UNIVERSITY OF SOUTHERN DENMARK

ROBOT SYSTEMS ENGINEERING

Building a Biped Robot

Bachelor Thesis

DUE DATE: 2. JUNE 2025

SUPERVISOR: ALJAZ KRAMBERGER (ALK@MMMI.SDU.DK)

CO-SUPERVISOR: ROBERTO DE NÓBREGA (RNO@MMMI.SDU.DK)

Submitted By:

ANAS BUTT HUSSAIN

15-12-2003

anhus22@student.sdu.dk

Contents

1 Project Description **1**
- 1.1 Introduction . . . 1
- 1.2 Project Objectives . . . 1
- 1.3 Methodology . . . 1

2 Mechanical Design **2**
- 2.1 Leg Assembly . . . 2
- 2.2 Chassis Design . . . 2
- 2.3 Neck Assembly . . . 3
- 2.4 Head Structure . . . 3
- 2.5 Kinematic Structure . . . 4

3 Electronics & Hardware Integration **5**
- 3.1 Chassis Electronics . . . 5
- 3.2 Head Electronics . . . 8

4 Assembly and Build Process **8**

5 Serial Hardware Communication and Uploading **11**
- 5.1 Serial Communication Setup . . . 11
- 5.2 DTR Reset Limitation on Pro Mini . . . 11
- 5.3 Simplified Upload via USB (Arduino Nano) . . . 11

6 Robot Movement **12**
- 6.1 Servo Selection and Behavior . . . 12
- 6.2 Servo Initialization and Direction Mapping . . . 13
- 6.3 Programming the Walking Sequence . . . 15
- 6.4 Challenges and Observations During Gait Development . . . 15
- 6.5 Hand Detection and Motion Triggering . . . 16

7 Conclusion **20**

8 Appendix **21**

9 References **22**

1 Project Description

1.1 Introduction

Walking robots have been a key focus of research in robotics and control systems. They come in different shapes, functions, and movement types. Among them, bipedal robots are particularly challenging because they are naturally unstable, requiring strong control systems to stay balanced and move smoothly.
This project aims to develop a bipedal locomotion system, consisting of two lower limbs connected through a central member, designed to replicate human-like walking. The primary goal is to create a functional bipedal robot capable of balancing itself while walking. Since balancing is a critical aspect of bipedal locomotion, the system must ensure dynamic stability through efficient control algorithms and actuation mechanisms. In a simplified walking robot, the structure above the hip primarily serves as payload, while locomotion is driven by the activation of joints below the hip.

1.2 Project Objectives

The project is structured into three main work areas:

- **Mechanical Design & Prototyping** – Designing and fabricating a stable and lightweight bipedal frame, ensuring that the structure supports smooth and efficient movement. Most components will be 3D printed, allowing for rapid prototyping, customizability, and ease of iteration. The mechanical framework will integrate actuators for precise joint movement and maintain an optimal weight distribution for stability.
- **Electronics Integration** – Implementing an Arduino-based servo control system to manage joint actuation efficiently. Additionally, a Raspberry Pi will function as the robot's primary processing unit, handling complex computations required for movement control and sensor data processing.
- **Software Development & Testing** – Developing the control algorithms necessary for gait generation, balance correction, and interactive functionalities. The robot will feature a hand-tracking mechanism, enabling it to detect a person's hand, follow it with its "head," and move closer if the hand is within a predefined range. This interactive capability will be achieved through computer vision techniques integrated with the Raspberry Pi.

Given the complexity of bipedal locomotion, the project will emphasize real-time feedback control to ensure stable walking. The integration of mechanical, electronic, and software components will allow the robot to perform dynamic balance corrections and interact with its environment effectively.

1.3 Methodology

The development process followed an iterative and exploratory approach. Rather than relying on a fixed sequence of research and execution, design choices evolved through hands-on experimentation, testing, and adaptation.
The mechanical structure, electronic components, and control logic were based on the open-source Modular Biped project developed by Dan Makes Things [1], complemented by a detailed instructional YouTube series [2]. These resources provided the foundational models and guidance for the assembly and setup process.
The 3D-printed components were reused from the original design, and the focus was on assembling them correctly and integrating them with the electronics and control system. No redesign of mechanical parts was performed; instead, the goal was to get the existing design working reliably within the project's hardware constraints.
Servo motor control and sensor input were tested alongside software development, allowing real-time feedback and iterative tuning. Testing focused on practical assessments such as gait movement, balance under motion, and interactive responsiveness through computer vision (e.g., hand-tracking). Observations from each test phase informed adjustments to code, wiring, and physical positioning to improve walking behavior and stability.

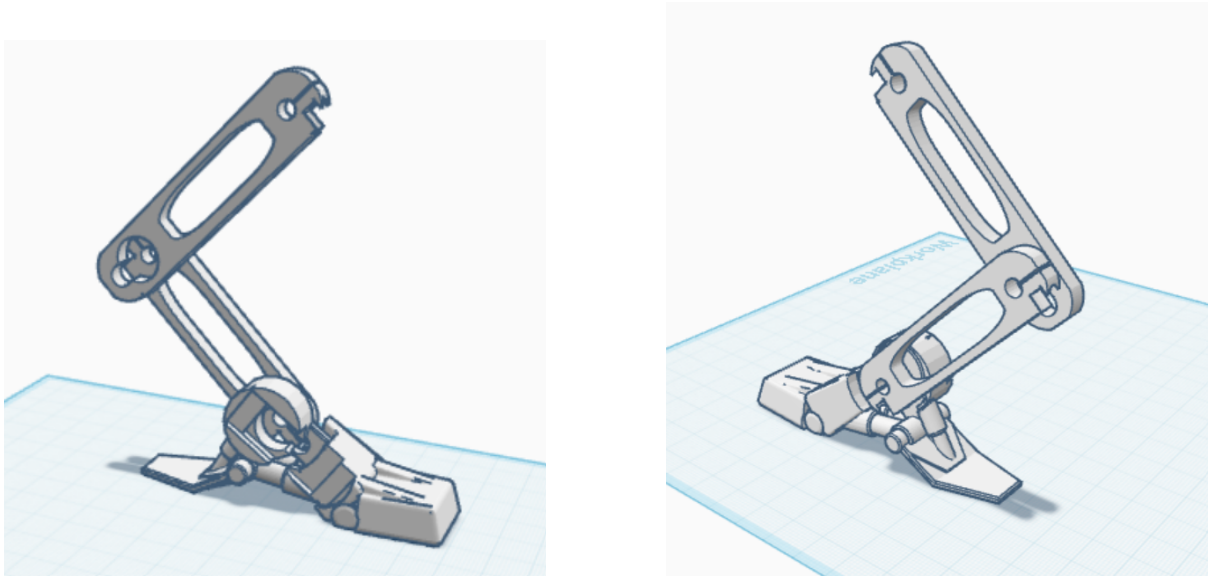
2 Mechanical Design

The mechanical design of the robot centers around a modular structure composed entirely of 3D-printed PLA parts with 15% infill. The design consists of 18 distinct components that together form a multi-articulated leg system, a central chassis, a flexible neck mechanism, and a compact head housing the core computing hardware.

2.1 Leg Assembly

Starting from the bottom, each leg begins with a foot component. At the "ankle" joint, the structure includes pre-formed cavities shaped to accommodate micro servo motors. These are inserted from the inner side of the leg for ease of installation. On the outer side, the servo's metal shaft inserts into a matching hole at the base of the lower leg. A small screw hole enables secure attachment of the shaft, anchoring the servo directly to the PLA structure.

The lower leg connects to the upper leg via another set of micro servos, once again seated into precisely modeled cavities. The servo shafts interface directly with the upper leg component and are secured with screws, providing both articulation and structural rigidity. No additional brackets or fasteners are needed, as all servos are mounted directly into the PLA through screw fittings.



(a) Inner side

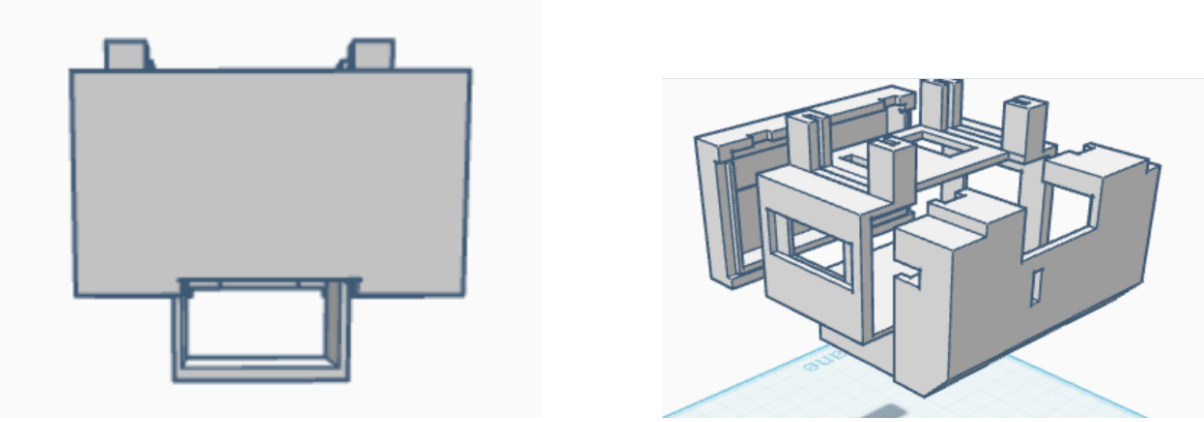
(b) Outer side

Figure 1: Views of the leg design showing both inner and outer sides

2.2 Chassis Design

The central chassis serves as the main structural core and houses the electronic subsystems (detailed in Section 3). It includes both front and rear panels that are push-fit for easy assembly and maintenance. The rear panel features a small cutout for USB access.

