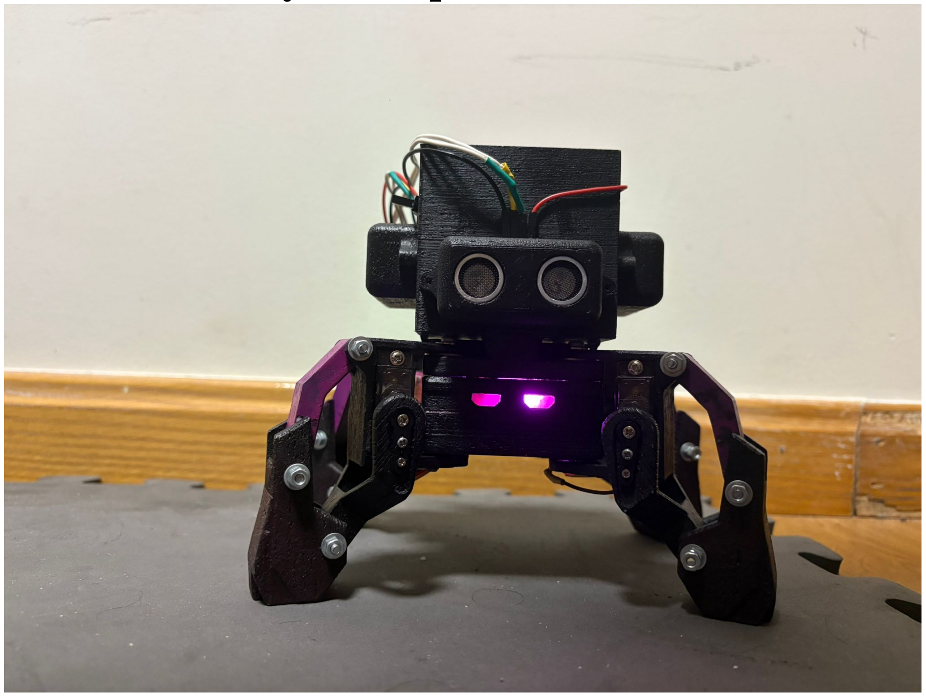
Brutus: Design and Implementation of a Quadruped Robot



Sergio Sanchez Hernandez, Jorge Barroso Saugar, Sergio Cobos Blanco

January 2026

Contents

1 Introduction

This technical report details the design and implementation of the Brutus quadrupedal robot, providing an in-depth analysis of its structural design, hardware specifications, and FreeRTOS-based embedded software control.
The robot must meet the following specifications:

- Its dimensions must not exceed $30 \times 30 \times 30$ cm.
- It must be powered by rechargeable batteries.
- The robot must feature a basic interface to start and stop its operation.

It must also pass the following two tests:

- **Wall-Following Test:** In less than 1 minute, the robot must move parallel to a flat wall, placed on either side of the robot, for at least 1 meter. Additionally, it must carry a 250-gram load.
- **Obstacle Avoidance Test:** In less than 2 minutes, the robot must be capable of avoiding obstacles while moving within a 2×2 meters area, reaching each of the adjacent walls. The robot starts from the center of the square.

2 Structural Design

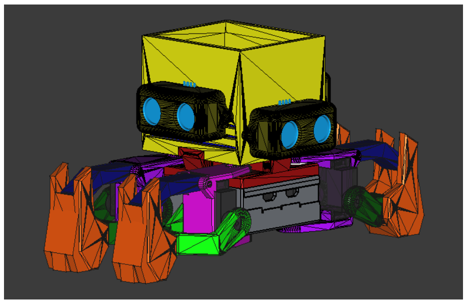


Figure 1: Full assembly of the Brutus robot in FreeCAD.

2.1 Design selection and justification

The design of Brutus is based on a model found on Thingiverse.

This design was selected for its compact footprint, ensuring full compliance with the size specifications and less effort for the servos at the joints. Additionally, the two degrees of freedom in each of the four legs allow for simplified control while maintaining high mobility standards.

One of the most remarkable features of this design is the **parallel linkage mechanism** incorporated into the legs. This configuration ensures that the foot remains perpendicular to the ground throughout its vertical range of motion. Consequently, onboard sensors are isolated from tilt variations along the X and Y axes, maintaining a stable reference frame during locomotion.

