calculates the joint angles required to position the leg's end effector at the desired coordinates (x, y, z) of the leg's frame.

Moonbot is also implemented with motion control in both translation and rotation around the axis. The rotation matrix $\mathbf{R}(\theta_\mathbf{x}, \theta_\mathbf{y}, \theta_\mathbf{z})$ is used to transform coordinates of the body's frame through matrix multiplication.

The elements of the rotation matrix are computed using the following expressions:

$$\mathbf{R}(\theta_x, \theta_y, \theta_z) = \begin{bmatrix} c_y c_z & s_x s_y c_z - c_x s_z & c_x s_y c_z + s_x s_z \\ c_y s_z & s_x s_y s_z + c_x c_z & c_x s_y s_z - s_x c_z \\ -s_y & s_x c_y & c_x c_y \end{bmatrix} \tag{4.11}$$

where c_i represents cosine of θ_i and s_i represents sine of θ_i.

4.4 Quadruped Operation

The process of gait generation in legged locomotion is fundamental for achieving stable and efficient movement patterns. It involves determining the stride length and trajectory for each leg's step to maintain balance and driving force. Let's delve into the mathematics behind generating a gait for legged robots.

The control structure for the full configuration of Moonbot is depicted in Figure Fig. 4.4. Drawing inspiration from control strategies utilized in various quadruped and legged robots [17], [33], [34], Moonbot adheres to the classic approach for controlling quadruped robots.

At the low-level, an actuator interface serves as a bridge between user commands and the actual hardware. This interface, designed with the life cycle concept in ROS2,

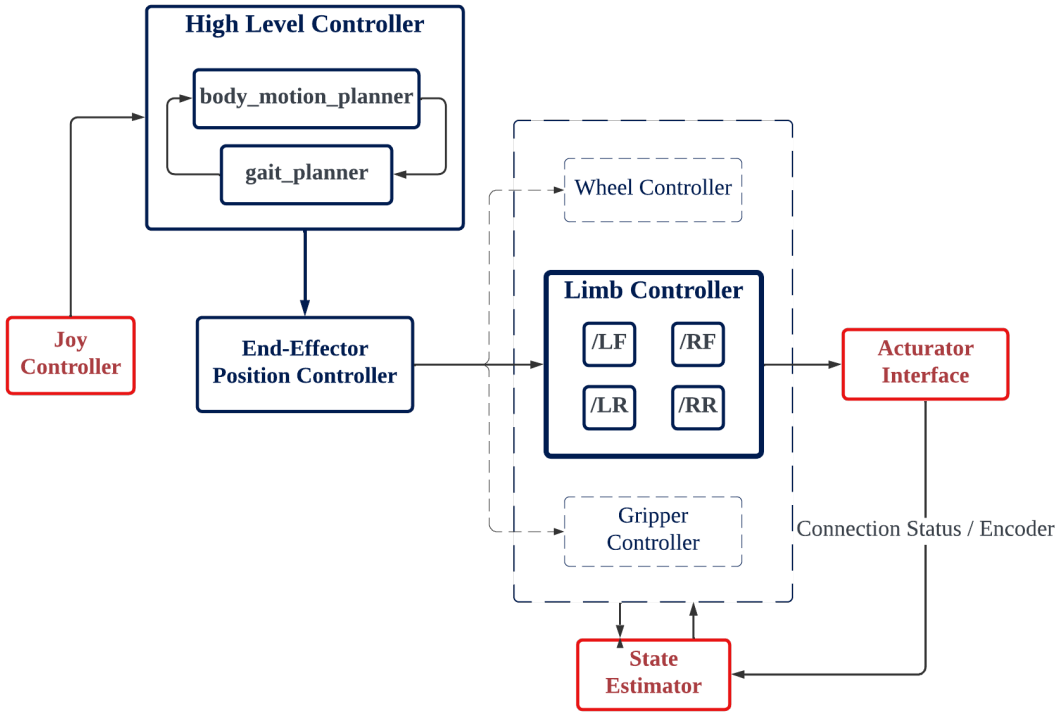


Fig. 4.4: Control architecture for full configuration Moonbot.

