## 1. Introduction and Problem Definition

The problem is to design a hexapod robot that can walk on both plain ground and terrain of reasonable roughness e.g. a meadow, a beach with pebbles. The robot has to be remote controlled, and can self-adjust its gait pattern in response to different controller commands and varying terrains.

## 2. Design Overview

This design is a six-legged robot with an overall geometry of 45 cm (in diameter), making it a robust platform ideal for carrying loads on unpaved roads. The robot is designed for user control through a customized remote controller, while simultaneously autonomously adjusting its pose and balancing its body in response to varying terrain. The robot's mechanical structure is divided into a base and six legs. The base comprises two large support panels (Figure 1), each 3D printed using PLA. The bottom panel houses all the electrical components and serves as the mounting bracket for the legs, while the top panel is installed using a snap-fit mechanism. This top panel is interchangeable, allowing users to replace it with different shapes to fit various needs.

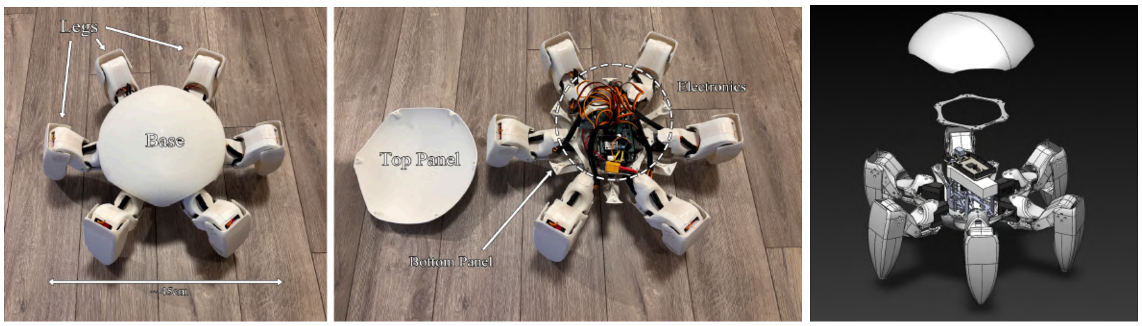


Figure 1. Robot

Each leg is outfitted with three joints, with each joint driven by a dedicated DC servo motor, giving the entire robot a total of 18 degrees of freedom. As illustrated in Figure 2, Motor 1 controls the forward and backward swing (comparable to a human thigh), while Motors 2 and 3 manage the leg flexion and extension (comparable to a human calf and ankle). Here, the tip of each leg is defined as the **end effector**. Controlling the robot's locomotion relies on an embedded system that processes incoming signals, continuously recalculates and plans motion trajectories (of the end effector), and computes and deploys precise joint rotations in real time. This ensures that the robot maintains proper ground contact while following a specific locomotion pattern to drive movement.

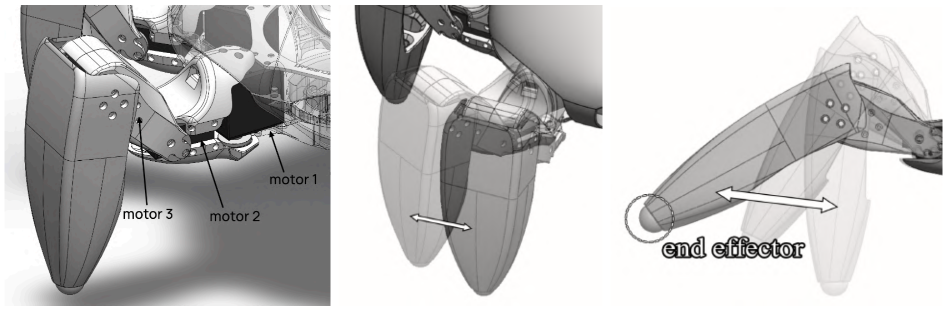


Figure 2. Leg

A complete embedded system has been designed, which includes both the robot body and a dedicated remote controller. A high-level block diagram is presented in Figure 3, the reader is encouraged to refer to the system map and layout diagram for visual reference as each subsystem is described.

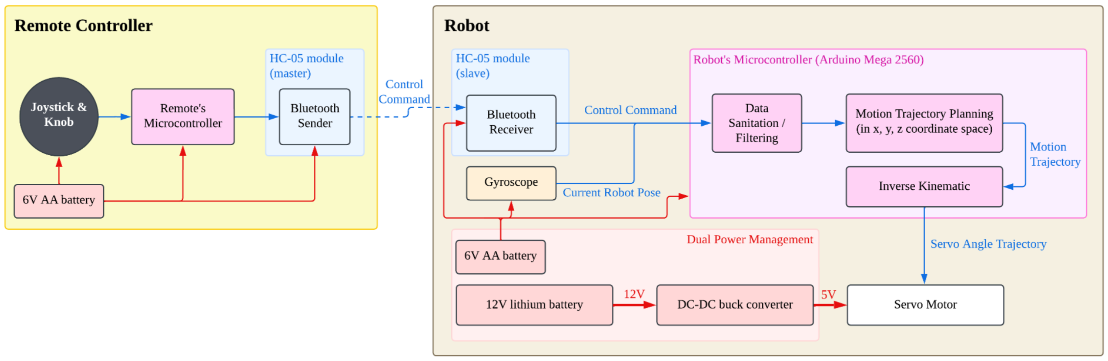


Figure 3. Embedded system overall architecture

The remote controller, powered via 4 AA batteries and equipped with joysticks and knobs, allows the user to directly control the robot's movement pattern—whether walking, turning, or swaying its body. Joystick signals are processed by the remote's built-in microcontroller and transmitted via a Bluetooth module to the robot. The robot's motion is managed through the combined efforts of a Bluetooth module and an onboard gyroscope module. The Bluetooth module processes user inputs, while the gyroscope module is responsible for self-sensing and adjustment, that is, stabilizing the robot's body to prevent excessive tilting before any locomotion commands are executed. Data from both modules is thoroughly sanitized and filtered through specially designed filters to reduce noise. Upon receiving a control command, the robot uses posture information from the gyroscope to first adjust and stabilize its body, then generate coordinated trajectories for each leg in the XYZ coordinate space. These trajectories are then converted into target joint angles using inverse kinematics, and the calculated angles are sent to 18 servo motors via a PWM (Pulse Width Modulation) signal to execute the movement. Notably, these trajectories are both calculated and updated in real-time, ensuring adaptive locomotion and dynamic control.

One of the key highlights of the design is the implementation of a dual power supply system. To meet the high current demands of the 18 servos during motor starts, a 12V lithium battery is stepped down to 5V using a DC-DC buck converter to power all the servos (see Section 3.1.1). Meanwhile, other peripherals such as the MCU and gyroscope are powered by four AA batteries.

Additionally, to streamline the wiring and efficiently manage the high current induced by motors, the team leveraged their experience from the MIE346 course to design a custom 4-layer power distribution board mounted on the MCU (Figure 4). This board not only simplifies plug-and-play connectivity for all peripherals but also ensures robust power delivery (for details, see Section 3.1.3).

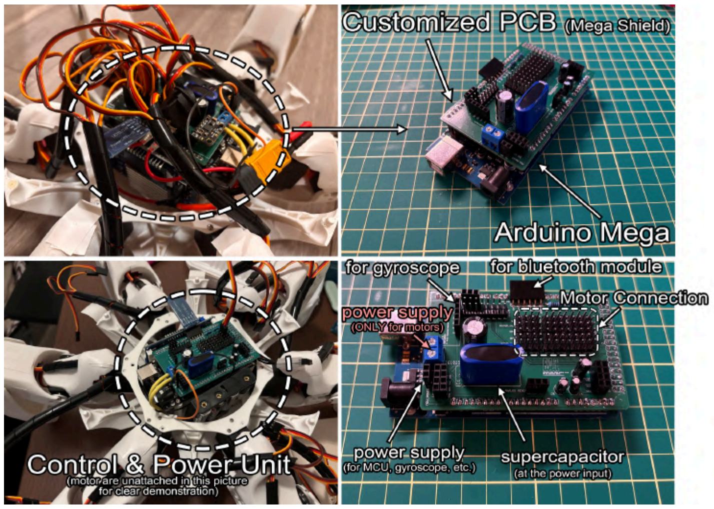

Customized PCB (Mega Shield)
Arduino Mega
for gyroscope
for bluetooth module
Motor Connection
power supply (ONLY for motors)
power supply (for MCU, gyroscope, etc.)
supercapacitor (at the power input)
Control & Power Unit
(motor are unattached in this picture for clear demonstration)

Figure 4. Customized 4-layer power distribution PCB

## 3. Hardware Design

In this section, the detailed composition of the embedded system is presented, covering the design and selection of components. Each hardware subsystem is introduced in the **sequence that a user follows during their interaction with the robot**. A complete Bill of Material (BOM) is presented in Appendix B.

