University of Basel

RoboRetriever - quadruped robot

Anastasiya Merkushova, Jiri Käser, Mathieu Groenen, Rahel Kempf
Lecture "Computer Architecture" by Prof. Dr. Christian Tschudin, HS24

Abstract
In this project, our aim was to build a walking quadruped robot that can be controlled via Wi-Fi. Our four-legged robot is able to walk using servo motors and an ESP32 microcontroller. The project involved designing a 3D model of the robot and printing it, determining the circuit design and power supply, and programming the servos for the robot's movement. The robot is capable of basic movements such as tilting, walking, standing up, and sitting down.

1. Introduction

Inspired by Boston Dynamics' Spot and other developments in quadruped robotics, we set out to create our own quadruped robot, driven by curiosity and a desire to explore the possibilities of four-legged locomotion. This project aims to design, develop, and build a quadruped robot from the ground up, from mechanical design and electronics to inverse kinematics and programming.

Quadruped robots are interesting because they are much more agile than their wheeled counterparts. Research shows their potential in traversing uneven terrain, recovering from falls and more. To remain realistic in the scope and ambitions for our project, we set out simple and hopefully achievable goals. Our goal was to build a robot that could stand up and walk.

In this report, we will delve into our journey to achieve these goals, beginning with the 3d design, in which we had to develop mechanics that would support the movements of our robot. We also had to choose an appropriate PCB design to control 12 servos at once and supply these servos with enough power. For the leg movement, we had to calculate the inverse kinematics and implement them in our code. Finally, to be able to remotely control the robot, we had to implement WIFI connection.

2. 3D Design

In this section, we will discuss the process of designing and testing the 3D model of our robot, as well as the challenges we encountered.

2.1 Physical design

First, we had to get the mental image of the robot moving. To provide our robot with the potential to turn, walk, and stand up we decided to go with three servos per leg, giving it three degrees of freedom.

To understand successful quadruped robot designs, we compared different examples online. In the end, we decided to position the servos in a way that minimizes movement, stabilizing the robot's center of mass. This should simplify programming because we could neglect the factor of mass shifting, though it adds complexity to the mechanical design, requiring precision belts to transfer motion to the lower leg.

2.2 Model

Once the initial design was envisioned, we needed to model it in 3D software, print it, and test it excessively. For modelling, we used Onshape, a lightweight, browser-based 3D modelling software.

2.2.1 Upper and Lower Leg Design

We first modelled the upper and lower legs, along with the knee connection using a bearing. This was done very early on so we didn't yet have the servos as reference, to check if the part would fit. Once the servos arrived, we tested the model. The initial design had several minor defects: the servo didn't fit, there was insufficient space for a gear to be places between servo shaft and enclosure, the screw holes and the bearing hole were too small, and the strength of the part in general was too low. The final design is shown in Figure 1.

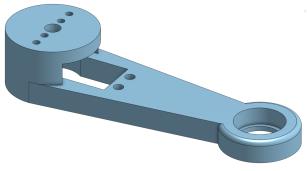

Figure 1. Final version of the upper leg

The lower leg had less issues but there were still a few. Issues were addressed by iterating between adjusting the 3D model according to the problems noticed during testing, reprinting and testing again. The final version of the lower leg can be seen in Figure 2.

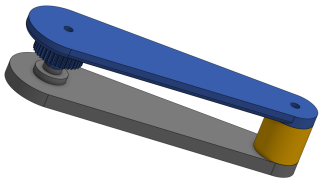

Figure 2. Final version of the lower leg

2.2.2 *Lower Leg Motion*

The next part that was addressed was the gear to drive the belt and therefore the motion of the lower leg. This part faced three main challenges: the gear needed to be compact yet strong enough to support several kilos of force, ensure the belt stayed securely in place, and have the correct gear ratio for optimal timing belt grip while keeping the height minimal.