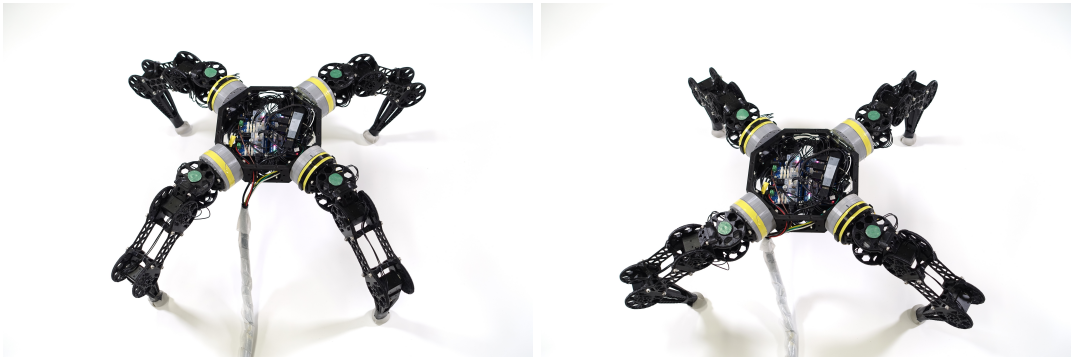
facilitates the transmission of commands to the motors while retrieving important feedback, including joint position, velocity, and torque.

End-effector control primarily focuses on solving the inverse kinematics problem to determine the joint positions corresponding to desired end-effector positions. The computed solutions are then forwarded to the appropriate controller based on the type of module in use.

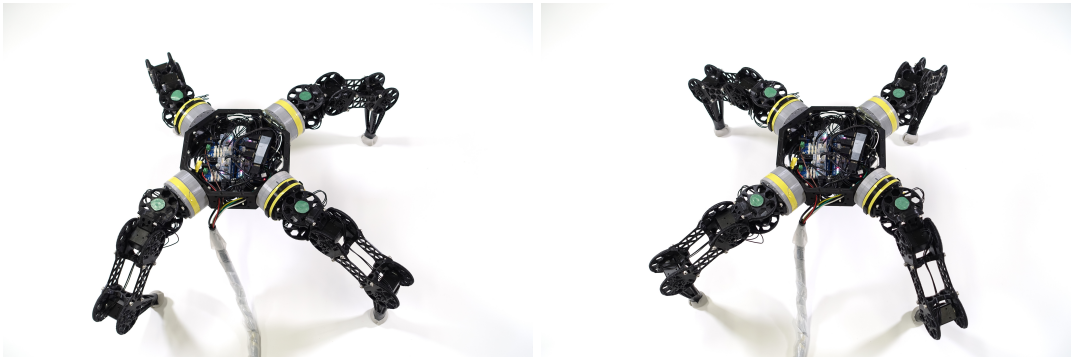
Furthermore, the high-level controller encompasses a body motion planner and a gait generator, essential components for coordinating complex movements and locomotion patterns. Additionally, teleoperation control via a PS4 joystick has been implemented to provide manual control capabilities.

4.4.1 Body Movement

The body position control is managed by the body planner, which plays a critical role in maintaining balance during locomotion and in navigating uncertain terrain. Fig. 4.5 shows the translational movement of the body in four directions, while Fig. 4.6 depicts its rotation around three axes.



(a) Body moves 5 cm to the front. (b) Body moves 5 cm to the back.



(c) Body moves 5 cm to the left. (d) Body moves 5 cm to the right.

Fig. 4.5: Body translational movement.