### 3.1.1 Robot Power Management

Before interacting with the robot, the user has to power the system. The embedded system includes multiple modules, including the microcontroller, motors, Bluetooth receiver, and gyroscope. To accommodate the power demands of these modules and ensure system stability, a dual power management architecture was adopted, dividing the system into **control** and **power circuits**. The control circuit supplies low-current modules such as the MCU, Bluetooth, and gyroscope. This section is powered by four 1.5V AA batteries connected to the MCU's Vin pin, the on-board regulator steps it down to steady 5VDC, then distributes power to peripherals. On the other hand, the power circuit dedicated to supplying high-current motors employs an 11.1V, 2250mAh, 25C lithium battery ( theoretical continuous current capability up to ~56.25A), which is stepped down via a DC-DC buck converter rated for up to 12A, delivering a stable 5V power source to the motors.

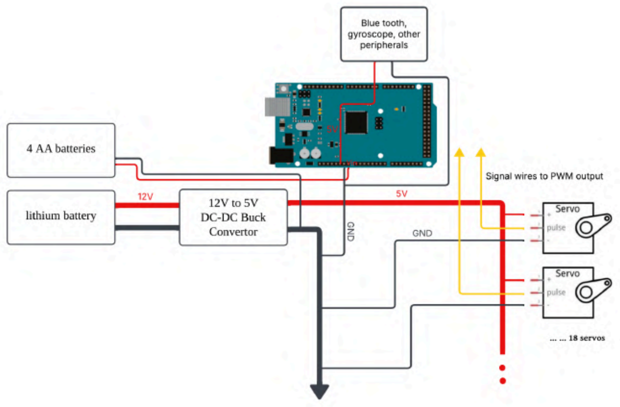

Blue tooth, gyroscope, other peripherals
4 AA batteries
lithium battery
12V
12V to 5V DC-DC Buck Convertor
5V
GND
GND
Signal wires to PWM output
Servo
pulse
Servo
pulse
... ... 18 servos

Figure 5. Power management schematic

The rationale behind the dual power system lies in **three** core considerations. First is the need for current supply adequacy. Although all modules operate at 5V, the current demands vary significantly, particularly for 18 servos. During startup, the absence of back electromotive force (EMF) combined with the extremely low winding resistance causes a surge in current; this is further amplified by the need to overcome the load's inertia. During the design process, test results show that peak (motor) bus current can reach up to 11A during motor startup or rapid speed changes. In contrast, the Arduino Mega's onboard 5V output pin can only supply approximately 500mA, far from sufficient to handle motor startups. Thus, an independent power source for the motor subsystem is essential.

Second is the issue of electrical interference. Initially, the team employed a single power supply unit, wherein the motors and control modules shared the same power rail. However, when all 18 motors moved simultaneously, the abrupt current increase, combined with the resistance of wires and connectors, caused a transient voltage drop — at times below the MCU's minimum operational threshold and caused severe instability, including unexpected MCU resets and loss of control signals. This type of interference proved particularly critical during intensive operations, such as rapid robot movement. To mitigate this, the power supply to the motors and control modules was physically isolated, thereby eliminating current-induced disturbances and enhancing the system's reliability.

Third, the dual power system offers a cleaner and more manageable hardware architecture. Maintaining system stability under a unified power source would require the introduction of additional voltage regulation and distribution circuitry, all of which add complexity, cost, and potential risk of failure. In contrast, the dual power design simplifies modular debugging and improves both maintainability and practical engineering implementation.

This specially designed power system is implemented on a customized power distribution board, which will be detailed further in Section 3.1.3..

### 3.1.2 Remote Controller Design

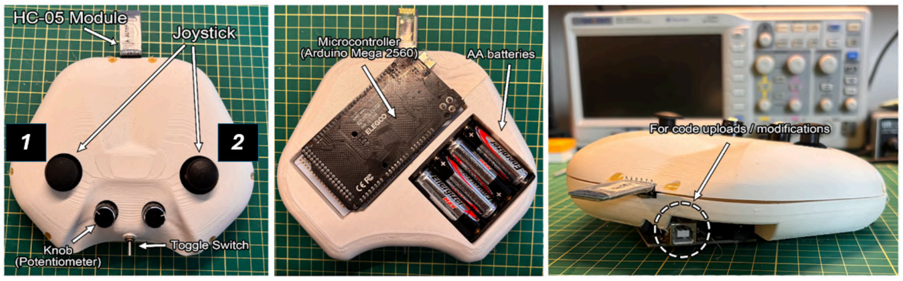

HC-05 Module
Joystick
1
2
Knob (Potentiometer)
Toggle Switch
Microcontroller (Arduino Mega 2560)
AA batteries
For code uploads / modifications

Figure 6. Remote Controller

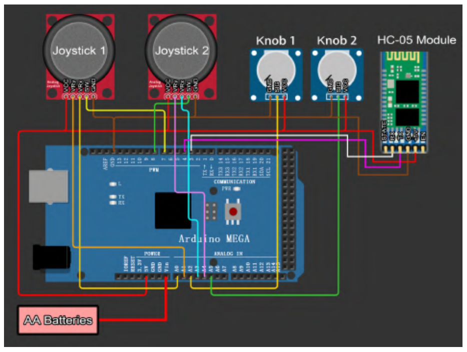


Figure 7. Remote Controller Schematic

After the robot is powered on, it can be controlled using a remote controller (Figure 6), with a complete schematic shown in Figure 7. The user can turn on the remote via a toggle switch, which activates the power supply consisting of four 1.5V AA batteries—chosen for their ready availability and ease of installation. The battery voltage is introduced into the system via the Arduino's Vin pin, where the onboard regulator steps it down to a stable 5V; this regulated voltage is then distributed to all components (Vcc) through the Arduino Mega's 5V pin. The remote controller is equipped with two joysticks and two knobs (potentiometers). Analog output signals (via SIG pins) from these components are read by the Arduino's analog input pins (A0–A5) and converted by the onboard ADC into digital values ranging from 0 to 1023; these values are then filtered at the algorithm level to form the robot's commands (see Section X). For wireless control, an HC-05 Bluetooth module was selected due to its availability and ease of programming. Operating in a master/slave full-duplex configuration, the master module is mounted on the remote controller, with its RX and TX pins connected to the Arduino's virtual serial pins (3 and 4). The module communicates at a baud rate of 9600 and maintains a fixed update rate of 100 Hz. At each time step, joystick and knob inputs are converted into a string format and transmitted to the robot's slave Bluetooth module for reception. The use of virtual serial ports also provides the flexibility to connect the remote controller to a computer via hardware serial when necessary, facilitating debugging without disrupting wireless communication.

### 3.1.3 Sensing, Control, and System Integration

After the remote controller sends a signal, the robot first receives these signals, calculates the motion trajectory, and transmits the control commands to servo motors. To integrate all modules, the system uses Arduino Mega 2560 (16 MHz) as its MCU, primarily due to its abundant interface pins that can support a large number of peripherals. In this system, 18 servo motors require 18 PWM outputs, while there must also be sufficient pins reserved for the Bluetooth module, gyroscope, and other devices—requirements that an Arduino Uno or Nano cannot meet. Moreover, the Arduino platform offers strong compatibility with official libraries and easy programming (Arduino IDE). Considering that this project does not need to store large amounts of data and its design goal is to achieve dynamic movement rather than the micrometer-level absolute control precision demanded by industrial robot manipulators, a higher-performance processor such as the ESP32 would be overkill. Additionally, the Arduino Mega's support for custom Mega Shields provides greater flexibility for system expansion and module integration.