The chassis is designed with integrated slots on its sidewalls and top surface to accommodate servo motors responsible for neck articulation. The bottom section of the chassis includes a dedicated compartment for a LiPo battery; however, this was ultimately not used in the final implementation. The initial design included a selector switch accessible by removing the rear panel, which would have allowed switching between battery and USB-C power. Due to time constraints and a shift in project priorities, this functionality was left out.



(a) Front of chassis

(b) Back of chassis

Figure 2: Front and rear views of the chassis showing panel design, access points, and servo motor slots

2.3 Neck Assembly

Above the chassis is a neck made of five 3D-printed parts. One part holds a micro servo that lets the head rotate side to side. The other four parts work together to tilt the head up and down, controlled by servos mounted on top of the chassis. This setup gives the head two degrees of freedom while keeping the design compact.

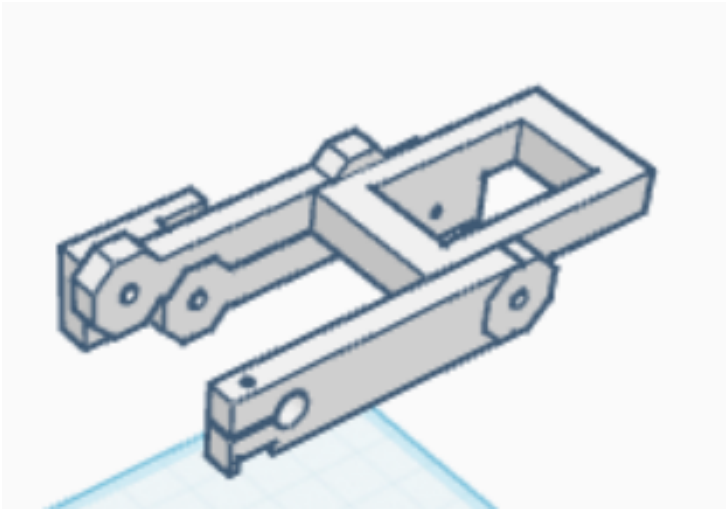


Figure 3: 3D view of the neck mechanism

2.4 Head Structure

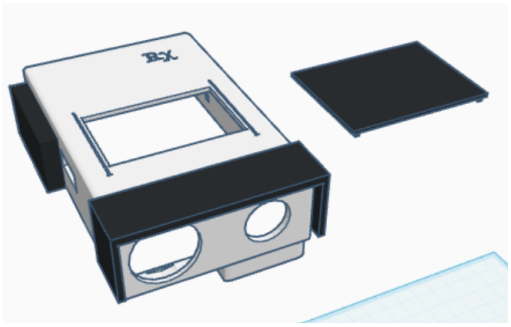
The head is assembled from four PLA components: a bottom shell, a top shell, a removable lid, and a visor. The bottom part connects to the neck's rotational servo and includes vent slots to facilitate airflow. It also has a small gap through which power and serial wires are routed—further details are provided in Section 3. Internally, the bottom shell houses a Raspberry Pi board and a logic level converter.

The top of the head features a removable push-fit lid that enhances ventilation and allows easy access to the Raspberry Pi. A purely aesthetic visor component is also mounted on the top shell.

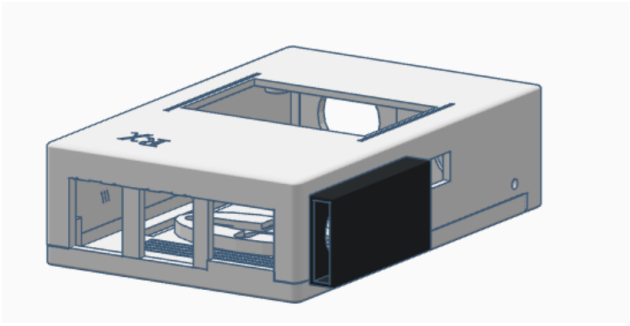
At the front of the top shell are two circular cutouts intended for an LED and a camera module. The LED was not included in the final build, and the installed camera differs from the one originally intended, resulting in a looser fit. Adhesive tape was used to hold the camera in place. These modifications were necessary due to time constraints and the emphasis on achieving basic locomotion. The top shell also includes rear-facing cutouts for Ethernet and USB ports, allowing access to the Raspberry Pi's I/O interfaces.

Overall, the mechanical design is made to be modular, easy to access, and compatible with standard electronic

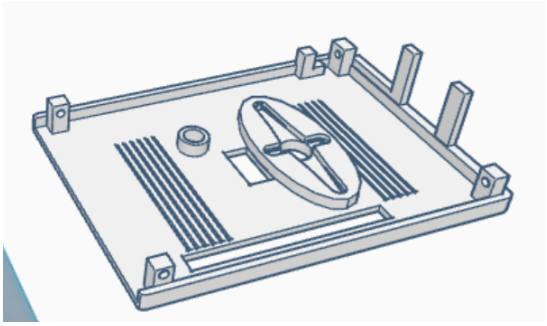
parts, with room for future improvements.



(a) Top shell & lid



(b) Rear view of top shell



(c) bottom shell

Figure 4: Top and bottom parts of the head structure, with multiple angles showing component details and internal mounting space.

All 3D model images in this section are adapted from the original robot model by Dan Nicholson [3], created in Tinkercad. The model was remixed to isolate and display individual components more clearly for documentation purposes.

2.5 Kinematic Structure

The robot's motion is driven by a total of 9 micro servo motors, each providing one rotational degree of freedom. These are distributed across the head, neck, and both legs, forming a simple yet functional kinematic chain suitable for bipedal locomotion and basic interaction.

Figure 5 illustrates the kinematic structure of the robot and the placement of its servo motors. At the top is the head servo, which enables yaw rotation. The head is also connected to two neck servos that work together to control pitch movement through a linked mechanism. These three servos together offer the robot's head two degrees of freedom for expressive movement.

Each leg is actuated by three servo motors: a hip servo for leg swing (pitch), a knee servo for bending, and a foot servo for adjusting foot angle during gait. The left and right legs are symmetric in their configuration. This structure allows for basic walking behavior and balance adjustments.

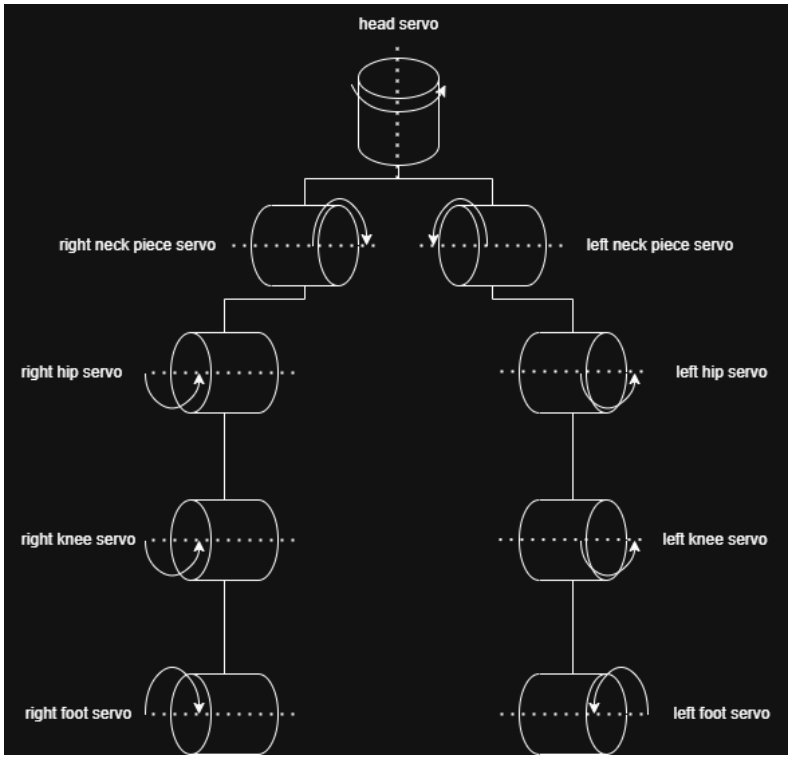


Figure 5: Kinematic layout of the robot's servo configuration. The design includes 9 micro servos: one for head rotation, two for neck articulation, and three per leg (hip, knee, and foot). Each joint is actuated by a single servo motor, forming a modular and easily programmable motion structure.