4.4.2 Gait Generation:

1. **End-effector Trajectory Generator**

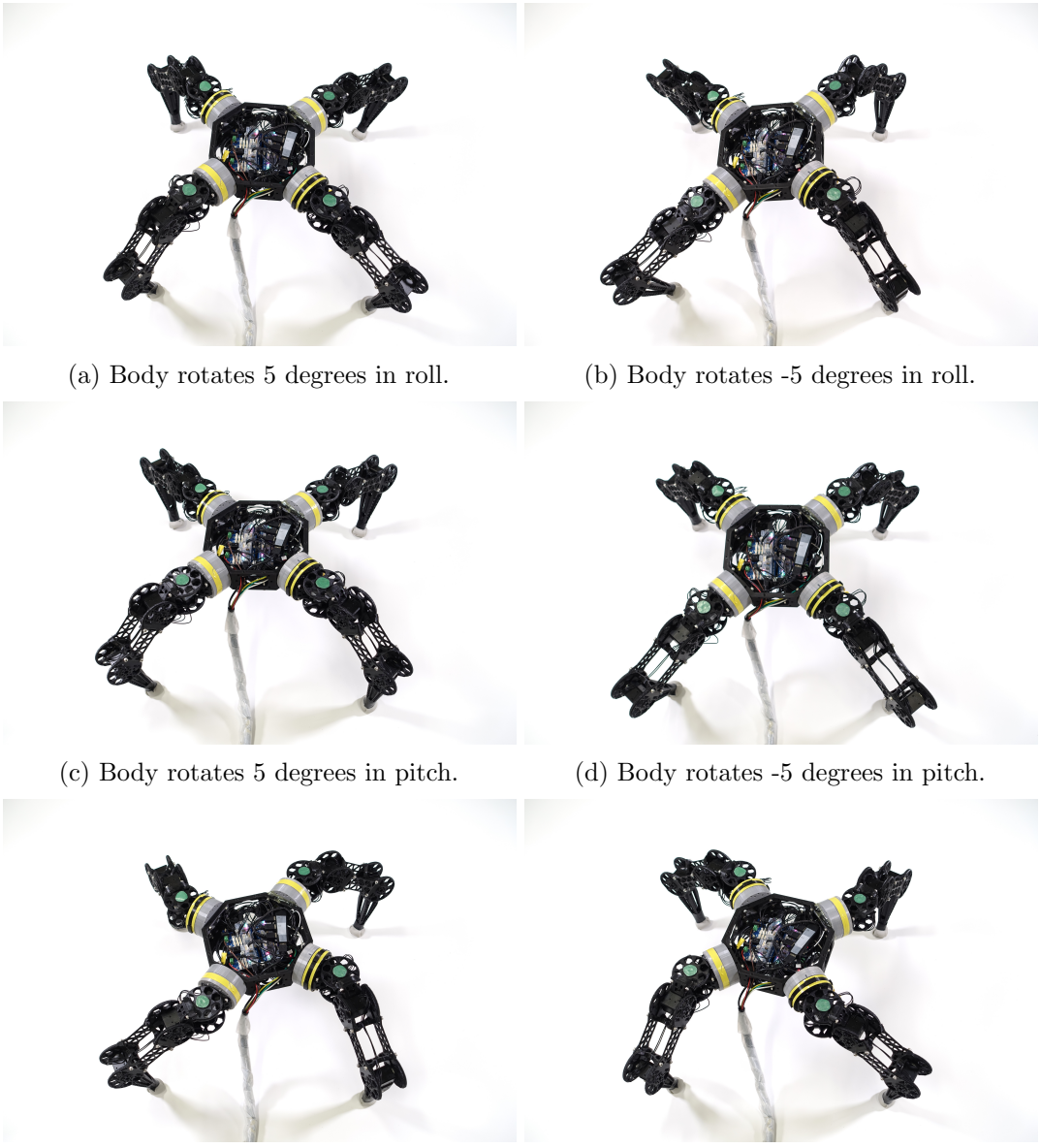
   The trajectory of the legs step is calculated in the gait generator to perform gait patterns and stride vector. Each step along the trajectory involves determining the position of the leg. During the swing phase of each gait pattern, the leg's tip follows a sinusoidal trajectory. By iteratively computing the trajectory for each step, legged robots can generate stable and efficient gaits.

   The trajectory equation for each step is given as:

$$z(t) = S \times \sin\left(\frac{\pi}{L} \times t\right) \tag{4.12}$$

   where:

   - S is step height.
   - L is step distance.
   - t is time step.



(a) Body rotates 5 degrees in roll. (b) Body rotates -5 degrees in roll.

(c) Body rotates 5 degrees in pitch. (d) Body rotates -5 degrees in pitch.

(e) Body rotates 10 degrees in yaw. (f) Body rotates -10 degrees in yaw.

Fig. 4.6: Body rotational movement.

2. **Gait Planner**

With stride vector generation, gait planner plans the swing and stance phases of leg motion, enabling the generation of various gait patterns, including trot and dynamic crawl gait, as shown in Figure Fig. 4.8. The timing of these gait cycles and the desired state of the leg are sent to the end effector's position controller to generate either swing or stance trajectory following the time step of the leg in the gait cycle. The series of snapshots for trot gait and crawl gait are presented in Fig. 4.9 and Fig. 4.10, respectively.