Within the robot system, two modules are responsible for receiving external signals. First, the HC-05 Bluetooth receiver, operating as a slave, receives control signals from the remote controller and transmits the data to the robot's onboard microcontroller via a hardware serial port at a baud rate of 9600. The decision to use hardware serial rather than virtual serial arose because initial tests with virtual serial resulted in unstable PWM duty cycles for motors — likely due to the noises induced by the motors on the virtual serial port. Meanwhile, the robot continuously reads data from the gyroscope module to obtain its current body posture, enabling it to automatically adjust its pose to fit uneven terrain. For instance, if one side of the robot tilts, the system can use the gyroscope data to self-correct and restore a level orientation. MPU6050 is selected as the gyroscope, which is widely available, cost-effective, and features three-axis sensing (yaw, pitch, and roll). Since the robot primarily operates within a single plane and does not jump vertically, only the pitch and roll angles need to be monitored, making this module entirely sufficient. The MPU6050 connects to the Arduino's SDA and SCL pins via the I2C protocol. Ultimately, the information from both the gyroscope and Bluetooth modules is processed collectively to calculate the motion trajectories for each leg (see Section 4.1 for details).

Due to the complexity of the system, it is challenging to wire numerous components together. Using simple jumper wires on a breadboard would be impractical, as the power bus for the 18 motors may carry up to 11A, which far exceeds the current-carrying capacity of standard 22AWG wires. To address these issues, the team designed a four-layer PCB (Arduino Mega Shield) that integrates the control and power circuits into a single board and mounts directly onto the Arduino's headers (Figure 7). The dual power supplies can be directly plugged into the terminal on the left side of the board.

The internal structure of the PCB is divided into a power layer, a GND layer, and two signal layers (Figure 8). The power layer features a main power bus with a trace width of 270 mil, which splits into two 130 mil branches dedicated to powering the 18 motors. At its input, a 0.6F supercapacitor is installed in parallel with a 1000µF capacitor to mitigate voltage drops that may occur during frequent motor operation (Appendix C) (please see the demonstration video for details).

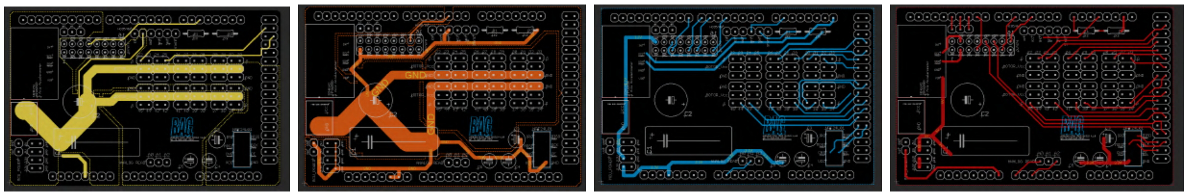


Figure 8. PCB layers (prior copper pour)
(left to right: Vcc layer, GND layer, Signal layer 1, Signal layer 2)

The GND layer is routed with 270 mil traces to connect the grounds of all components, employing a star grounding method to minimize potential parasitic effects. The signal layers use trace widths ranging from 14 to 40 mil. Due to the large number of signal traces, the signal routing is split into two layers to avoid crosstalk. The control signals for the 18 motors are connected to the Arduino Mega's pins 22 through 45 (with the PCB designed to support up to 24 motors, providing expansion possibilities for the future). The pins for the Bluetooth and gyroscope modules are also routed on these signal layers.

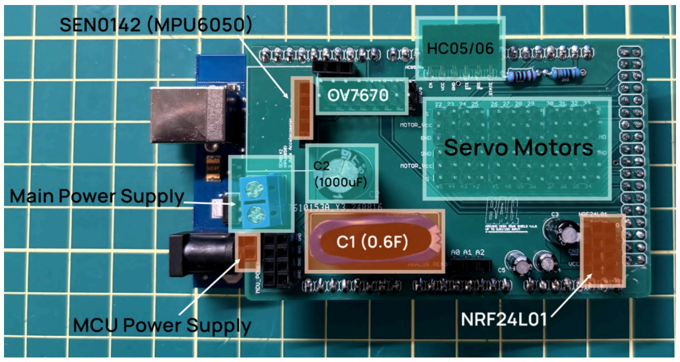


Figure 9. PCB top view

Finally, all layers are copper-poured. The PCB design includes additional expansion slots, such as interfaces for a camera module (OV7670) and a wireless transmission module (NRF24, using the SPI protocol), thereby offering flexibility for future functional enhancements and applications in other projects. Detailed PCB design schematic and figures please see Appendix D.

### 3.1.4 Motion Implementation

All control commands ultimately translate into joint rotations to move the robot body. Compared with DC motors or stepper motors, servo motors were selected for their integrated closed-loop feedback system that enables precise angle control using a simple PWM signal. In contrast, DC/stepper motors generally require additional encoders or driver boards to achieve closed-loop control, thereby increasing the design complexity. Moreover, servo motors can be directly controlled using Arduino's official servo library, making the coding process straightforward.

In this project, 18 RDS3225 motors were used, each supporting PWM control with a 5V power line and rated for a maximum torque of 25 kg·cm. One of the key objectives of this project is to enable the robot to carry and transport loads; therefore, high-torque motors are essential to ensure that the robot is not overwhelmed when under load. To quantify the load-bearing capability of these motors, the team has modeled each leg into a three-bar linkage for static force analysis (Appendix E) and revealed that the current motor selection theoretically supports a maximum load of approximately 10.7 kg.

## 4. Controller and Firmware Design

### 4.1 Finite State Machine

The controller algorithm at its core is a **finite state machine** that switches the robot's operational mode based on controller commands received from the HC-05 Bluetooth receiver, with adjustments based on the MPU6050 gyro data.

Specifically, there are three possible overarching locomotion modes available – a twist-based turning, a dynamic heading control mode, and a manual override mode. The gyro reading is used to fine-tune the motion commands based on its current pose, specifically its current rotation around two planar axes parallel to the ground, i.e. the pitch and roll angles. When these readings indicate a tilt (a value greater than 0°), the control system automatically adjusts the motion commands to **level the robot's body**, ensuring a stable and balanced gait.

The essence of locomotion is to control the robot's leg to follow a specific planned trajectory. To calculate and generate the trajectories, multiple reference frames (coordinate systems) are established, providing the necessary spatial context for motion planning. At the center of the robot's main body, a body frame labeled *{B}* where axis *z_base* points out of the page is set (Figure 10). This is fixed to the robot and rotates with the robot. For **each** leg, two additional frames are attached, a leg frame *{L}* at the base of each leg. A total of six *{L}* frames from 1 to 6, and six corresponding tip frames *{E}* is defined throughout the entire body. By defining the motion trajectory in the tip frame *{E}*, the trajectory coordinates can be transformed into *{L}* using matrix transformation, then using the inverse kinematic formula to compute the motor angles and deploying PWM signals.

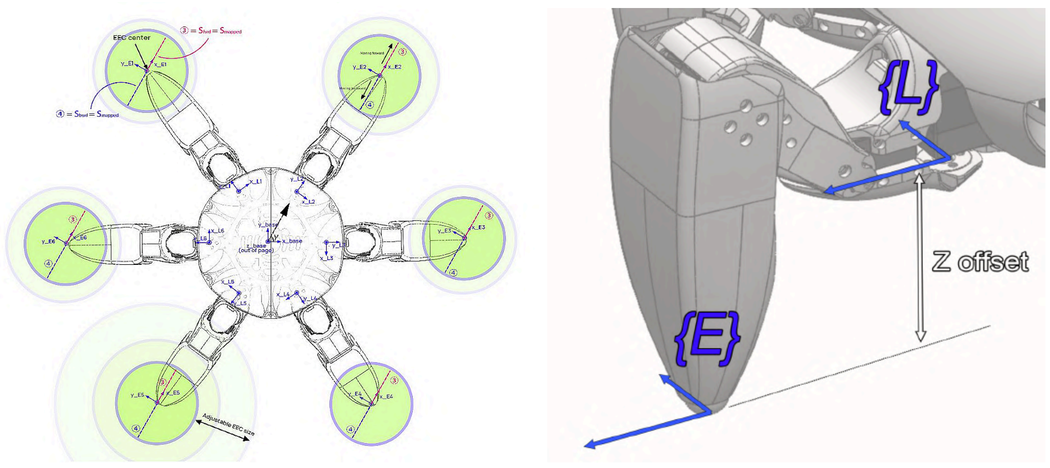
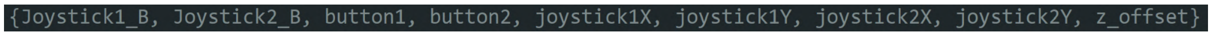


Figure 10. Frames used on the robot

When a valid motion direction is received — defined by γ, the angle between the controller's heading vector and the *x_base* axis (Figure 10 *{B}*) — each tip frame *{E}* rotates so that its *x axis* aligns with that direction. Consequently, all movements occur uniformly along this direction, enabling the robot to walk in any orientation. On top of this, each leg frame is offset from its corresponding tip frame in **Z offset** (Figure 10), where this offset is a combination of both the current configuration of the leg and an externally induced offset from the terrain itself that is captured by the gyro sensor.