2.2 Comparison with Original Design

The original models underwent several aesthetic and practical modifications to suit the project requirements:

- Both the **'shoulders'** and the **chassis** were modified to accommodate MG90s servo motors.
- A custom top module was designed to house the power supply battery and circuit, and distance sensors.
- The **chassis cover** was redesigned to attach the upper module (upside) and the ESP32 (facing down).

2.3 Materials and Manufacturing

The structural components were manufactured using 3D printing. **PLA** was selected as the primary material due to its high tensile strength, ease of printing, and dimensional stability, which is critical for the alignment of the leg mechanisms.

3 Electronic design

3.1 Actuators

The selected actuators must therefore provide sufficient torque, adequate speed, electrical compatibility and acceptable size and weight.

The servos must be capable of:

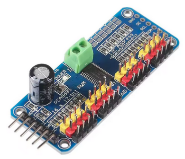
- Supporting the weight of the robot and the load during the support phase of walking.
- Producing enough torque to lift and move the legs.
- Moving fast enough to generate a stable and smooth gait.
- Allow PWM control.

The robot uses **MG90S micro servos** for all leg joints. Small, lightweight, strong and fast-rotation actuators, which are a really good election for the robot.

Figure 2: MG90s servo

The MG90S servo provides a nominal stall torque of approximately 1.86 $kg \cdot cm$, which correspond to 0.1824 $\frac{N}{m}$.

The typical speed of this servos is 0.1 $s/600°$, so we know that they can reach $600°/s \approx 10.47\ rad/s$. This ensures that the simulated gait is realistic and achievable by the real servos. The MG90S provides sufficient speed to generate a smooth and stable walking motion without causing excessive inertia or oscillations.


The 8 servos are controlled using a **PCA9685**. This is a **16-channel, 12-bit PWM I2C-bus controlled driver** that significantly offloads the processing work from the main microcontroller. By using this external driver, we can control all 8 servomotors using only two pins (SDA and SCL), which simplifies the wiring and ensures high-precision timing for the leg movements.

The final actuator is an **RGB LED module**, which serves as a visual diagnostic tool. This component enables real-time debugging by representing up to eight distinct color combinations, allowing for the intuitive monitoring of various system states and logic conditions.

3.2 Sensors

The sensors selected for resolving the exercises were the **HC-SR04** for distance measurement. These are cost-effective ultrasonic sensors that operate by emitting high-frequency sound waves and measuring the time it takes for the echo to return after hitting an object.

With a detection range of $\approx$ 2 cm to 400 cm and an accuracy of $\approx$ 3 mm, this small sensors are ideal for the **Wall-Following** and **Obstacle Avoidance** tests. In our configuration, three sensors were mounted on the custom top module, facing front, left, and right, to provide a 180° field of view, allowing the Brutus robot to maintain a constant distance from the walls while simultaneously detecting potential collisions in its path.

3.3 Control unit

The **ESP32-WROOM-32** was selected as the central processing unit due to its superior technical specifications. Its **dual-core architecture** enables the isolation of the gait control algorithm from perception and high-level logic tasks, ensuring stable locomotion. Integrated **WiFi and Bluetooth** connectivity facilitate remote commanding and real-time telemetry via an external web interface.

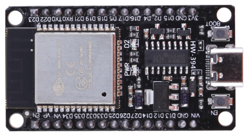


Figure 3: ESP32

Furthermore, its high clock speed and memory capacity provide a significant performance advantage over traditional platforms like Arduino UNO. The native support for **FreeRTOS** is crucial for implementing Brutus as a **Real-Time System (RTS)**, allowing for deterministic task scheduling. Finally, its compact form factor ensures a seamless fit within the chassis constraints.

3.4 Power supply unit

Total current (peak and continuous) required by the system:
$I_T = I_{ESP32} + I_{HC-SR04} \times 3 + I_{LED} + I_{MG90s} \times 8 + I_{PCA9685}$

$I_{peak} \approx 0.3 + 0.015 \times 3 + 0.06 + 1.1 \times 8 + 0.025 = 9.23\ A$
$I_{continous} \approx 0.3 + 0.002 \times 3 + 0.06 + 0.4 \times 8 + 0.025 = 3.29\ A$

We want Brutus to operate for at least half an hour, so the minimum **required capacity** in the worst case must be:
$q_T = 9.23\ A \times 0.5\ h = 4.615\ Ah = 4615\ mAh$

These are the components (with their nominal voltage) which will be powered by the battery:

- ESP32 (5 V)

- PCA9685, and servos connected to it (5 V)
- HC-SR04 (5 V)

Our battery has to give **5 V**.