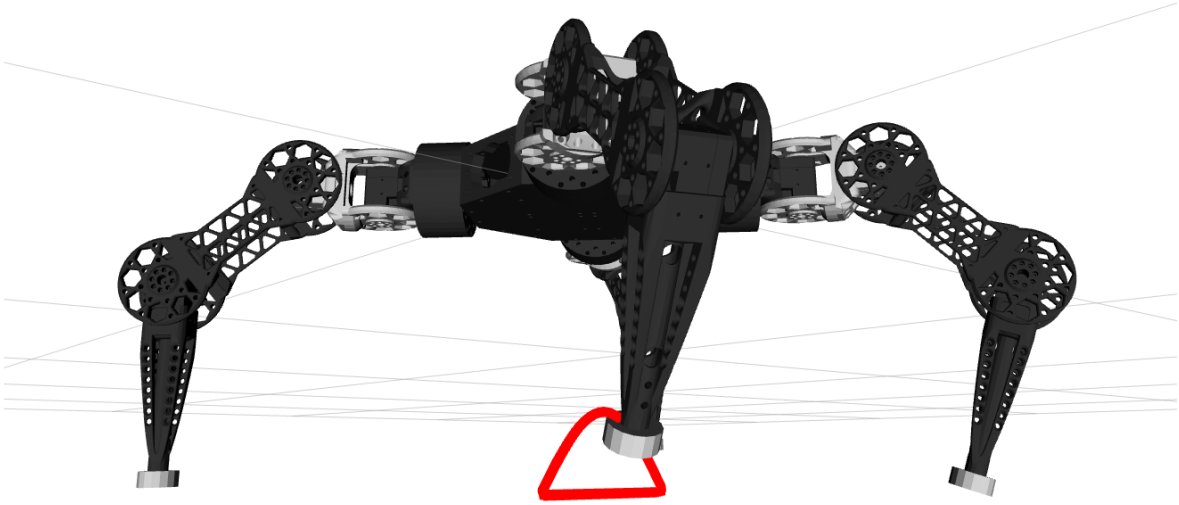


Fig. 4.7: Stride generation with sine function.

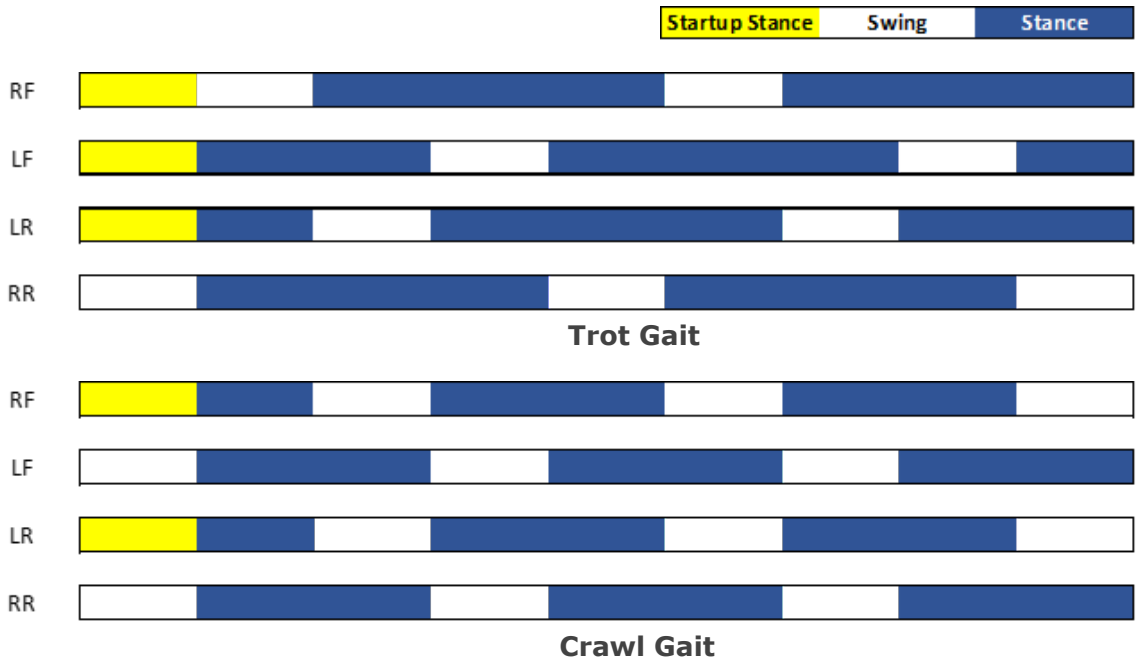


Fig. 4.8: Gait diagram of walking motion.

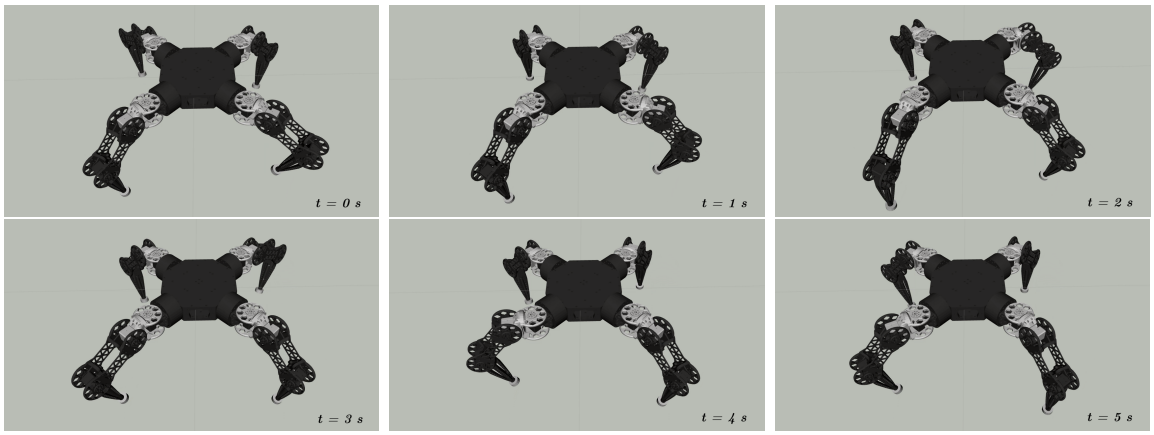


Fig. 4.9: Series pictures of Moonbot walking with trot gait towards the top of the page. The pictures were taken every 1 sex from left to right, top to bottom.