Additionally, a description of the data structure received by the HC-05 Bluetooth receiver lends itself well to the subsequent discussion as well. As shown in Figure 11, there are two joysticks and two knobs on the controller. The left knob controls the vertical height of the main body from the ground. The two joysticks produce a planar movement in the horizontal plane. This is captured by two variables for each joystick – `joystick_1_x`, `joystick_1_y`, `joystick_2_x`, and `joystick_2_y`. Pressing the joysticks also generates a button, or boolean, signal. Therefore, two additional variables are `joystick_1_button` and `joystick_2_button`, with 0 meaning pressed and 1 meaning unpressed.



```
{Joystick1_B, Joystick2_B, button1, button2, joystick1X, joystick1Y, joystick2X, joystick2Y, z_offset}
```

Figure 11. Transmitted data structure

Turning to the finite state machine, the most common and most important locomotion mode is where the robot's direction and its range of motion are both determined by the joystick input, specifically joystick 1 (Figure 11). To enable this mode, joystick 1 is unpressed. The heading in this mode is determined from the tangent values of the x and y inputs of the joystick 1 data, and its range of motion is determined from a linear mapping from the range of motion of the joystick to that of the gait pattern. That is, the more the joystick is pushed (essentially the Euclidean distance of the X and Y inputs), the longer and taller each step would be, and since the time taken for each step remains constant, the faster. The entire gait pattern from all six legs is aligned with the direction command γ by rotating the original default leg and tip coordinate frames by the angle γ. As the course is not a robotics course, the details of the subsequent gait pattern are provided in Appendix F. The locomotion in this mode is further adjusted to take account of the current terrain. This is achieved by adjusting the Z offset (Figure 10) for each leg tip frame with the roll and pitch angles sensed from the gyro. The principle behind this adjustment is that based the roll and pitch about the base X and Y axes, respectively (this is defined in the base frame *{B}* because the gyro is installed on the main body and its readings are thus defined in the said frame), a tilt vector can be formed and its direction can be defined as shown below.

$$input\_vector \;=\; \{roll\_deg,\, pitch\_deg\}$$

$$tilt\ direction\ =\ atan2(roll\_deg,\ pitch\_deg)$$

There are two additional possible settings that stem from this default state. First, if the left joystick is pressed down, an **override** to the previous dynamic heading determination is triggered. In this state, the heading is hard-coded to be aligned with the Y-axis of the base frame. This is achieved by setting all local orientations of the leg and tip frames to 0, meaning complete alignment with that of the base frame.

Aside from the above two states, when the robot has to turn by a large angle and thus change its heading beyond the mechanical range of motion allowed from each leg, it is capable of entering a state of in-place rotation i.e. spinning around its center, called "Twist". In this mode, **adjacent groups of legs push in opposite directions radial to a circular profile around the center of the robot.** Put more intuitively, each pair of legs try to walk around the center, but one leg pushes forward in its tip frame, the other leg pushes backward, thus enabling the robot to rotate in place around its center. The range of motion of the rotation is controlled by the X component of the right joystick. The more the right joystick is pushed to the left, and longer and taller each step would be. This is also achieved by a similar linear mapping from the range of motion of the joystick to that of the legs of the robot.

Lastly, given the desired XYZ position of each leg in a given step, an inverse kinematic formulation is used to determine the amount of rotation desired from each motor. The resultant rotation commands for all 18 servo motors are then sent to their respective motors using the `servo.write(angle)` function supported by the `Servo.h` stand library.

## 4.2 Data Filtering

The controller employs Kalman filtering at key points in its data processing pipeline to enhance motion stability and responsiveness. The discussion below outlines the use of Kalman filters in three aspects: (1) controller inputs, (2) gyro inputs, and (3) motor command outputs. Each use case serves a unique function, contributing to the overall smoothness and adaptability of the robot's gait.

### 4.2.1. Controller Inputs

Controller inputs are read from two joysticks connected via Bluetooth, providing heading, range of motion, which also determines the scaling of speed, and button states for mode switching. These inputs are **inevitably noisy** due to small fluctuations in joystick readings, electrical jitter, or random movements of the hand. To address this, Kalman filters are applied to smooth out the joystick axes. This filtered input is used in real-time by the twist motion and default gait generation functions.

The result is **more consistent step sizes, smoother start and stop of movement, and reduced motor jittering** from small hand adjustments.

### 4.2.2. Gyro Inputs

The MPU6050 IMU provides raw accelerometer data that is used to calculate the robot's roll and pitch angles. These angles enable real-time Z-offset compensation for each leg. However, accelerometer data is also inherently noisy and highly sensitive to micro-vibrations of the robot's main body.

To produce reliable tilt estimates, the system uses dedicated Kalman filters (`kf_roll` and `kf_pitch`) to process raw roll and pitch angles. These filters allow the robot to respond to actual body tilt with **more gradual transitions in response to changes in body tilt**, instead of reacting abruptly to minor bumps or rapid direction changes. The filtered tilt angles are then used to modulate the vertical position (Z-height) of each leg tip, helping the robot maintain a level support base during aggressive motion on uneven terrain. This significantly enhances gait stability and reduces the risk of tipping.

#### 4.2.3. Motor Commands

Once the desired foot positions are calculated—based on gait timing, joystick input, and tilt compensation—these are translated into joint angles using inverse kinematics. However, even small variations in desired tip positions between loops can cause noticeable and abrupt changes in the resulting joint motor angles. If sent directly to the servos, these would result in jerky or twitchy leg movements.

To prevent this, the joint angles are passed through a second set of Kalman filters (e.g., `kf_L1[0]`, `kf_L1[1]`, etc.) for each joint on every leg. These filters produce smoother transitions between joint states when changing directions and speeds, leading to **more natural, stable, and gradual motion**.

### 4.3 Communication Protocol

In this current version of the controller, communication is required mainly for data acquisition from the gyro unit and serial communication with a computer for debugging.

#### 4.3.1 I2C Protocol via Wire.h

The **I2C protocol** enables efficient communication between the Arduino (master) and digital sensors (slaves). In this system, I²C is implemented using the Arduino Mega's SDA (data) and SCL (clock) lines through the `Wire.h` library. The primary I²C device in use is the MPU6050 IMU gyro unit, which provides real-time roll and pitch data for dynamic balance control.

Specifically, the `Wire.h` library abstracts the complexities of I²C, allowing the program to [1]:

- Initialize the bus with `Wire.begin()`
- Request and receive sensor data using `Wire.write()` and `Wire.read()`
- Communicate with the MPU6050 via specific I2C addresses and registers

Such communication between the gyro unit and the microcontroller is critical for maintaining the robot's awareness and subsequent real-time adjustments of its tilt and orientation.

#### 4.3.2 Hardware Serial Communication

In parallel, the system uses **hardware serial communication** (via `Serial`) to interface with a Bluetooth module or computer terminal. This channel is used to [2]:

- Receive joystick data from a Bluetooth controller through the `parseData()` function
- Print debug information (e.g., filtered pitch/roll values)
- Tune parameters or monitor system behavior during development

Hardware serial is full-duplex, byte-stream-based, and typically runs over USB or UART pins. It complements the I²C interface by handling high-level human-computer interaction, while I2C handles low-level sensor feedback.

## 5. Conclusion & Future Work

A potential upgrade from the existing design is to first miniaturize the robot for better portability, agility, and for it to have the ability to traverse through tight corners. With a mounted camera unit, the robot will be able to aid search and rescue, or geological surveys by traversing where human operators are unable or have to risk great dangers to reach.

## 5. References

1.Arduino, “Wire,” Arduino Documentation, [Online]. Available: https://docs.arduino.cc/language-reference/en/functions/communication/wire/. [Accessed: Apr. 7, 2025].
2.Arduino, “Serial,” *Arduino Documentation*, [Online]. Available: https://docs.arduino.cc/language-reference/en/functions/communication/serial/. [Accessed: Apr. 7, 2025].