3 Electronics & Hardware Integration

The robot's electronic system is divided into two main sections: the chassis and the head. Together, they support power regulation, communication, actuation, and hand detection. The design emphasizes simplicity and modularity, focusing on essential locomotion features.

3.1 Chassis Electronics

Originally, the electronics were centered around a custom PCB designed for an Arduino Pro Mini. This board included the following key components:

- Arduino Pro Mini (ATmega328, 16MHz, 5V version)
- Two MP2307 DC Buck Converter Step-Down Modules
- A 5A fuse
- 1-channel relay module (5V compatible)
- Pins for power and UART communication with the Raspberry Pi
- Headers for power input from the USB-C PD Trigger Module
- Pins for powering and controlling nine MG90S micro servos

The system is powered by a USB-C PD Trigger Module capable of negotiating various output voltages as specified in its datasheet [4], including 5V, 9V, 12V, 15V, and 20V. However, in practice, the 12V option was not functional—the module switched directly from 9V to 15V. Since the system required a minimum of 9V to operate reliably, 15V was selected as the default supply voltage.

This 15V output feeds two MP2307 DC-DC buck converters [5]. These modules accept a wide input voltage range from 4.75V to 23V and provide an adjustable output between 0.925V and 20V. Output voltage was tuned using an integrated potentiometer included on the module. One converter was configured to deliver 5V for the Arduino and servo motors. This output is ideal for the 5V version of the Arduino Pro Mini [6] and falls well within the

recommended operating range of 4.8V to 6V for the MG90S micro servos [7]. The second converter was set to 5.1V to supply the Raspberry Pi 5, which is more power-demanding and sensitive to voltage drops, especially when peripherals such as a camera are connected [8]. Each converter is rated to supply a continuous output current of up to 3A [5].

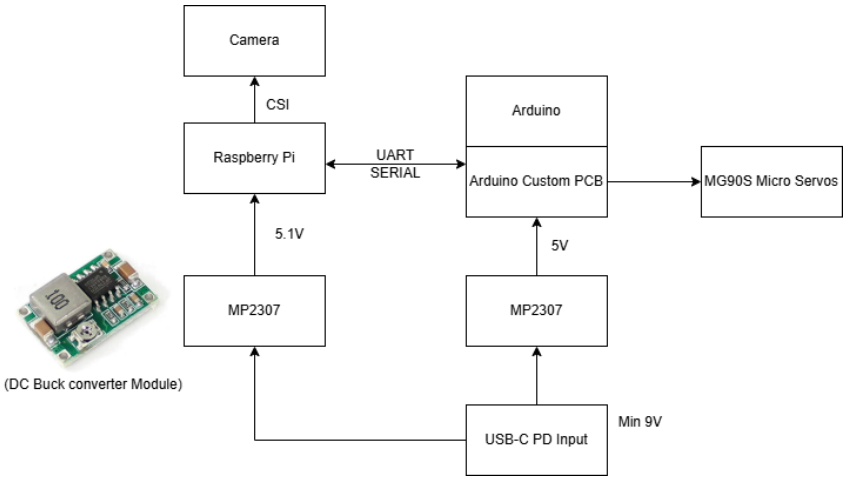
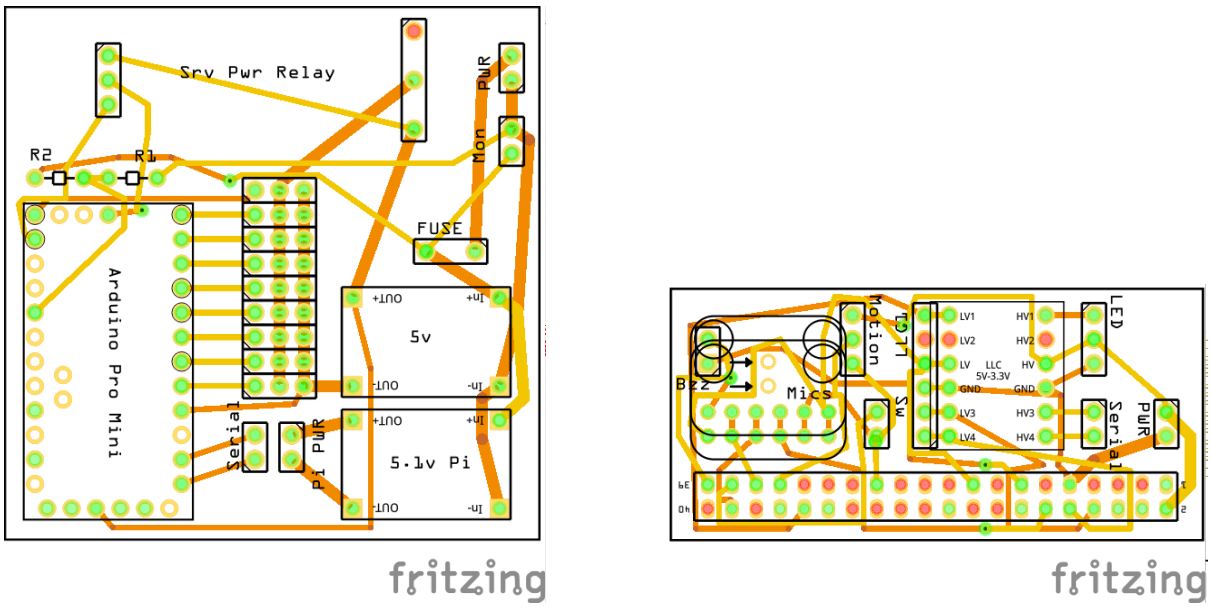


Figure 6: Block diagram of the system's power and communication flow.

A relay module (KY-019RM) was initially intended to work with a microwave radar sensor (RCWL-0516) to automatically disconnect power from the servos under idle conditions, serving as a simple power control feature. However, due to time constraints, this functionality was not implemented, and the relay was not mounted, although its footprint and traces remain present on the PCB.

Similarly, unpopulated resistor pads (R1 and R2) were originally included for a voltage divider circuit designed to monitor battery voltage using one of the Arduino's analog input pins. These components were ultimately left out of the final build.



(a) Chassis PCB Schematic (Original)

(b) Omitted Head PCB Schematic

Figure 7: Initial PCB layouts adapted from the Modular Biped project [9]. Left: the chassis-mounted PCB featuring unused battery monitoring and servo power control. Right: a head-mounted design that was ultimately omitted.

The head-mounted PCB shown in Figure 7b was designed to support additional sensory and feedback features. It included footprints for a microwave radar sensor (RCWL-0516), a logic level converter (BOB-12009), a buzzer

for simple audio output, two MEMS microphones (SPH0645) for sound input, and a small circular RGB LED ring (NeoPixel-compatible) for visual signaling. A tilt switch (SW-520D) was also included, intended to detect if the robot was turned upside down and trigger an immediate shutdown of the Raspberry Pi as a basic safety mechanism. Although this board was fully laid out as part of the original hardware design, it and its associated components were ultimately left out of the final build in order to focus on core locomotion and reduce integration complexity. The logic level converter was used for a short period during early development, until an alternative setup was adopted. This is described in Section 5.

Extension with Arduino Nano-Based PCB

To accelerate testing and implementation, a second PCB was introduced. It was designed to fit an Arduino Nano footprint and was provided by SDU Robotics. The board offered a simple, modular platform for extending servo control capabilities without modifying the original system.

The USB port on the Arduino Nano was used to establish a serial UART connection with the Raspberry Pi, enabling command transmission and programming via /dev/ttyUSB0. At the top-left corner of the PCB layout, a power indicator LED was included to visually confirm that the board is receiving voltage.

Power and ground were supplied to the board through a screw terminal block, which distributed voltage across the connected components. Six rows of 3-pin headers were placed on the left side of the board, each designed to power and control one micro servo. These headers provide signal (from the Arduino), 5V (from the screw terminal), and ground (common return).

Toward the bottom of the PCB are SPI breakout pins (MISO, MOSI, SCK, and SS) and a pair of UART pins (RX/TX), which were not used in this project but allow for future communication extensions. On the right side, a footprint for an 8-pole Phoenix-style connector was included for additional I/O expansion or grouped signal routing, though it was not used in the current configuration.