To ensure strength, the print options were set to 6 wall layers, 50% infill, and a 0.4mm nozzle. To resolve the height issue, a new part was designed to position the servo slightly farther out, providing the extra millimeters needed for proper gear placement. Gear teeth were refined by printing 10 gears with varying teeth counts, testing, and repeating with finer increments until the optimal 35-tooth gear was selected. The final design is shown in Figure 3.

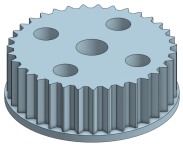

Figure 3. Final design of the gear

The final issue was belt tensioning, for which no ideal solution was found. The process went as follows:

1. A piece of timing belt was tensioned by hand hard enough not to slip any more
2. Marks for the loop closure, cut point, and overlap for sewing were added.
3. Three to four holes were punched into the belt
4. The belt was sewn together by hand with nylon string

This was the most painstaking and annoying process of assembling the robot since a little to less or too much tension meant to repeat the four steps above and they took around 20-30 minutes to complete. In the end we managed to get only two out of four running perfectly. The others worked but slipped when too much force was exerted on them. This had to then be reworked when we were in the programming phase.

2.2.3 *Shoulder Design*

The last two pieces needed were the shoulder, that would hold the servo rotating the upper leg and the main body holding the servos that rotate the shoulder and the electronics. The shoulder was rather straight forward as it simple needed to include a mount for the servo controlling the upper leg and a couple of holes to allow for the main body servos to control the shoulder. There were a couple of issues still, the most problematic being that one threaded insert hole was too large which caused one part of the shoulder to be ripped out during the programming phase. The final design of the shoulder can be seen in Figure 4.

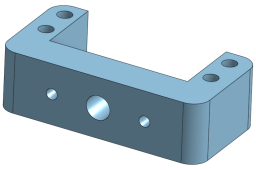

Figure 4. Final design of the shoulder

2.2.4 *Body Design*

The main body also caused a bunch of issues. We potentially wanted to power the robot by a lithium ion battery, therefore there had to be space for this, then the electronics needed to be fixed in space to make sure that cables don't come loose easily and it needed four servo mounts while at the same time staying as light as possible. For the battery mount simply a hole in the floor was made and therefore it can be taped below the center of the main body. For the electronic mount four pillars were designed, that would hold one screw each. The problem with those was that when inserting the ruthexes these pillars would deform easily due to the low heat resistant nature of PLA Filament which made it extremely difficult to keep the correct distances for the electronic mount. The final part can be seen in figure 5.

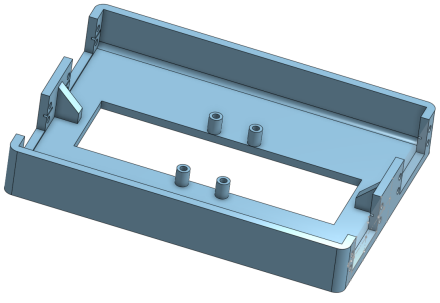

Figure 5. Final design of the main body

3. Electronic design

One of the core challenges in building a quadruped robot is the need to manage multiple servo motors simultaneously. The hexagonal PCB design, shown in Figure 6, was chosen

because it is designed to control up to 16 servo motors[a]. This PCB accommodates an ESP32 microcontroller (ESP32-WROOM-32 DevKit V1[b]), which features built-in Wi-Fi capabilities and allows for seamless control of the robot's operations. The ESP32 can be powered through its USB port (5V), VIN pin, or its 3.3V regulator.