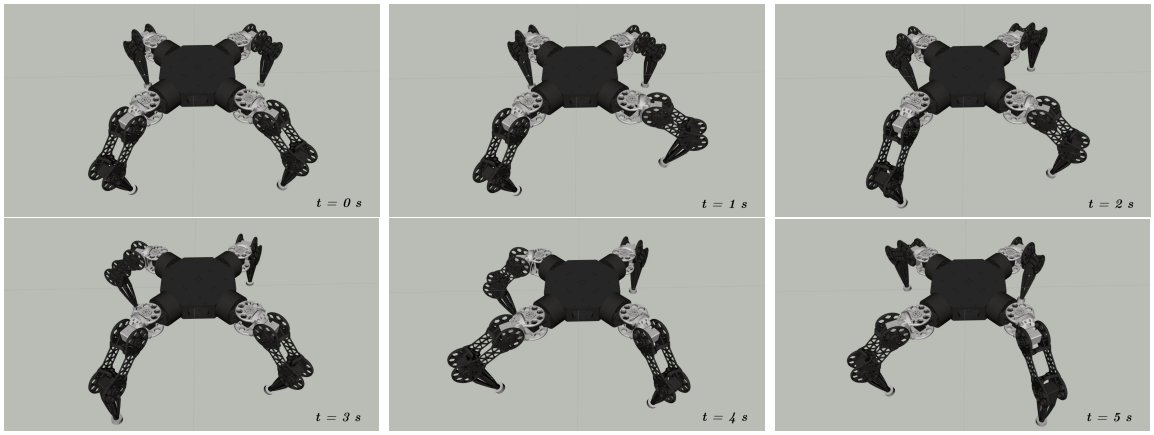


Fig. 4.10: Series pictures of Moonbot walking with crawl gait towards the top of the page. The pictures were taken every 1 sex from left to right, top to bottom.

Chapter 5

Conclusions

5.1 Conclusion

In this research, we have focused on the development of self-recognition and modular control capabilities for Moonbot, a modular legged robot. By providing adaptability to different configurations, Moonbot uses internal sensor to demonstrate the ability to recognize itself and its potential to operate in various configurations. The ROS2-based control framework implemented enables seamless integration of different modules and additional sensors, paving the way for future advancements in Moonbot's capabilities.

Furthermore, the development of a high-level controller for Moonbot's quadruped configuration, both in simulation and real-world scenarios, showcases its operational versatility and robustness. These achievements mark significant milestones towards autonomous and adaptable robotic systems for lunar exploration and beyond.

5.2 Future Improvements Plan

Looking ahead, future work will focus on enhancing Moonbot's control strategies and adaptability through the implementation of reinforcement learning (RL) algorithms, particularly tailored for the lunar surface environment. RL has shown promise in enabling robots to learn and adapt to complex and dynamic environments by continuously improving their decision-making processes based on feedback from the environment. In the context of legged robots, RL algorithms can be utilized to optimize locomotion patterns and adapt to varying terrain conditions, ultimately enhancing the robot's mobility and efficiency. Many researchers have driven into this new strategies to control legged robot, for example, Anymal from ETH Zurich [35], [36].

Furthermore, in the modularity side, the application of deep reinforcement learning (DRL) algorithms will be investigated to optimize the design process of modular legged robots [38], [39]. DRL techniques, which leverage deep neural networks to approximate