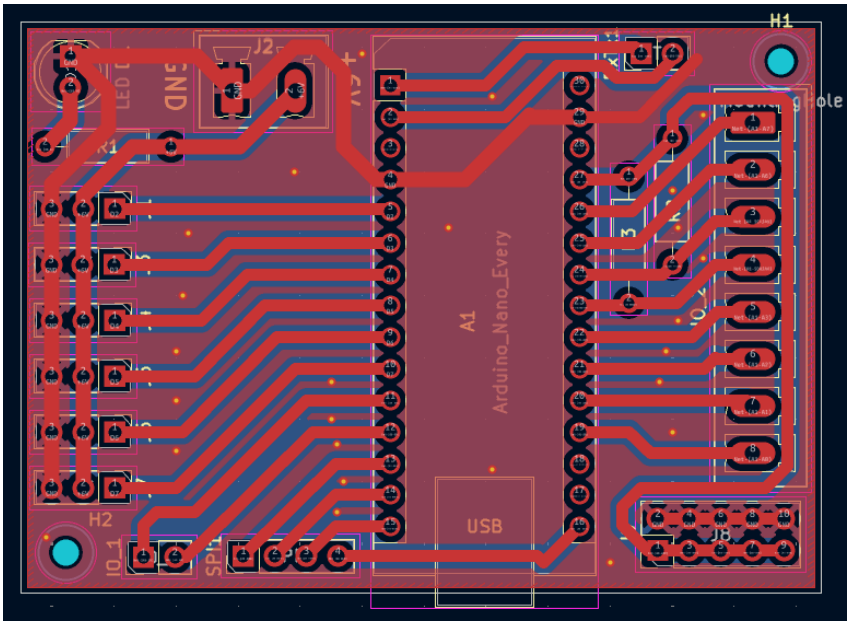


Figure 8: Layout of the Arduino Nano-based extension PCB.

Programming and Communication: Manual Reset Challenge

Originally, communication between the Arduino Pro Mini and Raspberry Pi was handled via UART over GPIO, using a logic level shifter to safely bridge the 5V/3.3V logic gap. However, uploading code through this method required a manual reset of the Arduino right before each upload. This is due to the absence of a DTR (Data Terminal Ready) or RTS line on the Pi's GPIO, which normally triggers the Arduino's bootloader via a brief reset pulse.

Without DTR, the Arduino IDE cannot automatically reset the Pro Mini to start the upload process. As a workaround, one would need to manually press the reset button at just the right moment. While some scripts

exist to emulate this behavior using a spare GPIO pin, they require setup and careful timing.

To simplify this and ensure a reliable programming process, the Arduino Nano was connected to the Raspberry Pi via USB, which appears as a /dev/ttyUSB0 device. This USB connection includes a DTR line, allowing automatic resets and seamless uploading through the Arduino IDE—eliminating the need for precise manual timing.

3.2 Head Electronics

The Raspberry Pi 5 is located in the robot's head and is powered directly by the 5.1V buck converter. Power is supplied via the GPIO header: pin 2 provides 5V from the buck converter, and pin 6 is connected to ground. This bypasses the USB-C power input and delivers regulated voltage directly to the board's power rail. The camera module is connected to the Pi via a 200 mm ribbon cable—the shortest length supported by the Raspberry Pi 5—and is mounted at the front of the head.

Although a head-mounted PCB was designed to support sensors (e.g., a microwave radar sensor, microphone, and LED), the board and its associated components were ultimately not used in the final implementation. (see Figure 7b)

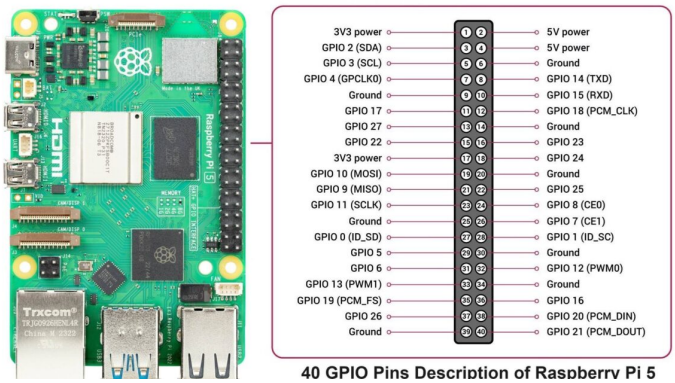


Figure 9: Pinout diagram of the Raspberry Pi 5 [10].

4 Assembly and Build Process

The robot was assembled in several stages, starting with the chassis and progressing through the legs, neck, head, and electronics. Each component was manually fitted, aligned, and tested to ensure correct mechanical function and structural stability.

Mechanical Assembly

Servo motors for the leg assembly were installed first. The chassis featured integrated sidewall slots designed to accommodate the micro servos, but the cable clearance was tight. A hand file was used to slightly widen the openings, ensuring safe insertion without pinching wires. Each servo was secured using two self-drilling screws, one on each side, and was manually tested to confirm a firm, wobble-free fit.

The upper leg components were mounted onto the metal shafts of the chassis servos and aligned perpendicular to the ground. They were fixed in place using screws through pre-modeled holes for secure coupling. The chassis was then tilted and rotated to confirm that the legs did not move under their own weight. If movement occurred, screws were tightened further.

Next, lower-leg servos were inserted into the cavities at the end of the upper legs and fastened similarly. Lower legs were attached to the output shafts, aligned vertically, and locked with screws. Finally, foot servos were mounted into the foot components, reinforced with superglue to reduce play, and attached to the lower legs using the same screw-tightening process.

Neck servos were inserted into the top of the chassis. As with the leg servos, the slots were carefully widened to ensure a snug fit. Although the neck servo mounting points were installed, the full five-part 3D-printed neck mechanism described in Section 2.5 was not included. Instead, a piece of rubber was taped across the servos to form a flat platform for mounting the head.

Head and Electronics Integration

The Raspberry Pi 5 was mounted on the upper shell of the head for easier port access. Power and ground wires were routed through openings in the lower shell and connected to the original PCB in the chassis, which supplied 5V and GND to the Pi via its GPIO header. The Raspberry Pi Camera Module 3 was attached to the front of the head using two self-drilling screws. Mini HDMI and USB-C port access was achieved by cutting into the top shell using a candle-heated hobby knife. Once wiring and port access were completed, both head shells were closed and taped to the rubber surface.

The original PCB was inserted into its designated slot in the chassis. A second PCB was introduced later in development and was initially mounted near the chassis base using tape, but this setup interfered with leg motion. It was eventually relocated to the top of the head, which raised the robot's center of mass and affected overall stability. Although this placement was less than ideal for balance, all essential servo connections—including those for the feet—were successfully routed to the head-mounted board, resolving earlier concerns about cable reach.

The USB cable connecting the Raspberry Pi to the Arduino Nano was initially long and heavy, which caused the head to tilt backward. The excess cable was coiled and tucked into the lower chassis (originally intended to house a LiPo battery), providing minor balance correction. A front-mounted steel connector plate (100×35mm) was added to the head as a counterweight to further improve balance.

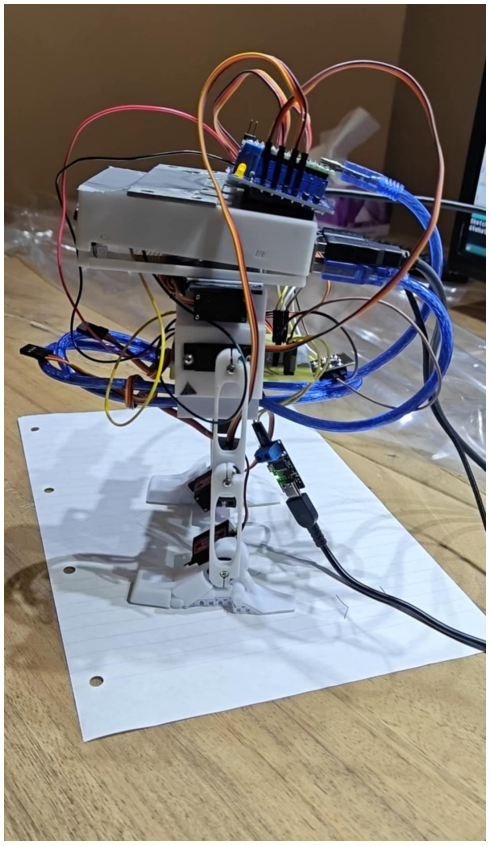
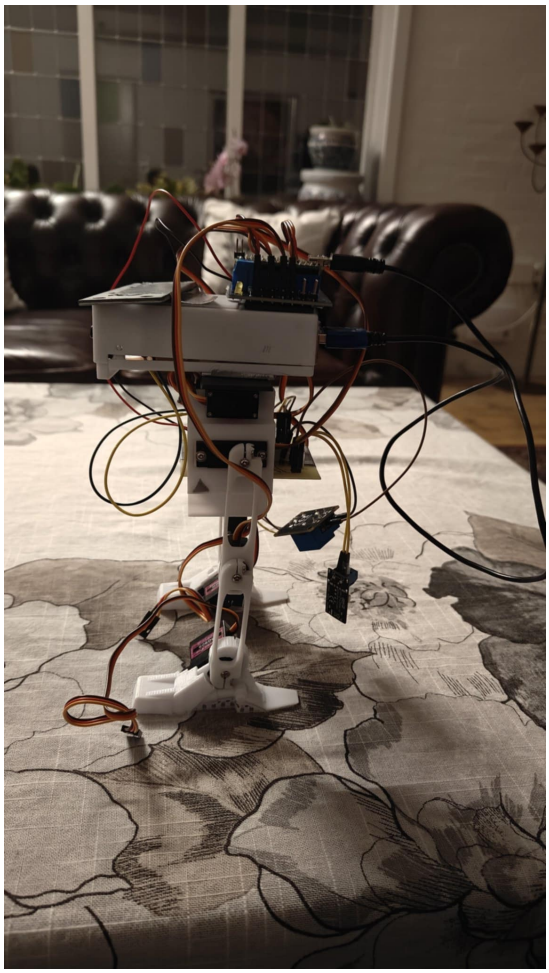
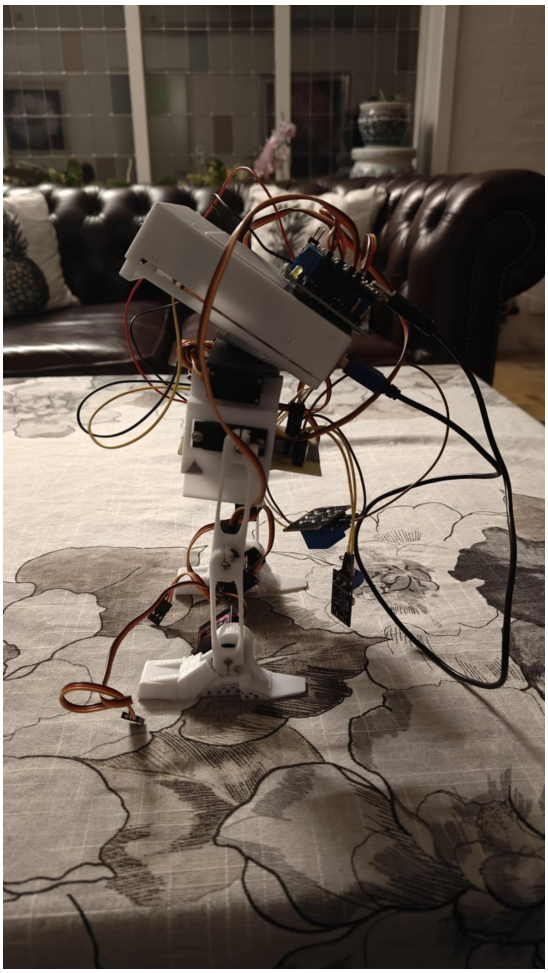