## 7. Appendices

### Appendix A - Demonstration Video

The demonstration video is uploaded to MyMedia: https://play.library.utoronto.ca/watch/572a60d962e60df8fe7632fa0d0cc9d6

### Appendix B - Bill of Material (BOM)

Table B1. Bill of Material

| Component ID # | Name | Model/Type/Description | QTY. |
|---|---|---|---|
| 1 | Microcontroller (MCU) | Arduino Mega 2560 | 1 |
| 2 | Motor | RDS3225 | 18 |
| 3 | Gyroscope | MPU6050 | 1 |
| 4 | Bluetooth Module | HC-05 | 2 |
| 5 | DC-DC Buck Converter | DROK DC Buck Converter, 5.3V-32V to 1.2V-32V 12A Adjustable Power Supply | 1 |
| 6 | Lithium battery | SIGP 3S 11.1V 25C 2250mAh Lipo Battery | 1 |
| 7 | AA battery | Energizer MAX AA Batteries | 4 |
| 8 | AA battery holder | QTEATAK 8 Pack AA Battery Holder | 1 |
| * Only key components in the design are listed in the BOM. Components such as electrical wires are not listed. | | | |

## Appendix C - Power Supply Input Capacitor

Based on the formula C = I × Δt / ΔV, tests have shown that the current can reach 11A and the voltage drop lasts for approximately 10ms (measured via the oscilloscope) when there’s no input capacitor installed. To ensure the voltage drop does not exceed 0.25V, it was estimated that a 0.6F capacitor is required for effective compensation.