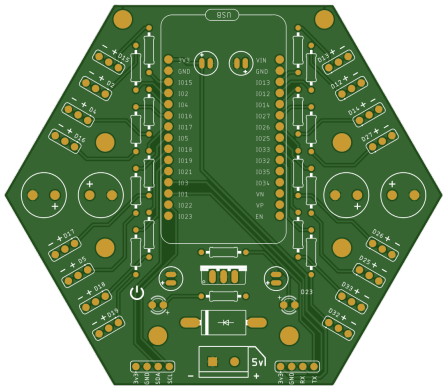

Figure 6. Hexagonal PCB design for servo motor control

3.1 PCB and Power

The PCB was assembled with the necessary components, including connectors, resistors, and capacitors. For the robot, we used 12 servo motors (DIYmore DM996 model), which operate at a voltage range of 4.8 V to 6.0 V, with optimal performance at 6.0 V.

Since the ESP32 operates at 5V using VIN pin, and the motors perform optimally at 6V, we opted to power the ESP32 directly via USB. For the servo motors, we used two power supplies connected in parallel: a 3-channel 5 V supply rated at 4 A and a single-channel 5 V supply rated at 4 A, providing a total current capacity of 8 A. This setup provided sufficient power to meet the robot's peak requirements, as each servo motor has a peak current draw of 500 mA.

4. Inverse Kinematics

In this part, we explore the calculations needed for the inverse kinematics of the robot. Inverse kinematics works by choosing the location of the leg specified by its coordinates (x, y, z) and calculating the angles needed to move the leg to these coordinates. The complete leg movement is thus described by the function $(\theta_h, \theta_s, \theta_w) = F_\theta(x, y, z)$. Our calculations are based on the calculations described in the OpenQuadruped project [c], with some changes made due to differing leg design.

a. PCB design
b. ESP32 Pinout Reference
c. https://www.adhamelarabawy.com/images/IK_Model.pdf

4.1 Derivation

In general, we know the coordinates x, y, z which are specified by the starting condition. Additionally, SL (Shoulder length), UL (Upper leg) and LL (Lower leg) are given.

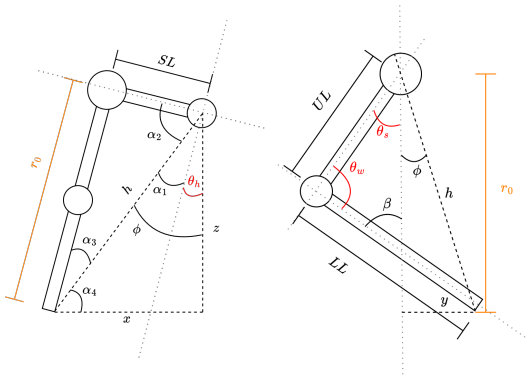

Figure 7. Front (left) and Side (right) view of the leg

First we look at the derivation of the θ_h angle as seen in figure 7, which is the angle between the body and the leg. We first calculate the distance r_0.

$$h = \sqrt{x^2 + z^2} \Rightarrow r_0 = \sqrt{h^2 - SL^2} = \sqrt{x^2 + z^2 - SL^2}$$

The different angles are then given as

$$\alpha_2 = \arccos\left(\frac{SL}{h}\right)$$
$$\alpha_1 = \frac{\pi}{2} - \alpha_2$$
$$\phi = \arctan\left(\frac{x}{z}\right)$$

Which then gives

$$\Rightarrow \theta_h = \phi - \alpha_1 = \arctan\left(\frac{x}{z}\right) - \frac{\pi}{2} + \arccos\left(\frac{SL}{h}\right)$$

Now for the side view as seen in figure 7 we have

$$\phi = \arctan\left(\frac{y}{r_0}\right)$$
$$h = \sqrt{r_0^2 + y^2}$$

For θ_s, the shoulder angle, using the cosine law we get

$$\cos(\theta_s + \phi) = \frac{UL^2 + h^2 - LL^2}{2UL \cdot h}$$
$$\Rightarrow \theta_s = \arccos\left(\frac{UL^2 + h^2 - LL^2}{2UL \cdot h}\right) - \arctan\left(\frac{y}{r_0}\right)$$

For θ_w, the knee angle, using again the cosine law

$$\cos(\theta_w) = \frac{UL^2 + LL^2 - h^2}{2UL \cdot LL}$$
$$\Rightarrow \theta_w = \arccos\left(\frac{UL^2 + LL^2 - h^2}{2UL \cdot LL}\right)$$

Thus, we get formulas for all the needed angles fully dependent on the variables x, y, z, SL, UL and LL.