Figure 10: Early prototype of the assembled biped robot.

Later, the long USB cable was replaced with a shorter, lighter version, which greatly reduced rearward strain and improved head stability. Two configurations were tested with the new cable: one with the counterweight and one without. With the plate removed, the head tilted slightly but remained attached. In contrast, using the heavier cable without the plate caused the head to detach during movement.



(a) New cable with counterweight.



(b) New cable without counterweight.

Figure 11: Testing the robot with the new USB cable under two configurations: with and without front-mounted counterweight.

While the prototype met its core functionality goals, several limitations became apparent during the build process. Future designs should address these issues by thickening the legs for added rigidity and developing a more compact, centralized electronics layout. Ideally, the functionalities of both PCBs should be consolidated into a single board, reducing both weight and wiring complexity. A practical and more reliable approach would be to finalize the electronics setup first, then design the supporting 3D model around it. This would help ensure mechanical stability, simplify assembly, and promote cleaner integration between hardware and structure.

5 Serial Hardware Communication and Uploading

The software system is divided between the Raspberry Pi 5, which handles perception and high-level logic, and the Arduino board, which executes low-level motion control and servo commands. Communication between the two is established over a serial interface.

5.1 Serial Communication Setup

Initially, the Arduino Pro Mini was connected to the Raspberry Pi's GPIO UART pins using a SparkFun BOB-12009 logic level converter [11] to safely convert between the Arduino's 5V logic and the Pi's 3.3V-tolerant GPIO. Specifically:

- Raspberry Pi GPIO 14 (TXD, physical pin 8) $\rightarrow$ Logic Level Converter $\rightarrow$ Arduino RX
- Raspberry Pi GPIO 15 (RXD, physical pin 10) $\leftarrow$ Logic Level Converter $\leftarrow$ Arduino TX
- Common GND shared between both systems

This UART interface is exposed on the Raspberry Pi as /dev/ttyAMA0. While functional, this method presented a major limitation during development: code could not be uploaded to the Arduino without manually pressing the reset button at the right moment.

5.2 DTR Reset Limitation on Pro Mini

Uploading code to the Arduino Pro Mini using the Raspberry Pi's UART GPIO pins requires the Arduino to be in its bootloader state—typically achieved by manually resetting it just before the upload. On standard Arduino boards or when using a USB-to-serial adapter, this reset is automatically triggered using the DTR (Data Terminal Ready) line.

However, the Raspberry Pi's GPIO UART interface does not provide a DTR or RTS line. As a result, no automatic reset occurs when uploading code, and the user must manually press the reset button on the Arduino Pro Mini immediately before initiating the upload in the Arduino IDE. Timing this reset correctly is crucial, as the bootloader only listens for incoming uploads for a brief window after reset.

Various community members have suggested workarounds to address the absence of a DTR line on the Raspberry Pi. One commonly proposed method is to control the Arduino's reset pin using a GPIO pin from the Pi—by briefly pulling the reset line low through direct GPIO control [12]. This approach allows the Pi to trigger the bootloader sequence automatically. Others have shared scripts and custom tools, such as avrdude-rpi, which integrate GPIO-based resets into the upload process [13]. While effective, these methods require extra hardware or configuration and were ultimately not used in this project.

5.3 Simplified Upload via USB (Arduino Nano)

To streamline the development workflow and avoid manual resets, an Arduino Nano was introduced and connected to the Raspberry Pi via USB. This shows up as `/dev/ttyUSB0` and includes a functional DTR line, allowing automatic resets during upload.

This setup required no logic level shifting, as the USB-to-serial converter onboard the Nano handles voltage differences internally and offers plug-and-play compatibility with the Arduino IDE. This dramatically improved reliability and ease of development.

6 Robot Movement

The Raspberry Pi 5 runs Raspberry Pi OS and serves as the robot's high-level control unit. The Arduino IDE was installed directly on the Pi, enabling code to be written and uploaded to the Arduino Nano without the need for a separate computer.

6.1 Servo Selection and Behavior

The MG90S micro servo was chosen for this project due to its compact size and torque capabilities, making it a common choice in robotics applications. However, it is important to distinguish between two functionally different types of MG90S servos:

- **Positional (Standard) MG90S**: This type can rotate to a specific angle between 0 degrees and 180 degrees and is used for precise joint-like movements. These servos interpret the write() function as a command to rotate to a specific angle, and hold position using internal feedback mechanisms.
- **Continuous Rotation MG90S**: This modified version can spin indefinitely in either direction, behaving more like a DC motor with directional control. Unlike standard servos, continuous variants cannot be positioned at an exact angle, nor can their speed be precisely controlled.

Although both versions are compatible with the Servo.h library, they respond differently to the same commands:

- write(90) – Neutral: stops movement (continuous) / midpoint (positional)
- write(0) – Full-speed one direction (continuous) / rotate to 0° (positional)
- write(180) – Full-speed opposite direction (continuous) / rotate to 180° (positional)

Figure 12 shows a test sketch for a continuous rotation servo. Figure 13 shows a sweep example for a positional servo, which was the type used in this project for all joint control.

Servo Motor Micro MG90S Test Program

```
/*
  Exercise 360 degree Servo motor
  Simply runs servo in one direction for 3 seconds, stops, then reverses direction
  Uses built-in Servo.h library
*/
#include "Servo.h"
#define SERVO_PIN 9   // Can use any PWM pin

Servo servo;          // creates servo object used to control the servo motor
//=============================================================================
//  Initialization
//=============================================================================
void setup()
{
  servo.attach(SERVO_PIN); // assigns PWM pin to the servo object
}
//=============================================================================
//  Main
//=============================================================================
void loop()
{
  servo.write(0); //Spin in one direction
  delay(3000);
  servo.write(90); // Stop
  delay(500);
  servo.write(180);  // Spin in opposite direction
  delay(3000);
  servo.write(90);  // Stop
  delay(500);
}
```

Figure 12: Test code for a continuous rotation MG90S servo [14]. The servo spins in one direction, stops, then reverses.

```
/* Sweep
 by BARRAGAN <http://barraganstudio.com>
 This example code is in the public domain.

 modified 8 Nov 2013
 by Scott Fitzgerald
 https://www.arduino.cc/en/Tutorial/LibraryExamples/Sweep
*/

#include <Servo.h>

Servo myservo;  // create Servo object to control a servo
// twelve Servo objects can be created on most boards

int pos = 0;    // variable to store the servo position

void setup() {
  myservo.attach(9);  // attaches the servo on pin 9 to the Servo object
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15 ms for the servo to reach the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15 ms for the servo to reach the position
  }
}
```

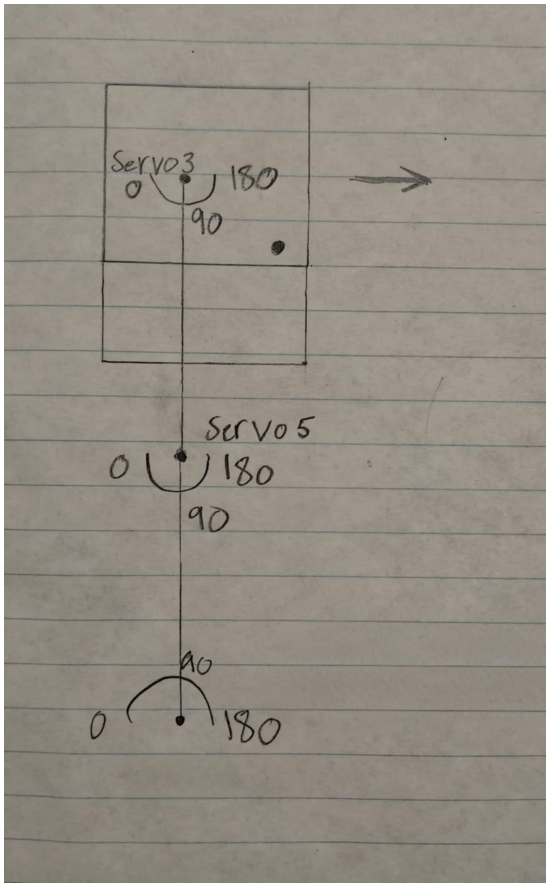
Figure 13: Sweep example for a positional MG90S servo using the built-in Arduino example.