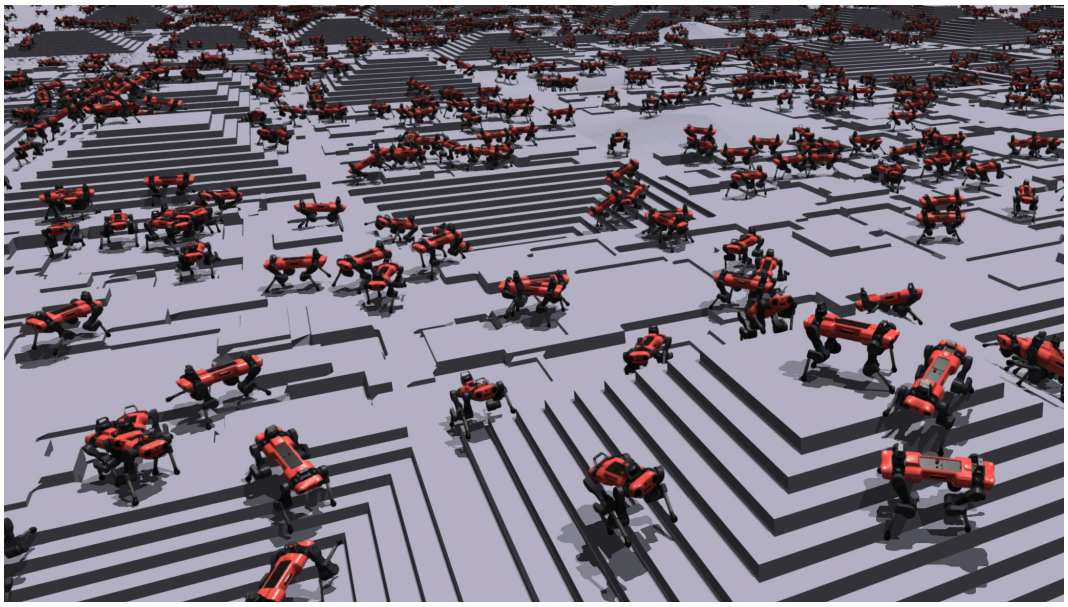
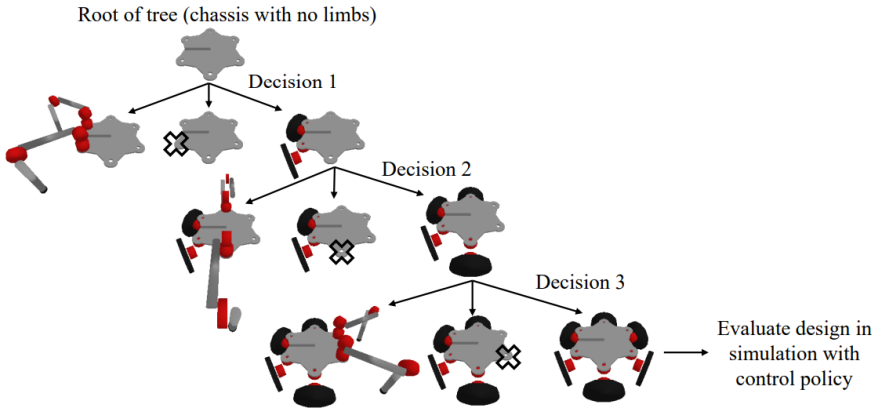


Fig. 5.1: Anymal robot walking training in Isaac Gym environment. [37]

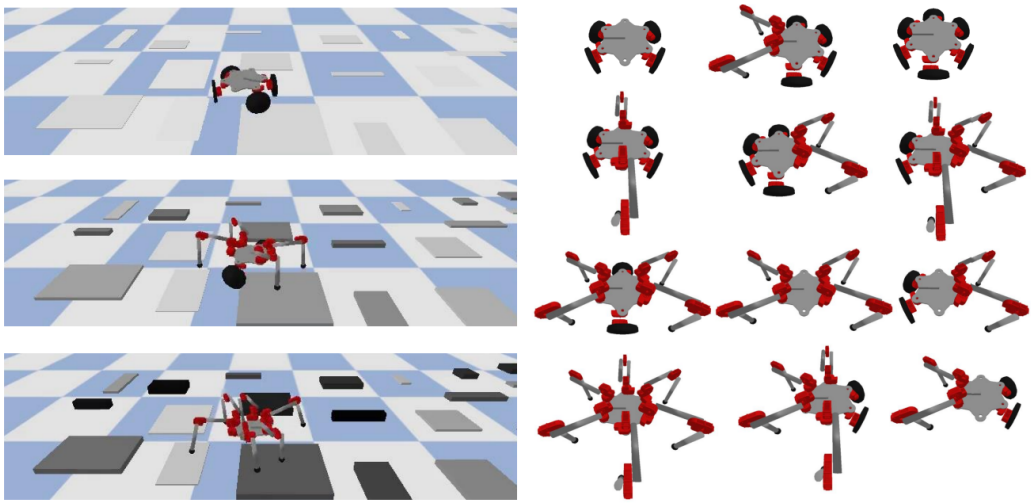
complex functions, offer the potential to revolutionize the way legged robots are designed and controlled. By using DRL, Moonbot and similar robots can autonomously learn and refine their locomotion strategies, leading to more agile and robust robotic systems.