The selected battery is a **3S1P Li-ion battery** (12 V, 5200 mAh) was chosen because it satisfies all the electrical, power and autonomy requirements of the robot while remaining compact and lightweight. This battery has a discharge continous current of 5 A and a discharge peak current of 10 A.

If we consider the "worst case" as a sustained demand close to the battery's continuous current rating:
$T_{worstcase(peak)} = 5.2/9.23 = 0.56\ h \approx 33.6\ min$
$T_{worstcase(continuous)} = 5.2/3.29 = 1.58\ h \approx 45.6\ min$

The battery provides a voltage range of 8.0 V to 12.6 V, so we choosed to use a **DC-DC Buck Module** to generate a stable 5 V supply for all electronics regardless of the battery's state of charge.

3.5 Electronic Schematic

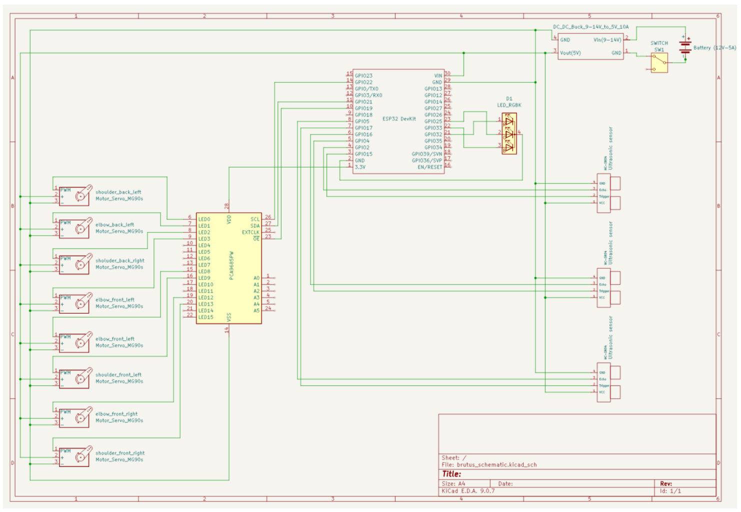


Figure 4: KiCad schematic of the circuit

4 Software and control

5 Software Implementation

The robot's firmware is built upon **FreeRTOS**, enabling its operation as a **Real-Time System (RTS)** and ensuring deterministic execution of all critical tasks. Leveraging the **dual-core architecture** of the ESP32, the locomotion control is isolated on one core, while perception, high-level logic, and communication protocols run independently on the second.
The primary codebase is developed using **Object-Oriented Programming (OOP)** within the Arduino framework, establishing a modular and scalable architecture. The system is structured through the following specialized classes:

- `Pca9685Servo.h`: A low-level interface for direct pulse-width modulation (PWM) control via the PCA9685 driver.
- `BrutusLegInterface.h`: A kinematic abstraction layer for individual leg positioning (dependent on `Pca9685Servo.h`).
- `BrutusPerception.h`: This class manages the operation of the HC-SR04 ultrasonic sensors.
- `Brutus.h`: A comprehensive manager for the robot's global state, coordinating all integrated actuators and sensors (dependent on `BrutusLegInterface.h` and `BrutusPerception.h`).
- `BrutusComms.h`: A dedicated interface for asynchronous MQTT-based communications.

5.1 Locomotion and Gait Algorithm

The locomotion strategy is based on a **diagonal trot gait**, characterized by a sequence of poses where two diagonal pairs of legs move in synchronization. While one pair performs the *swing phase* (moving through the air), the opposite pair executes the *stance phase* (contact with the ground), providing the necessary thrust to propel the chassis forward.
For high-level maneuvering, a **velocity-based control scheme** was implemented. This allows the system to translate linear velocity commands (v_x) and angular velocity commands (ω) into coordinated joint movements:

- **Translation:** Achieved by modulating the step frequency and stride length.
- **Rotation (Yaw Control):** Instead of complex inverse kinematics, an efficient **differential thrust** method was used. By reducing the angular sweep (amplitude) of the "shoulder" servos on the inner side of the turn, the outer legs generate a larger displacement vector, effectively pivoting the robot toward the desired direction.