6.2 Servo Initialization and Direction Mapping

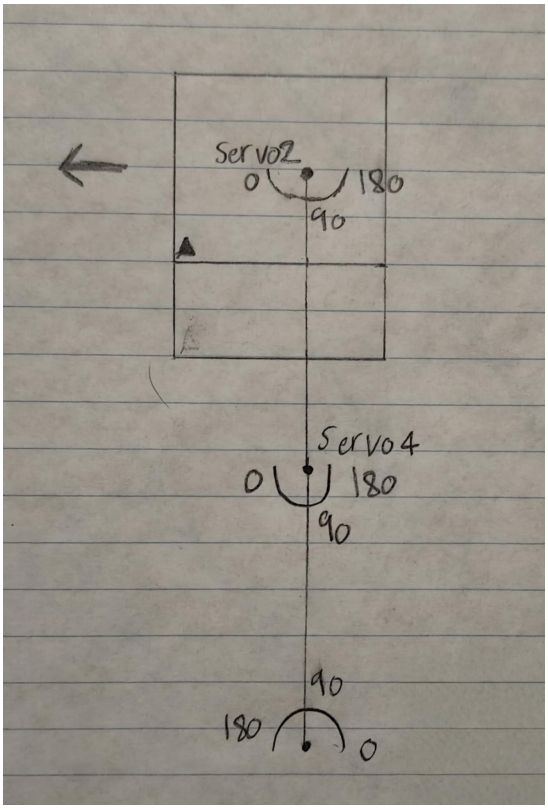
Before the legs could be mounted, it was necessary to ensure that each servo was physically positioned at a neutral midpoint. This was achieved by uploading a simple Arduino sketch to set each servo to a default angle of 90 degrees, the typical neutral position for positional servos. This initialization aligned all servo shafts consistently and allowed components such as the upper legs to be mounted perpendicular to the ground, as detailed earlier in Section 4.

While assembling the robot, it was initially unclear how each servo was oriented—that is, which direction it would rotate when given values below or above 90 degrees. To resolve this, a quick test was conducted for each servo: it was first commanded to rotate to 0 degrees, then to 180 degrees, allowing observation of its movement direction relative to the neutral midpoint 90 degrees. Once the direction was confirmed, the servo was reset to 90 degrees and mounted into position with its output shaft aligned vertically, as described in the Assembly and Build Process section.

To keep track of rotation direction and avoid confusion during further testing, small visual markers were added to the chassis: a circle was drawn on the right side and a triangle on the left. At the same time, a quick sketch of the robot was made from both the left and right perspectives. These visual cues and diagrams served as personal reference tools during development, helping streamline the testing process and reduce trial-and-error when adjusting servo values in software.



(a) Right-side sketch with movement annotations (circle symbol).



(b) Left-side sketch with movement annotations (triangle symbol).

Figure 14: Sketches made during servo testing to document direction of movement for each side of the robot. Used alongside physical markers to assist with calibration.

6.3 Programming the Walking Sequence

Inspired by the human gait cycle depicted in Figure 15, the robot's motion was developed using a frame-based approach. To simulate walking, the full sequence was broken down into individual stages, each written and tested in a separate Arduino sketch. These stages will be referred to as frames throughout this report, a term I use to describe each discrete segment of the walking sequence.

The process began with a neutral standing pose. The first frame made the robot take a step forward with its right leg. The next frame shifted the robot's weight slightly forward using a combination of hip and ankle movement. After that, the left leg was programmed to take a step forward. Finally, the robot returned to a neutral standing position, ready to repeat the cycle.

Once all frames were tested individually and found to work reliably, they were combined into a single walking sequence. This sequence was later triggered by hand detection input from the Raspberry Pi, causing the robot to walk one step each time a hand was detected. All code files for the individual movement frames are included in the appendix.

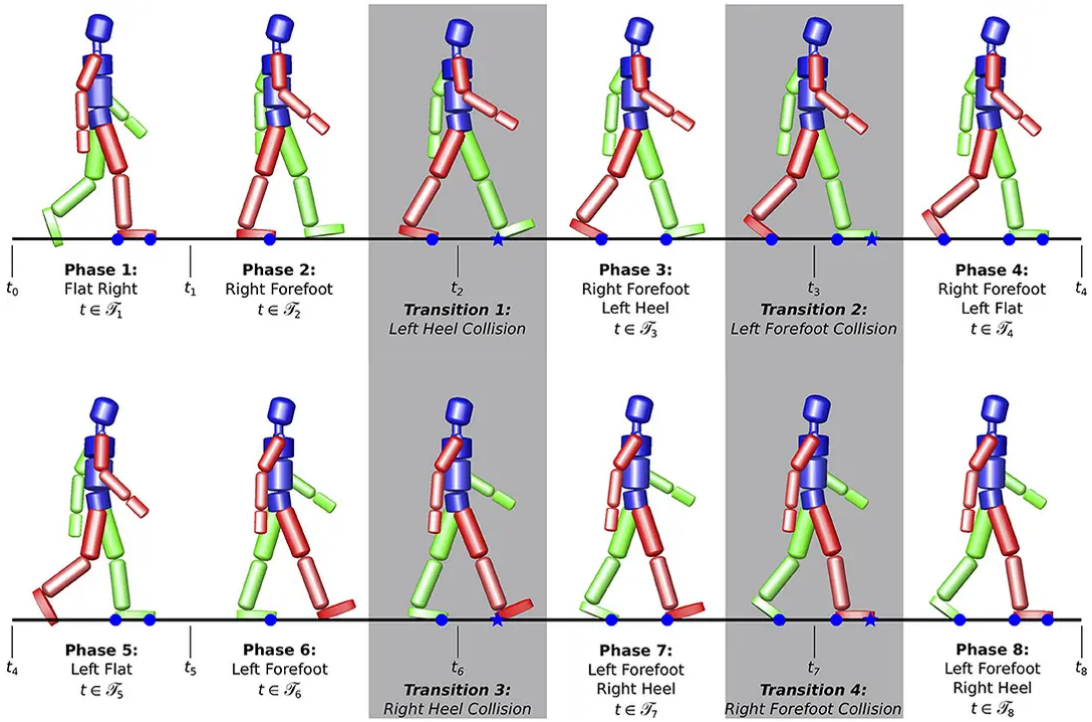


Figure 15: Human gait phases from heel strike to toe-off and the transitions between single and double support periods [15].

6.4 Challenges and Observations During Gait Development

Human bipedal locomotion involves a delicate interplay of coordinated joint motion, weight transfer, and continuous balance correction. Drawing from this model, the goal was to replicate a simplified gait using timed servo movements. The gait cycle, divided into stance and swing phases, includes subphases such as heel strike, mid stance, toe off, and swing recovery [15]. While these principles informed the frame-based walking sequence, many of the biomechanical subtleties proved difficult to replicate given the robot's hardware limitations.

One of the main challenges encountered was knee actuation. Early experiments that involved bending the knees often led to collapse during movement. The MG90S servos lacked sufficient torque to hold position under load, especially in the lower joints, and the 3D-printed parts exhibited slight flexing that reduced overall stability. As a result, knee-based movement was abandoned in favor of using only the hip and ankle joints for stepping and balance shifts.

The order and timing of joint movements also required careful tuning. For example, introducing ankle tilts too early in a frame often destabilized the robot, while insufficient hip movement failed to generate forward momentum. Through repeated testing of individual frames, servo angles and delays were adjusted until each transition

could be executed without tipping.

Testing conditions played a major role in early instability. Much of the initial development was done on a soft tablecloth, which introduced unpredictable surface deformation. When testing was moved to a rigid, level surface, the robot performed more consistently and exhibited fewer falls. This highlighted the importance of a controlled environment when evaluating balance and walking behavior in bipedal systems.

Although the resulting gait remains basic compared to human locomotion, the structured and modular programming approach enabled a functional and repeatable stepping motion. By isolating and refining each movement frame, a complete walking cycle was achieved that could be reliably triggered through visual input. The process provided valuable insight into the constraints and opportunities of servo-based gait control and serves as a strong foundation for future iterations.