## Appendix D - Detailed PCB Schematic

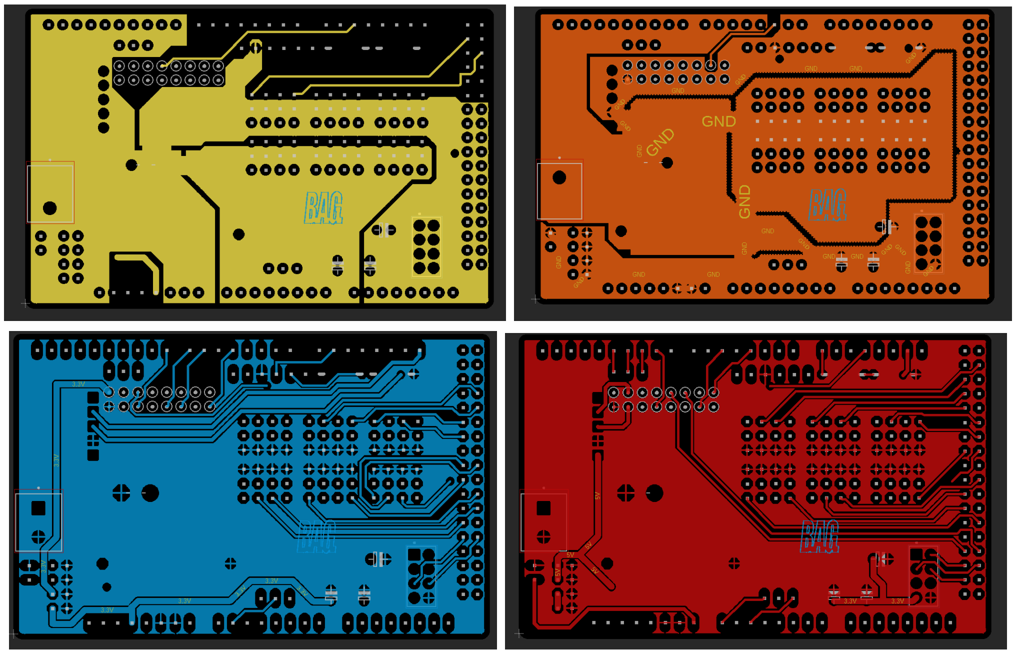


Figure D1. Layers after the copper pour

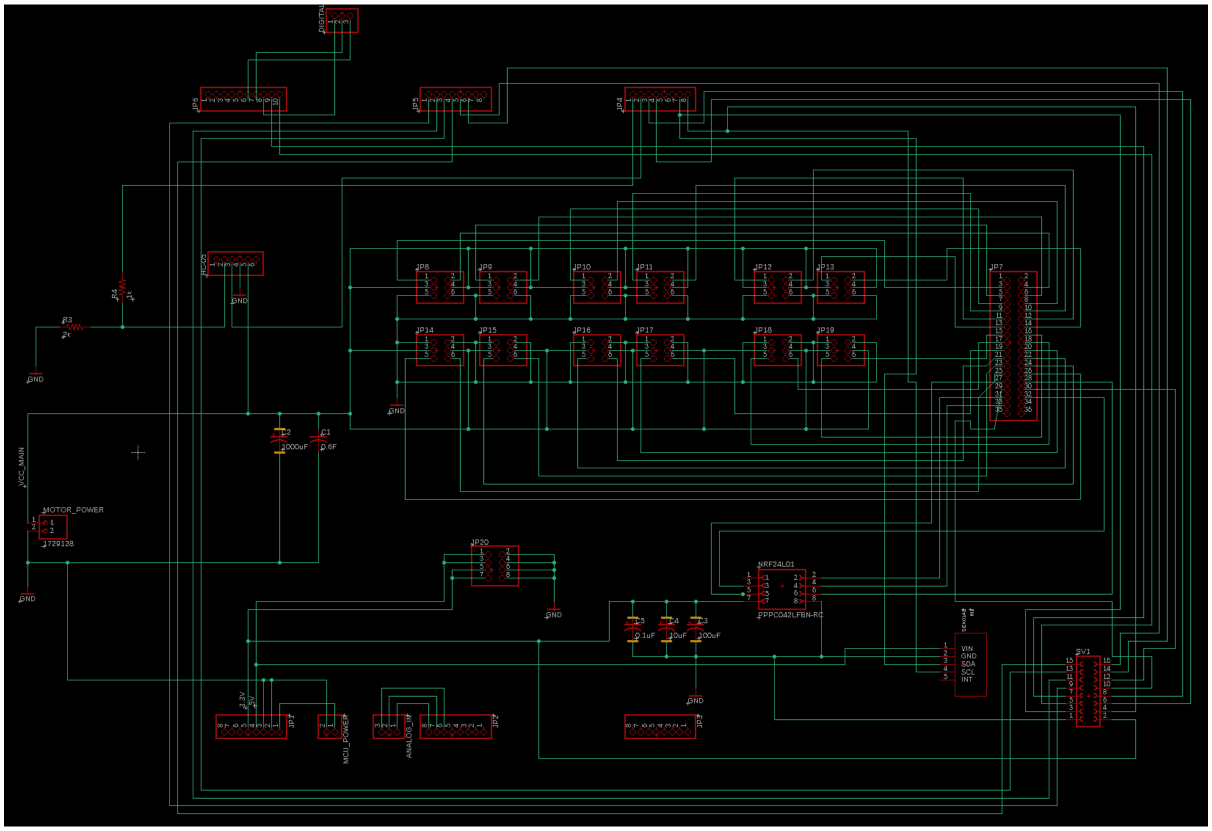


Figure D2. Complete PCB schematic

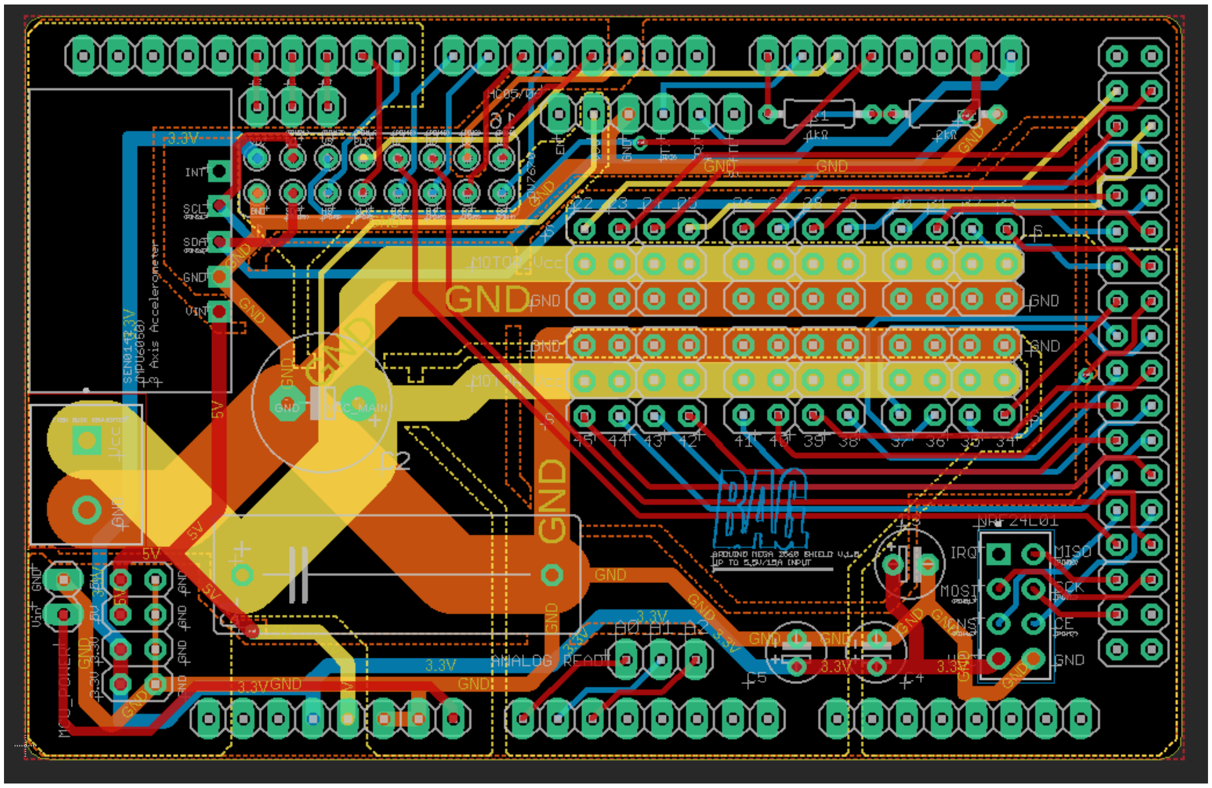


Figure D3. PCB layer diagram

## Appendix E - Static Load Analysis

Given that the robot employs a tripod gait, during each phase of the gait cycle, three legs are in contact with the ground supporting the robot's total weight (F, including the external loads and the weight of the robot itself), meaning each leg carries roughly a maximum contact force with the ground F/3 (Figure M1).

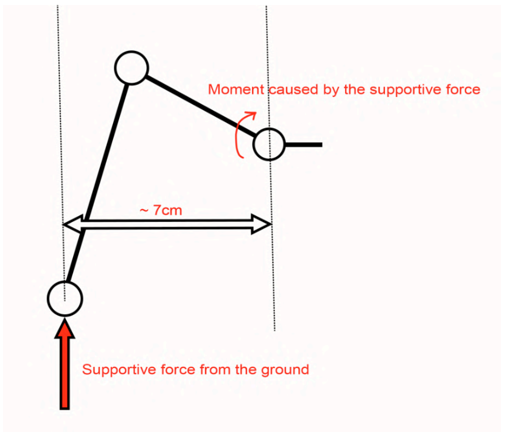

Moment caused by the supportive force
~ 7cm
Supportive force from the ground

Figure E1. Leg static load analysis model

The moment around joint 1 is the largest among all joints on a leg, calculated as (7/3)F. To ensure the robot can withstand this moment (i.e., to prevent the joint from bending), an equal and opposite torque must be applied at that joint. Given that the motor's maximum torque is 25 kg·cm—which converts to 245.25 N·cm—balancing the moment by setting (7/3)F equal to 245.25 N·cm implies that the maximum possible load is approximately **104.967 N** (~10.7 kg) (this value includes the robot's own weight of about 2.7 kg).

## Appendix F - Gait Control Logic

In the gait control algorithm, the underlying principle is to establish a sequence of trajectory coordinates within a specified coordinate system, and subsequently direct the robot to traverse these predetermined paths. In this context, the term "frame" is employed interchangeably with "coordinate system" or "coordinate axis."

Before discussing gait control, several important frames are redefined. First, the base frame {B} is established with its origin at the center of the base link. Additionally, each leg is associated with a relative frame $\{L_i\}$ (where i indicates the leg number) used to describe the leg's position relative to the base link; the origin of each $\{L_i\}$ is located at the connection point between the leg and the base link. Finally, the end effector frame $\{E_i\}$ is defined for each leg, with its origin positioned at the center of the End Effector Circle (EEC). The EEC represents the range within which the leg's contact point or tip can move, as indicated by the green circle in Figure F1.

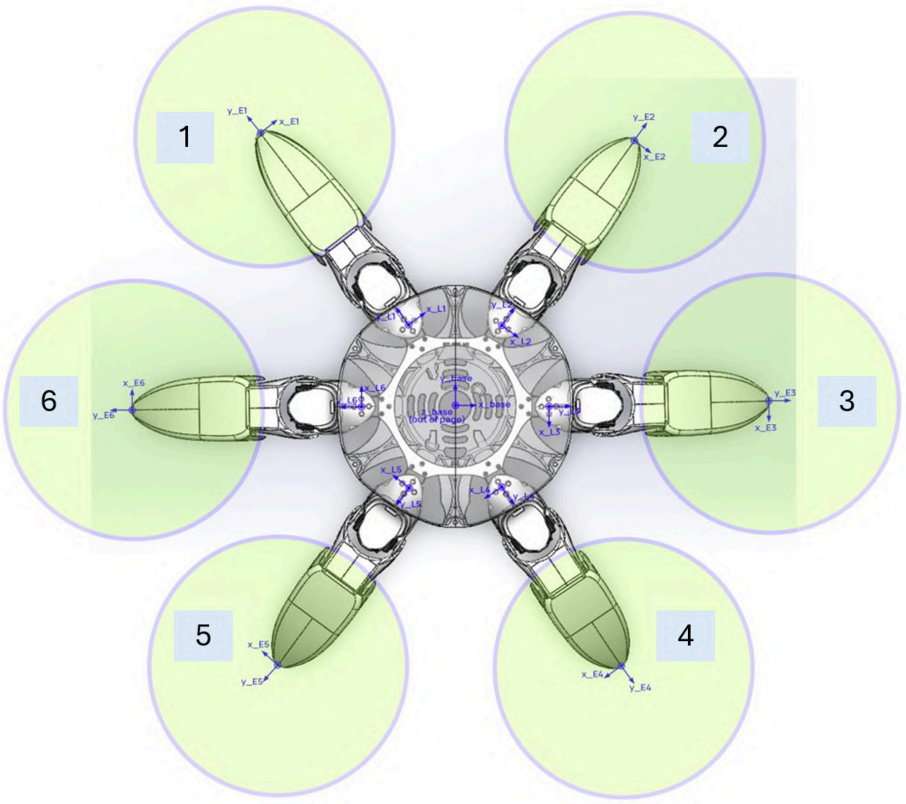


Figure F1. Frames definition, all z-axes are pointing out of the page; The green circles indicate the EEC for each leg. In this figure, the offset between {E} origin and the EEC center is 0

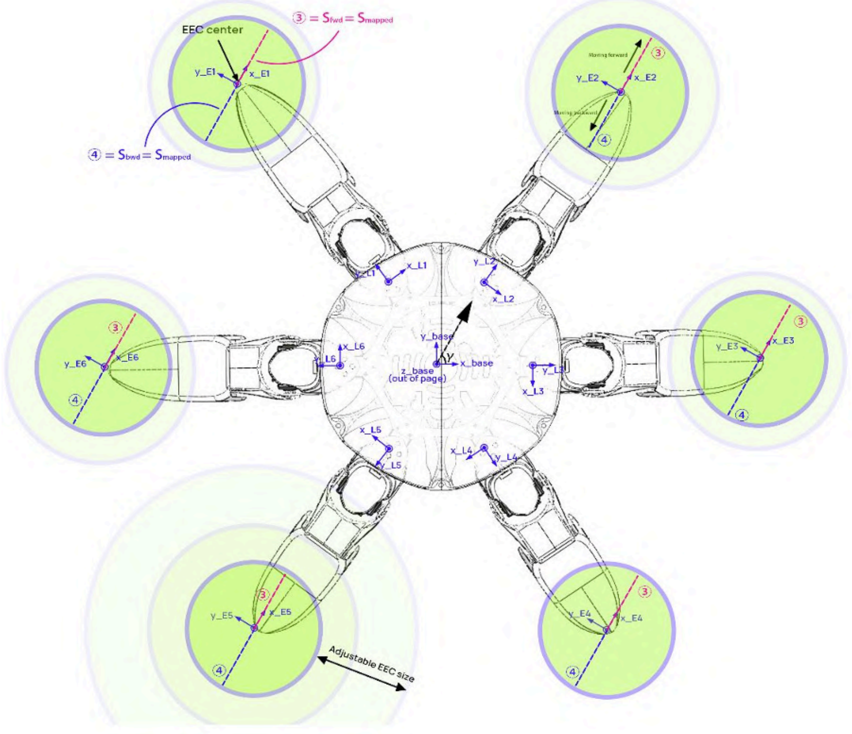


Figure F2. In planar translational walking, the orientation of the x-axis in every {E} should “follow” the same direction as the input vector in the view of {B}. The offset between {E} origin and the EEC center is 0.