5.2 Remote Interface and Telemetry

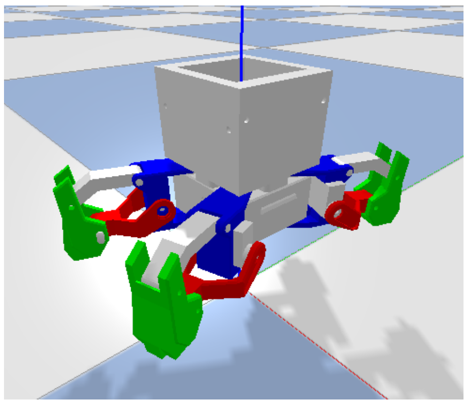
Brutus maintains a persistent connection to an **MQTT broker**, facilitating real-time telemetry, state switching, and remote commanding. To satisfy the requirement for a visual user interface, both a native **Android application** developed in Android Studio and a custom **CLI (Command Line Interface)** script were implemented, allowing for seamless interaction with the robot's internal states.

5.3 Concurrency and Thread-Safety

Perception is handled by a periodic FreeRTOS task that samples ultrasonic distance data. To ensure **data integrity** in a multi-core environment, these values are accessed via **thread-safe methods** protected by **Binary Semaphores (Mutexes)**. This synchronization mechanism prevents *race conditions* when the high-level logic task requests sensor data while the perception task is updating it. Similar protection is applied to the velocity command variables.
The priorities for the multi-task core are: 1. Perception, 2. High-Level logic, 3. Communications.

6 Simulation and Design Validation

The primary objective of the simulation was to validate the robot's stability, center of mass (CoM), and joint torque requirements through a detailed URDF model. Realistic mass and inertia properties were assigned to each link, including the external load on the top module, to ensure the physical design could withstand operational stresses and maintain coherent kinematics before manufacturing.



Due to the complexity of closed-loop kinematic chains in physics engines, the parallel linkage was simplified in the URDF while functionally maintaining foot perpendicularity. This causes a not really realistic analysis of the torques and the movement. The servomotors were modeled using position control with real-world torque and speed limits ($600°/s$), enabling real-time logging to CSV files to verify that the selected actuators could handle peak loads during locomotion.

7 Experimental Results: Wall-Following Test

Figure 5 shows the optimal trial where the loaded robot traversed 1.6 meters. The right-hand sensor data reveals minor oscillations around the setpoint, caused by intrinsic noise, gait-induced mechanical shifts, and control logic tuning. Despite these fluctuations, the system demonstrated robust wall-following performance under loaded conditions.

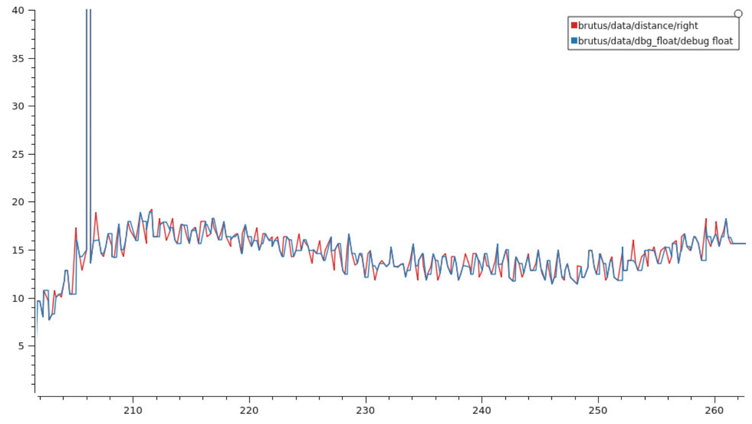


Figure 5: Ultrasonic sensor data during the 1.6-meter wall-following test.

8 Future Work and Proposed Improvements

- **Sensor Fusion and Pose Estimation:** Integrating an **Inertial Measurement Unit (IMU)** to implement filtering algorithms (such as a Kalman or Complementary filter). This would compensate for gait-induced oscillations during the swing phase and provide real-time data on the robot's **attitude (roll, pitch, and yaw)**, ensuring a stable reference frame for navigation.
- **Active Perception (Scanning Radar):** Mounting the ultrasonic sensors on dedicated micro-servos to implement an **active scanning radar** system. This configuration would significantly expand the **Field of View (FoV)**, allowing for the generation of local occupancy maps and improving obstacle detection and path planning in complex environments.
- **Structural Optimization for Serviceability:** Expanding specific chassis sections to enhance **thermal management** through improved airflow. Additionally, redesigning the internal layout to increase **accessibility**, thereby facilitating assembly, cable routing, and routine maintenance procedures.