6.5 Hand Detection and Motion Triggering

Once each movement frame was functioning correctly, they were combined into a single Arduino sketch named robot_movement (included in the appendix) and paired with a Python-based hand tracking script running on the Raspberry Pi 5. Using the Picamera2 and MediaPipe libraries, the script extracted landmark positions from the detected hand and calculated the distance between the base of the index and pinky fingers. This measurement served as an indicator of proximity.

```
# Draw hand landmarks and calculate distance
if results.multi_hand_landmarks:
    for handLms in results.multi_hand_landmarks:
        mp_draw.draw_landmarks(frame, handLms, mp_hands.HAND_CONNECTIONS)

        h, w, _ = frame.shape
        lmList = []
        for lm in handLms.landmark:
            lmList.append((int(lm.x * w), int(lm.y * h)))

        if len(lmList) >= 18:
            x1, y1 = lmList[5]   # Base of index finger
            x2, y2 = lmList[17]  # Base of pinky finger

            pixel_width = math.hypot(x2 - x1, y2 - y1)

            if pixel_width != 0:
                distance_cm = calculate_distance(KNOWN_WIDTH, FOCAL_LENGTH, pixel_width)
```

Figure 16: Python code snippet showing hand landmark extraction and distance estimation using the base of the index and pinky fingers.

When the estimated distance exceeded a threshold of 40 cm, the Raspberry Pi sent the character 'M' over USB serial to the Arduino Nano. On the Arduino, this input triggered the walking sequence. Since the Python script ran continuously, a state management flag named isMoving was introduced in the Arduino code to avoid mid-motion interruptions. This flag temporarily blocked new inputs until the current movement cycle completed, as illustrated in Figure 17.

This integration marked an important milestone in the robot's development. Motion was no longer triggered manually but instead activated in response to a visual stimulus, creating a more natural and intuitive form of interaction.

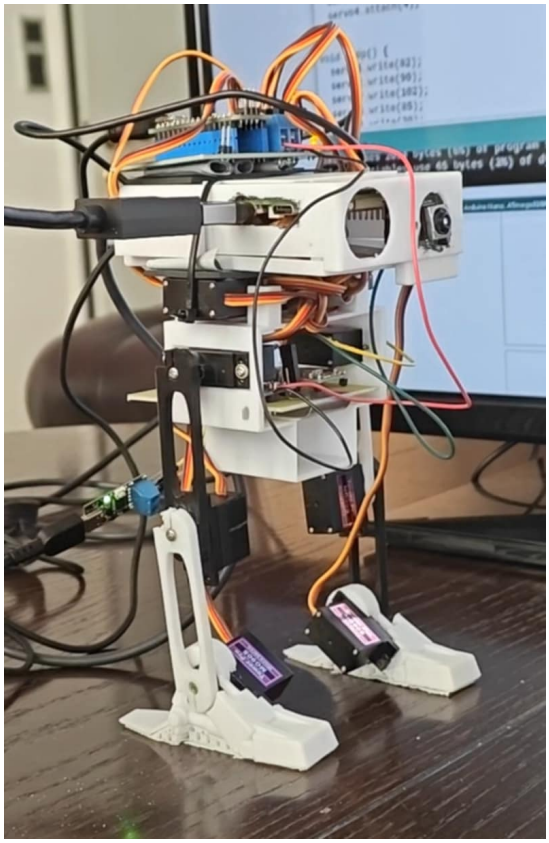
```ino
void setup() {
  // Attach servos
  servo5.attach(5);
  servo2.attach(2);
  servo7.attach(7);
  servo3.attach(3);
  servo6.attach(6);
  servo4.attach(4);

  // Set initial neutral pose
  servo5.write(82);   // Left hip
  servo2.write(90);   // Left ankle
  servo7.write(102);  // Right hip
  servo3.write(85);   // Right ankle
  servo6.write(30);   // Right knee
  servo4.write(65);   // Left knee

  Serial.begin(9600); // For Raspberry Pi communication
}

void loop() {
  if (Serial.available()) {
    char command = Serial.read();
    if (command == 'M' && !isMoving) {
      moveSequence();
    }
  }
}
```

(a) Initialization and serial input handling. The servos are attached, the robot is set to a neutral position, and serial communication with the Raspberry Pi is initialized. The loop continuously listens for the character `'M'` and checks whether the robot is already in motion before triggering a movement sequence.

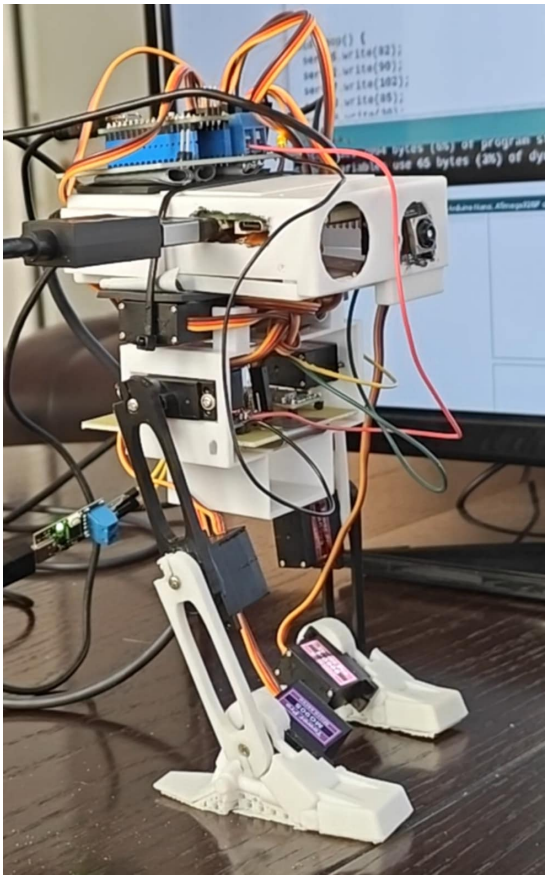
```ino
void moveSequence() {
  isMoving = true;

  // === Sequence 1 ===
  servo7.write(130);
  servo3.write(100);
  servo2.write(95);
  delay(180);
  servo7.write(117);
  servo3.write(105);

  delay(600);

  // === Sequence 2 ===
  smoothMove(servo2, 95, 105, 15);
  smoothMove(servo3, 105, 85, 15);

  delay(600);

  // === Sequence 3 ===
  servo3.write(75);
  servo5.write(70);
  servo2.write(100);
  delay(180);

  delay(600);

  // === Sequence 4 ===
  smoothMove(servo7, 117, 102, 15);
  smoothMove(servo3, 75, 85, 15);
  smoothMove(servo5, 70, 82, 15);
  smoothMove(servo2, 100, 90, 15);

  isMoving = false;
}
```

(b) Movement routine. The isMoving flag is set to true at the beginning of the sequence and reset to false at the end, ensuring only one complete movement cycle runs at a time.

Figure 17: Key segments from the Arduino sketch that handle incoming commands, initiate motion, and prevent overlapping execution.



(a) Initial position: robot in neutral stance before executing the first movement frame.



(b) After first step: right leg has been moved forward as part of the initial walking frame.

Figure 18: Demonstration of the robot executing the first programmed movement frame: stepping forward with the right leg.