The integration of reinforcement learning techniques into the control and design processes of modular legged robots represents a significant step towards achieving autonomous operation in challenging environments, such as lunar surfaces. By leveraging RL and DRL algorithms, Moonbot and future generations of legged robots can overcome obstacles, navigate complex terrains, and accomplish tasks with greater efficiency and adaptability.

In conclusion, the combination of reinforcement learning, modular robotics, and space exploration holds immense potential for advancing the field of robotics and unlocking new capabilities for autonomous systems. By bridging the gap between theoretical research and practical application, we can pave the way for the next generation of robotic explorers, capable of operating effectively in the harsh and unpredictable environments of space.



(a) Design space tree for modular mobile robot, using deep neural network.



(b) Evaluation of modular designs in simulation across terrains of varying roughness, showcasing the adaptability of different robot configurations.

Fig. 5.2: Modular mobile robot design selection with deep reinforcement learning. [39]

Bibliography

[1] "Apollo Lunar Missions".

[2] NASA, "NASA Looks to Advance 3D Printing Construction Systems for the Moon and Mars", October 2020.

[3] Mark A. Post, Xiu-Tian Yan, and Pierre Letier, "Modularity for the future in space robotics: A review", *Acta Astronautica*, Vol. 189, pp. 530–547, 2021.

[4] Martin Nisser, Leon Cheng, Yashaswini Makaram, Ryo Suzuki, and Stefanie Mueller, "ElectroVoxel: Electromagnetically Actuated Pivoting for Scalable Modular Self-Reconfigurable Robots", In *ICRA 2022*, pp. 4254–4260, 2022.

[5] MIT Space Resources, "WORMS - Water-Only Resource Mining System", 2022.

[6] "JST Moonshot Project Goal 3B: Realization of AI robots that autonomously learn, adapt to their environment, evolve in intelligence and act alongside human beings, by 2050".

[7] Philip Arm, Radek Zenkl, Patrick Barton, Lars Beglinger, Alex Dietsche, Luca Ferrazzini, Elias Hampp, Jan Hinder, Camille Huber, David Schaufelberger, Felix Schmitt, Benjamin Sun, Boris Stolz, Hendrik Kolvenbach, and Marco Hutter, "SpaceBok: A Dynamic Legged Robot for Space Exploration", In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6288–6294, 2019.

[8] D. Daniel, B. Krogh, and M. Friedman, "Kinematics and open-loop control of an ilonator-based mobile platform", In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, Vol. 2, pp. 346–351, 1985.

[9] McKerrow Philip, *Introduction to Robotics*, Addison-Wesley Publishing Co., 1991.

[10] Bares John, Hebert Martial, Kanade Takeo, Krotkov Eric, Simmons Reid, and Whittaker William, "Configuration of an autonomous robot for Mars exploration",

In *World Conference on Robotics Research: The Next Five Years and Beyond*, Vol. 1, pp. 37–52, Gaithersburg, 1989.

[11] Julian Whitman, Raunaq M. Bhirangi, Matthew J. Travers, and Howie Choset, "Modular Robot Design Synthesis with Deep Reinforcement Learning", In *AAAI Conference on Artificial Intelligence*, 2020.

[12] A. Abdel-Rahman, C. Cameron, B. Jenett, and et al., "Self-replicating hierarchical modular robotic swarms", *Commun Eng*, Vol. 1, p. 35, 2022.

[13] Joohyung Kim, Alexander Alspach, and Katsu Yamane, "Snapbot: A reconfigurable legged robot", In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5861–5867, 2017.

[14] Cabinet Office, Government of Japan, "Moonshot Project: Realization of AI Robots for Lunar Exploration", Accessed: 2023.

[15] Kevin G. Gim and Joohyung Kim, "Snapbot V2: a Reconfigurable Legged Robot with a Camera for Self Configuration Recognition", In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4026–4031, 2020.

[16] Julian Whitman, Shuang Su, Stelian Coros, Alex Ansari, and Howie Choset, "Generating gaits for simultaneous locomotion and manipulation", In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2723–2729, 2017.

[17] Benjamin Tam, Fletcher Talbot, Ryan Steindl, Alberto Elfes, and Navinda Kottege, "OpenSHC: A Versatile Multilegged Robot Controller", *IEEE Access*, Vol. 8, pp. 188908–188926, 2020.

[18] FlashForge, "FlashForge 3D Printer - Adventurer 3", 2023.

[19] ROBOTIS, "ROBOTIS: Dynamixel Servos", `https://www.robotis.us/`, 2023.

[20] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng, "ROS: an open-source Robot Operating System", In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[21] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild", *Science Robotics*, Vol. 7, No. 66, p. eabm6074, 2022.

[22] William Woodall, "ROS on DDS", Online documentation.

[23] ROS 2 Control Working Group, "ROS 2 Control", 2023, Online documentation.