Because of the design and implementation, the leg has three degrees of freedom, as depicted in figure 8. With our calculations, we are enabled to freely position the robot's leg based on its coordinates.

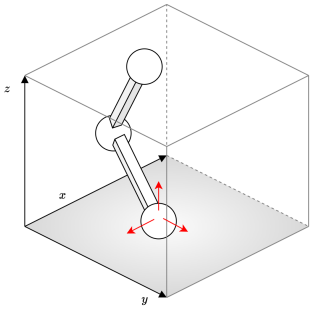

Figure 8. Degrees of freedom of the robot leg

5. Implementation

For programming we used an ESP32 Microcontroller, Platform.io and C++. In this section, we explain how we implemented the robot's movements and the user interface.

5.1 Servo Positioning and Calibration

For the controlling of the servos, we use the ESP32Servo library.[d] The servos on the left and right side are mirroring each other, so the right side moves clockwise, while the left side moves counterclockwise. For the default positions, we had to take into consideration, that servos are only able to rotate 180 degrees. For the right side, both knee and shoulder servos are set to a default value of 0 degrees, while on the left side the default value is 180 degrees. This allows the servos on both sides to perform movements in the same range.

A large amount of time was spent correctly setting the servo angles and the positions of the legs. The need for calibration was important, as the Inverse Kinematics calculations required the correct initial placement. An offset in this placement would cause the leg to move not correct or not at all. The reason for this time-consuming part was because each time the legs needed to be recalibrated, the legs needed to be disassembled, adjusted and retested.

5.2 Movement

In this section, we describe some basic movements that the robot is capable of and how these movements are implemented. Figure 9 depicts the general architecture.

d. https://github.com/madhephaestus/ESP32Servo

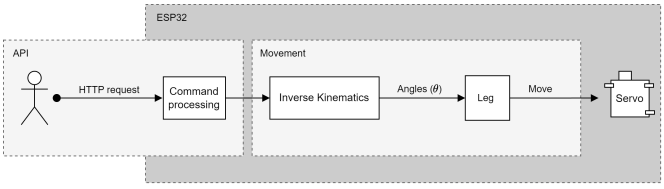

Figure 9. Code Architecture

5.2.1 Standing up

The robot can stand up by changing the z coordinate. The angles for this movement are calculated as described in chapter 4. With our default angles, the robot can go from 6 to 19 centimeters in height.

Standing up turned out to be harder than anticipated, as it not only depends on the correctly programmed movement but also on the stability and mechanical movement of the robot. During development, this caused a lot of problems due to the high tension and weight on the knee belts. A too quick movement of the legs would cause the knee to buckle under its own weight and would cause the robot to fall into itself. To solve this, we adjusted the movement to be incremental, such that the robot slowly moves upwards instead of one quick motion.

5.2.2 Tilting

To move the robot at an angle α in the side view (see the right image in 10) we need to specify the wanted x, y and z coordinates for each leg and based on these coordinates, calculate the needed angles. The only coordinate that changes in this movement is the z coordinate, i.e. the height of the robot.

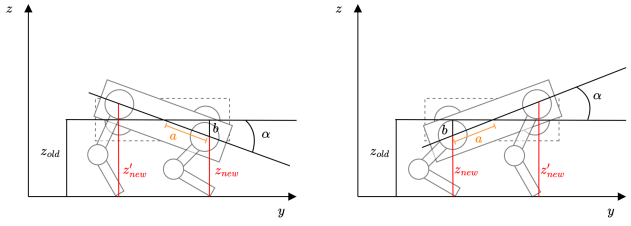

Figure 10. Side view of body tilting movement

$$
\begin{aligned}
b &= a\sin(\alpha) \\
\Rightarrow z_{new} &= z_{old} - b \\
\Rightarrow z'_{new} &= z_{old} + b
\end{aligned}
$$

We can then calculate the new angles for each leg. For the hind leg, the angle calculation becomes $(\theta'_h, \theta'_s, \theta'_w) = F_\theta(x, y, z_{new})$ and the front legs $(\theta'_h, \theta'_s, \theta'_w) = F_\theta(x, y, z'_{new})$.