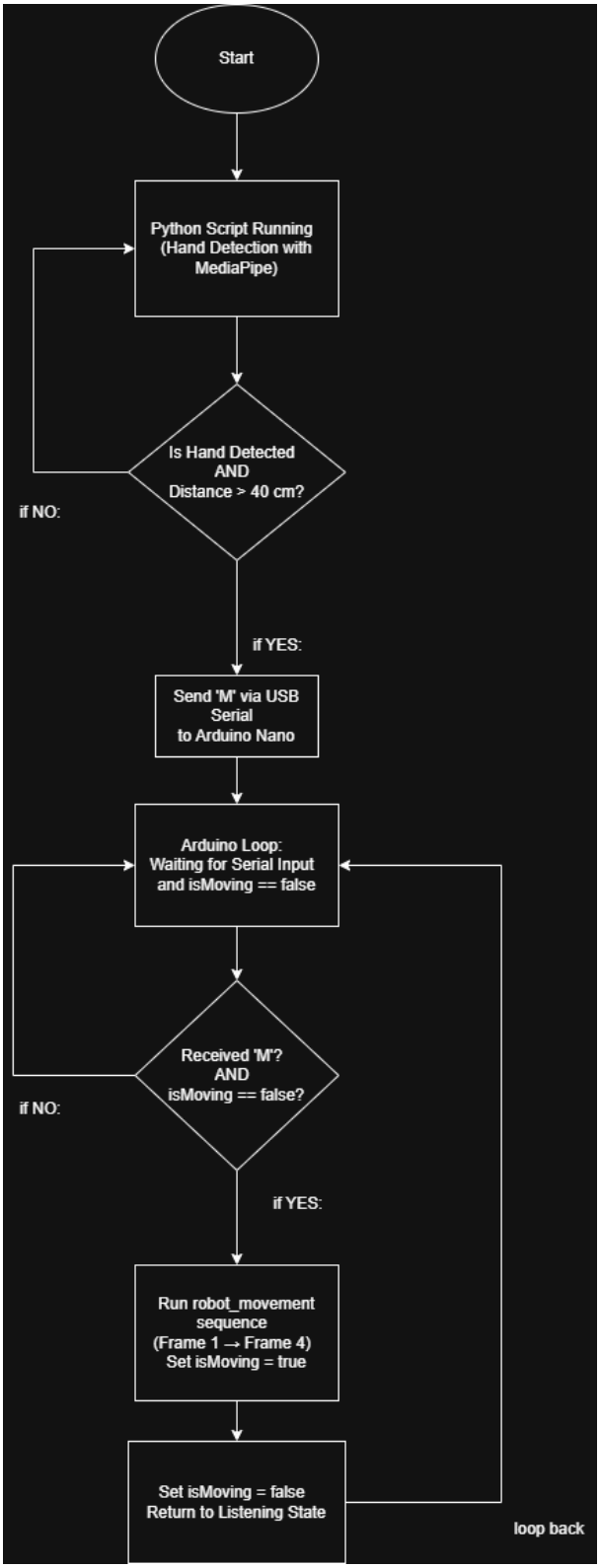


Figure 19: Flow diagram illustrating the interaction between the hand detection script on the Raspberry Pi and the Arduino Nano. When a hand is detected at a distance greater than 40 cm, the character `'M'` is sent to the Arduino. If the robot is not already moving, it begins the walking sequence. Once the sequence completes, the isMoving flag resets and the system returns to listening state.

7 Conclusion

This project set out to design, build, and implement a functional bipedal walking robot using accessible components, open-source designs, and basic sensing capabilities. From mechanical assembly to perceptual integration, each stage of development offered insight into both the challenges and creative opportunities in low-cost humanoid robotics.

The robot was constructed primarily from 3D-printed PLA components based on the Modular Biped project by Dan Makes Things. These parts were combined with MG90S servo motors, an Arduino Nano for joint actuation, and a Raspberry Pi 5 for high-level computation and vision-based input. Modifications were made to accommodate a custom extension PCB, reroute internal wiring, and mount the Raspberry Pi securely within the existing head structure. These adaptations deepened the understanding of key mechanical principles such as weight distribution and center of mass, as well as practical design considerations related to hardware integration, such as the challenges of uploading code to the Arduino Pro Mini without automatic reset, and the trade-offs involved in using multiple PCBs for development and control.

A major focus of the project was walking behavior. Drawing inspiration from human gait phases—visualized in robotics literature—the robot's walking sequence was developed through a modular, frame-based approach. Each step of the gait was isolated, programmed, and tested individually to ensure proper timing and coordination. The final implementation successfully integrated both hip and ankle articulation, allowing the foot to remain level during the swing phase and improving overall balance and realism.

The robot's walking behavior was then linked to hand detection using a Python-based vision pipeline running on the Raspberry Pi. A hand in view triggered a predefined walking sequence via serial communication to the Arduino. A simple flag system ensured that each movement sequence completed before a new one could begin. This integration demonstrated a foundational form of perception-driven interaction, turning a manually controlled robot into a more autonomous and reactive system.

Some limitations still remained. Active knee actuation was ultimately excluded due to mechanical instability in the joints, and the structure's high center of mass restricted the range and fluidity of possible movements. The walking motion, while structured and consistent, lacked real-time adaptation and could not dynamically respond to external forces or terrain variation. Due to the robot's open-loop architecture and absence of onboard sensors, it was not possible to implement feedback-driven stability or terrain-aware adjustments.

Nevertheless, these constraints became learning opportunities. Structural reinforcements, reallocation of electronics, and improved joint coordination all contributed to incremental progress. The iterative process—from visual sketches and servo calibration to debugging Python–Arduino communication—captured the essence of practical engineering: trial, error, revision, and persistence.

For future iterations, several improvements are evident. Mechanically, the leg joints could be reinforced and designed with tighter tolerances. Electronically, a consolidated custom PCB would reduce weight and wiring complexity. From a control standpoint, incorporating proprioceptive feedback such as encoders or IMUs would enable closed-loop movement, dynamic stability correction, and terrain-adaptive behavior.

Overall, the project highlights how thoughtful iteration, guided by theory and grounded in testing, can transform simple components into a functional robotic system. Although the robot's capabilities remain limited, the experience bridges the gap between academic models and hands-on implementation, laying a solid foundation for more advanced, intelligent, and lifelike bipedal robots in the future.

8 Appendix

The project source code is included as a ZIP archive titled Source_code.zip, organized as follows:

- hand_detection.py — Python script running on the Raspberry Pi for real-time hand detection and distance estimation using Picamera2 and MediaPipe.
- 1.right_leg_forward — Arduino sketches used for developing the first movement frame: stepping forward with the right leg.
- 2.Shift_robot_weight — Code for shifting the robot's center of mass forward after the initial step.
- 3.left_foot_forward — Code for advancing the left leg during the walking sequence.
- 4.return_to_neutral — Code for restoring the robot to a balanced neutral stance, completing one walking cycle.
- robot_movement — Final combined Arduino sketch integrating all four movement frames into a complete walking sequence, triggered by hand detection.

9 References

[1] Dan Makes Things. Modular biped: Archie home – wiki documentation. https://github.com/makerforgetech/modular-biped/wiki/Archie-Home, 2021. Accessed: 2025-05-24. While the repository contains commits from 2019, documentation and build materials related to the modular biped project date from October 2021.

[2] Dan Makes Things. Building a modular biped robot – youtube series. https://www.youtube.com/watch?v=2DVJ5xxAuWY&list=PL_ua9QbuRTv6Kh8hiEXXVqywS8pklZraT, 2021. Accessed: 2025-05-24. Series includes:
1. *Companion Robot: Overview - Raspberry Pi — Arduino — DIY Robotics* (Oct 11, 2021),
2. *Companion Robot: Part 2 - Construction and Teardown* (Nov 29, 2021),
3. *Part 3 - Software Deep Dive* (Dec 4, 2021),
4. *Part 4 - 3D Model* (Dec 10, 2021).

[3] Dan Nicholson. Companion robot 'archie' complete. https://www.tinkercad.com/things/63OXZasclFs-companion-robot-archie-complete, 2021. Tinkercad project by dan.nicholsonBD7PJ, created December 2, 2021. Accessed: 2025-05-24.

[4] Com-zypdn-st usb-pd trigger module datasheet. https://cdn-reichelt.de/documents/datenblatt/A300/COM-ZYPDN-ST_DATASHEET_2021-11-10.pdf, n.d. Accessed 27 May 2025.

[5] Monolithic Power Systems. Mp2307 – 3a, 23v, 340khz synchronous rectified step-down converter. https://www.monolithicpower.com/en/documentview/productdocument/index/version/2/document_type/Datasheet/lang/en/sku/MP2307/document_id/104/, 2008. Rev. 1.9. Accessed 27 May 2025.

[6] Arduino. Arduino pro mini. https://docs.arduino.cc/retired/boards/arduino-pro-mini/, 2024. Accessed 27 May 2025.

[7] Micro servo motor mg90s - tower pro. https://www.electronicoscaldas.com/datasheet/MG90S_Tower-Pro.pdf?srsltid=AfmBOop3w7wDgRmk9IoJd_QNrWNt90ToFO-uQVepQJ8OPty2tkW7NvrR, n.d. Accessed 27 May 2025.

[8] Raspberry Pi Foundation. Raspberry pi documentation - raspberry pi hardware. https://www.raspberrypi.com/documentation/computers/raspberry-pi.html, n.d. Accessed 27 May 2025.

[9] Dan Makes Things. Modular biped - circuitsv2. https://github.com/makerforgetech/modular-biped/tree/archie/circuits/v2, 2021. Accessed: 2025-05-24.

[10] Hackatronic. Raspberry pi 5 pinout, specifications, pricing – a complete guide. https://www.hackatronic.com/raspberry-pi-5-pinout-specifications-pricing-a-complete-guide/, 2023. Accessed: 2025-05-24.

[11] SparkFun Electronics. Sparkfun bob-12009 logic level converter. Mouser Electronics, 2024. Accessed: 2025-06-02.

[12] technix. Programming arduino without usb. https://forums.raspberrypi.com/viewtopic.php?t=78099, 2014. Raspberry Pi Forums, Accessed: 2025-05-24.

[13] Dean Mao. Using avrdude with the raspberry pi. https://github.com/deanmao/avrdude-rpi, 2012. Accessed: 2025-05-24.

[14] ProtoSupplies. Servo motor micro mg90s – 360 degree continuous rotation. https://protosupplies.com/product/servo-motor-micro-mg90s-continuous-rotation/, n.d. Accessed: 2025-05-26.

[15] Fiveable. 5.1 bipedal locomotion – robotics and bioinspired systems, 2024. Edited by Becky Bahr. Accessed 27 May 2025. https://library.fiveable.me/robotics-bioinspired-systems/unit-5/bipedal-locomotion/study-guide/7AsvW7aYU24DFkhP.