The robot's planar walking logic is built on control via a remote controller paired with a Bluetooth module. The remote signal generates an input vector representing the desired movement direction, the input vector is then transformed into the base frame {B}. The vector's magnitude ranges from 0 to 10—indicating the distance each leg should move, as shown by the black arrow in Figure F2 (with 0 as the minimum and 10 as the maximum)—and its direction, defined by the angle it forms with {B}, sets the robot's overall movement direction. To ensure coordinated motion, the x-axis of every end effector frame {$E_i$} dynamically aligns with this input vector, while the {B} frame remains fixed.

Building upon the established coordinate frames and the initial determination of movement direction and distance, the next step is to define the parameters that govern the execution of a full step. To achieve smooth and adaptable motion, three key parameters are introduced. First, $S_{fwd}$ represents the step size along the positive x-axis of the corresponding {E} frame, representing “moving forward”, under the given input vector, while $S_{bwd}$ is the step size in the opposite direction. Smax is the maximum step size possible for both $S_{fwd}$ and $S_{bwd}$ regardless of the input. Although the input signal has a magnitude ranging from 0 to 10, it is proportionally mapped to the range [0, $S_{max}$] to yield the corresponding step size, referred to as $S_{mapped}$. Note that $S_{fwd}$ and $S_{bwd}$ are not always equal; their equality depends on the input vector. In cases of abrupt commands, such as sudden stops, different step sizes are adopted to allow for more adjustable landing points and smoother motion. By dynamically adjusting $S_{fwd}$ and $S_{bwd}$ and incorporating gyroscope data, the robot can adapt to various terrains.

1. Same step size

$$S_{fwd} = S_{bwd} = S_{mapped}$$

2. Different step size

$$S_{fwd} = C_3 \times S_{bwd} = C_4 \times S_{mapped}$$

Both case 1 and 2 are constrained by

$$D_{EEC} = S_{fwd} + S_{bwd}$$

When the robot walks, the trajectory of each leg's end effector on the xz-plane of its corresponding {E} frame is segmented into six distinct phases, as illustrated in Figure F3. The trajectory itself is composed of discrete sampling points. In an ideal scenario, the time interval between each consecutive point would be fixed; however, in practice, this interval is influenced by the microcontroller's processing speed. To ensure smoother transitions between segments, a quadratic easing function is applied, which increases the density of sampling points at the beginning and end of each motion segment. This method not only refines the start-stop effects but also contributes to a more natural and fluid motion pattern.

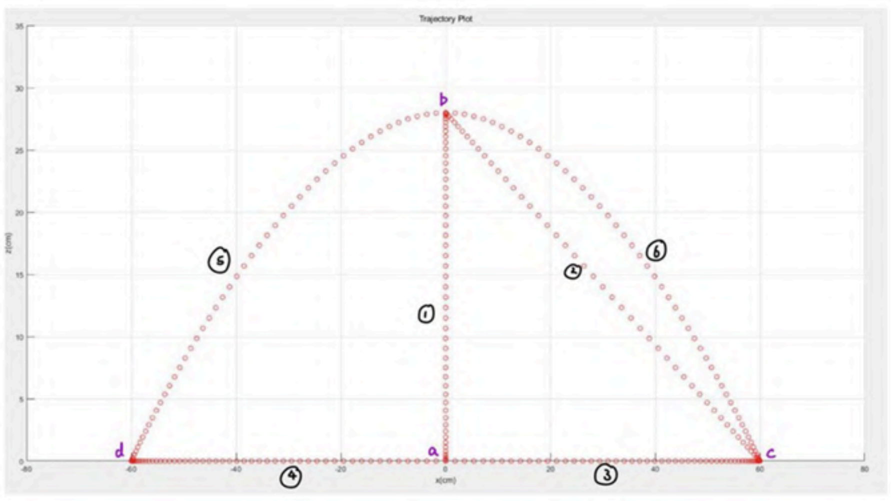


Figure F3. The motion trajectory within the {E}. Paths ③ and ④ correspond to the $S_{fwd}$ and $S_{bwd}$ in Figure 7. (③+④= $S_{fwd}$ + $S_{bwd}$ = the diameter of the EEC).

Previously, the method for determining step size, movement direction, and individual leg motion was described. Next, the algorithm for coordinating all six legs is introduced. Initially, assume that the end effector of every leg is located at point "a" in its corresponding relative frame {E} (see Figure A3), which serves as the neutral or original position. In practice, when the joystick input falls below a predefined threshold, the computed step sizes for all segments reduce to zero, causing the EEC's diameter to collapse and all end effectors to return to point "a," thereby halting the robot's movement. The six legs are then divided into two groups: Group A consists of legs 1, 3, and 5, while Group B comprises legs 2, 4, and 6 (as shown in Figure A1). Similar to human walking—where one group of legs lifts as the other makes ground contact—during one alternating step cycle, Group A moves from point ***a*** to point ***b*** along path ① (see Figure 8), then from point b to point c along path ②. Next, while Group A moves from point c to point d along paths ③ and ④, Group B simultaneously follows paths ① and ②. The duration for Group A's movement along paths ③ and ④ must match the time for Group B's movement along paths ① and

②, ensuring that at least one group of legs is always in contact with the ground. After that, Group A proceeds along paths ⑤ and ⑥ while Group B follows paths ③ and ④. Thereafter, Group A again moves along paths ③ and ④ and Group B along paths ⑤ and ⑥. This cycle repeats continuously, entering the "Loop Phase" (see Table A1).

Table A1. Gait Sequence Table (“-” means on the ground)

| | Start-up Phase | | | Loop Phase | |
|---|---|---|---|---|---|
| Group A | ① | ② | ③④ | ⑤⑥ | ③④ |
| Group B | - | - | ①② | ③④ | ⑤⑥ |

In addition, by dynamically adjusting $S_{fwd}$ and $S_{bwd}$, each step's landing point can be precisely controlled. Coupled with gyroscope feedback, this enables the robot to adapt effectively to different terrains. During motion, the magnitudes of paths ① through ⑥ can be adjusted. In Figure F3, when paths ③ and ④ are equal (i.e., $S_{fwd} = S_{bwd}$), the movement remains symmetrical; however, if they are unequal (i.e., $S_{fwd} \neq S_{bwd}$), the origin of the {E} frame will shift radially relative to the center of the End Effector Circle (EEC) (see Figure F4).