[24] ROS Controls, "ros_control".

[25] ROS Wiki Contributors, "combined_robot_hw".

[26] ROS 2 Design, "ROS 2 Node Lifecycle", 2015, Date Written: June 2015.

[27] ROBOTIS, *ROBOTIS Dynamixel SDK*, 2010.

[28] "Dynamixel Workbench", `https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_workbench/`, Accessed: December 2023.

[29] ROBOTIS, "Dynamixel SDK - Python Ping Protocol 2.0", 2023.

[30] ROBOTIS, "Dynamixel Wizard2".

[31] ROS 2 Control Authors, "ROS 2 Controllers Documentation", `https://control.ros.org/master/doc/ros2_controllers/doc/controllers_index.html`.

[32] "Gazebo: An open-source 3D robotics simulator", Available at `http://gazebosim.org`, 2002.

[33] Kentaro Uno, Naomasa Takada, Taku Okawara, Keigo Haji, Arthur Candalot, Warley F. R. Ribeiro, Kenji Nagaoka, and Kazuya Yoshida, "HubRobo: A Lightweight Multi-Limbed Climbing Robot for Exploration in Challenging Terrain", In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 209–215, 2021.

[34] Nipun Dhananjaya Weerakkodi Mudalige, Iana Zhura, Ildar Babataev, Elena Nazarova, Aleksey Fedoseev, and Dzmitry Tsetserukou, "HyperDog: An Open-Source Quadruped Robot Platform Based on ROS2 and micro-ROS", In *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 436–441, 2022.

[35] Myeongseop Kim, Jung-Su Kim, and Jae-Han Park, "Automated Hyperparameter Tuning in Reinforcement Learning for Quadrupedal Robot Locomotion", *Electronics*, Vol. 13, p. 116, 12 2023.

[36] Vassilios Tsounis, Mitja Alge, Joonho Lee, Farbod Farshidian, and Marco Hutter, "DeepGait: Planning and Control of Quadrupedal Gaits Using Deep Reinforcement Learning", *IEEE Robotics and Automation Letters*, Vol. 5, No. 2, pp. 3699–3706, 2020.

[37] Legged Robotics Group, "Legged Gym: OpenAI Gym Environment for Legged Robots", GitHub repository.

[38] Julian Whitman, Raunaq M. Bhirangi, Matthew J. Travers, and Howie Choset, "Modular Robot Design Synthesis with Deep Reinforcement Learning", In *Proceedings of 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pp. 10418 – 10425, February 2020.

[39] Julian Whitman, Matthew J. Travers, and Howie Choset, "Modular mobile robot design selection with deep reinforcement learning".

Acknowledgements

This book was created at the Space Robotics Laboratory, Faculty of Engineering, Tohoku University and supported by JST Moonshot R&D Program, Grant Number JPMJMS223B.

I extend my sincere appreciation to the Space Robotics Laboratory of Prof. Kazuya Yoshida, where I have had the privilege of conducting my research. I am deeply grateful for the mentorship, guidance, and opportunities provided by Prof. Yoshida and the lab members, which have been invaluable to my growth and development in the field of robotics.

I am indebted to Assistant Prof. Uno Kentaro and Prof. Shreya Santra for their invaluable support and guidance throughout my journey. Assistant Prof. Uno's insights and advice on legged robotics have been instrumental in shaping my understanding of this complex field, while Prof. Shreya's support in ideation and thesis writing has been invaluable.

I would like to express my heartfelt thanks to Danish AI for his exceptional work in developing the hardware design of Moonbot. Additionally, I extend my gratitude to Pascal Pama and Torii Keigo for their support during experiments and their insightful comments on my work.

Special thanks are due to Gustavo Diaz for his significant contributions to Moonshot project team. His advice on robot software development and debugging has been invaluable to our progress.

I am deeply grateful to the Limbing Robotics team for their invaluable contributions to my understanding of legged robotics. Their expertise and knowledge-sharing have been instrumental in shaping my understanding of this complex field. Special thanks are due to the Limbero climbing quadruped robot team, whose insights and advice on Moonbot's software development have been invaluable to our project's success.

I would also like to express my sincere appreciation to Uchida Akiyoshi of the Orbital Robotics team for his assistance with ROS2 control. His guidance and support have been instrumental in navigating the intricacies of robotic control systems, enabling us

to enhance Moonbot's capabilities.

Additionally, I extend my gratitude to Nagaoka Keita for his support with video shooting, which has been essential in documenting and sharing our project's progress effectively.

To each of these individuals and teams, I offer my deepest thanks for their invaluable contributions, guidance, and support, which have significantly enriched my journey in robotics and contributed to the success of the Moonshot project.

February, 2024

Tharit Sinsunthorn