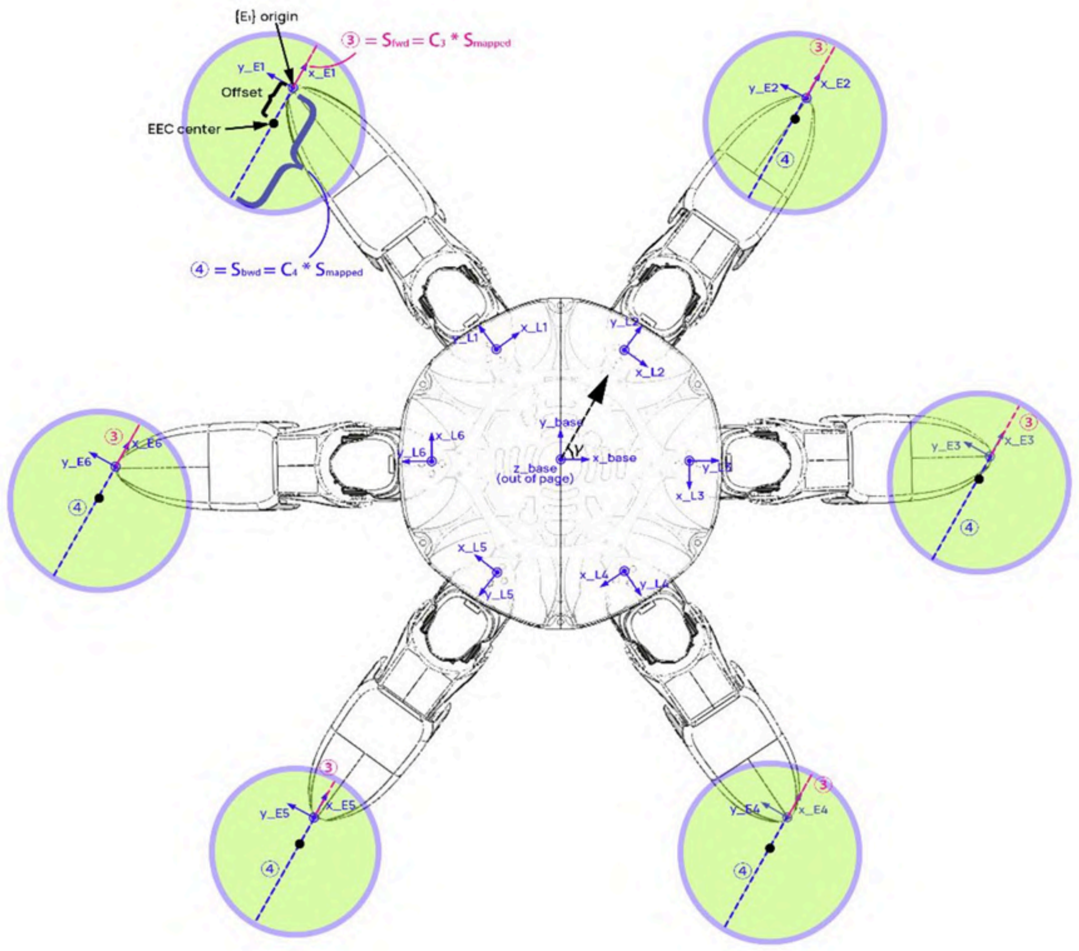


Figure F4. Planar translational walking when ③ and ④ are different, a non-zero offset between the {E} origin and EEC center is introduced.

When paths ③ and ④ are equal (i.e., $S_{fwd} = S_{bwd}$, as shown in Figure A2) under a linearly increasing input u(t), the trajectory in {E} is plotted in MATLAB (Figure F5.a, where red represents Group A and blue represents Group B). Figure F5.b shows that as u(t) increases linearly, Smapped (x(t)) and the length of path ① (z(t)) also grow linearly; Figures F5.c and F5.d display the x and z coordinates of each group of legs in {E} over time, demonstrating the gait's suitability during walking.

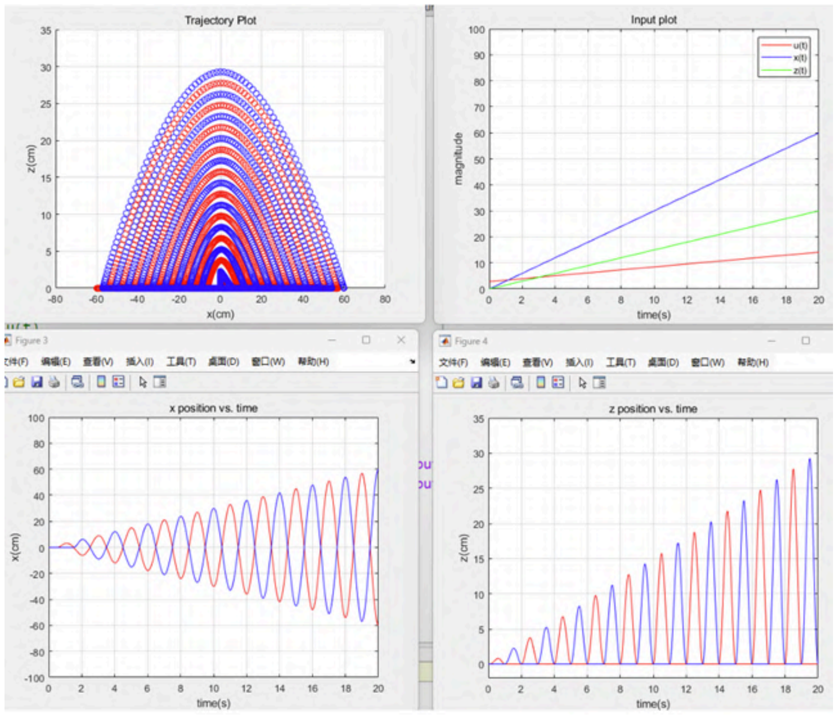


Figure F5. a (top left): trajectories for each group of legs (red-Group A, blue-Group B) in {E}; b(top right): magnitude of input u(t), $x(t) = S_{fwd} = S_{bwd}$, z(t) = length of path ①, with respective to time t; c(bottom left): x position of the end effector in {E}. d (bottom right): z position of the end effector in {E}.

When paths ③ and ④ are unequal, and paths ⑤ and ⑥ are also unequal, with a nonlinear input u(t) (see Figure F6.b), paths ③, ④, and ① exhibit nonlinear trends of increase and decrease. In Figures F6c and F6.d, the x and z coordinates of each group of legs over time are plotted (red for Group A and blue for Group B).

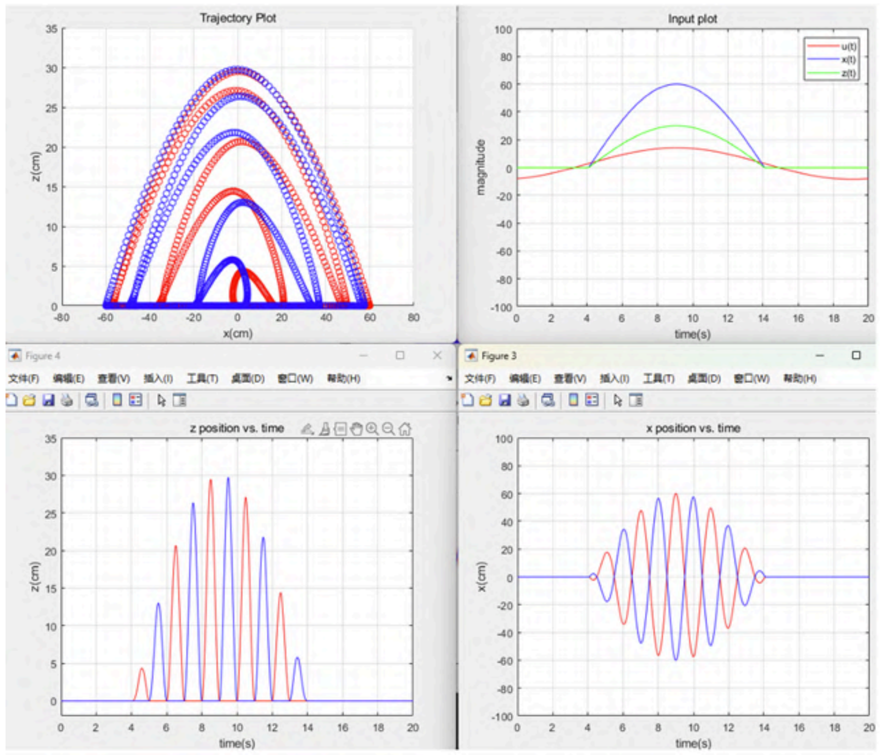


Figure F6.
a (top left): Trajectories for each group of legs (red – Group A, blue – Group B) in {E}.
b (top right): Magnitude of input, $x(t) = (1/C_3) \cdot Sfwd = (1/C_4) \cdot Sbwd$, z(t) = ① with respect to time.
c (bottom left): x position of the end effector in {E}.
d (bottom right): z position of the end effector in {E}.

Furthermore, each path segment (①–⑥) is traversed over a fixed time interval T, ensuring that the timing remains constant regardless of the movement. However, the step size is determined by the magnitude of the input vector. Consequently, even with a fixed duration T, an increase in step size results in a higher end effector speed along each segment (since v = distance/T).

It is worth noting that all trajectory coordinates were initially computed in the {E} frame to facilitate intuitive trajectory planning. Therefore, for motor control, these coordinates must be transformed into the {L} frame, which aligns with the leg's physical configuration. This conversion is critical because the inverse kinematics equations—responsible for converting desired end-effector positions into corresponding joint angles—are formulated in the {L} frame. Such a transformation ensures that the computed trajectories accurately correspond to the robot's mechanical structure and joint configurations, thereby enabling precise and coordinated leg motions.

Finally, due to the High-frequency noise during Bluetooth communication impacted the received data. To address this issue, seven Kalman filters were implemented—one in the Bluetooth reception code and one for each of the six legs in the control code before sending motor commands. The parameters of each filter were meticulously tuned through repeated trials to guarantee robust performance, effectively smoothing the control signals and improving overall gait stability.