5.2.3 Walking

To be able to walk, the robot needs to not only move the leg forward, but lift it up from the ground. For the

walking motion, we implement a parametrized curve.

$$\gamma_1(t) = \begin{pmatrix} t \\ 0.05t^2 \end{pmatrix}; \quad -3 \leq t < 3$$
$$\gamma_2(t) = \begin{pmatrix} t-6 \\ 1.281\cos(t-6) + 1.718 \end{pmatrix}; \quad 3 \leq t \leq 9$$

The curves $\gamma_1(t)$ and $\gamma_2(t)$ then together define the walking motion of the robot, which can be seen in figure 11. The legs move along this curve clockwise. This curve was discretized, and the robot leg moves along the points in a certain timeframe. The leg starts at $y = 0$, moves clockwise and completes the movement in one iteration.

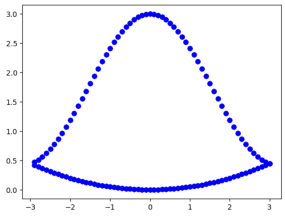

Figure 11. Parametrized curve of walking motion

It is important to note, that while the walking motion has been tested and works, there are still issues with the robot holding itself up due to improper balance.

5.3 User Interface

The robot's control interface is implemented through a WiFi-enabled web server running on the ESP32 microcontroller. The system creates a wireless access point that allows for remote control of the robot's movements via HTTP requests. Multiple commands can be chained together using semicolon separators, enabling complex movement sequences. The system implements automatic Wi-Fi reconnection and error handling to ensure reliable operation during extended use.

Table 1. Command Table

Command	Value	Function
z	6-19 (cm)	Move vertically
y	-19-19 (cm)	Move forward/backward
bt	Angle	Tilt backward
ft	Angle	Tilt forward
w	Steps	Walk x steps forward

6. Conclusion

Our team successfully tackled an ambitious and engaging project. We gained valuable knowledge in quadruped robotics, electronics, and servo programming. Despite various challenges, we successfully completed our project and created a robot capable of executing basic commands, including walking.

6.1 Future Work

For future iterations of this project, several improvements could be implemented:
Power Supply: It would be beneficial to replace the current setup with a rechargeable battery for improved portability.
Mechanical Design: The printed parts could be revised to improve stability, and optimizing weight distribution would enhance the robot's performance. Additionally, upgrading to wider belts with better tension mechanisms would support the robot's weight more effectively.
Software: Implementing additional movement commands would expand the robot's capabilities and allow for more complex actions.
Project Management: Developing a more detailed project timeline, with careful consideration of dependencies between phases, would improve the project flow.

6.2 Division of Labor

Together we started planning the project, researched the topic, and decided on the required components. We distributed the work evenly, with each team member having specific areas of responsibility as outlined below. All team members contributed to the robot's assembly and testing phases.
Anastasiya: PCB soldering, Wi-Fi interface
Jiri: 3D model design and development
Mathieu: Inverse kinematics calculations, leg movement programming
Rahel: Robot leg movement programming, Calibration

Appendix

Images

All figures depicted in this project report were self made. The 3D models are screenshots from during the design model. All other graphs were made using draw.io.

Material

All the most relevant materials used in the project can be found below

1. ESP32 WROOM 32 [Link]
2. ESP32 Servo Control PCB [Link]
3. Servo motors [Link]
4. Timing Belt [Link]
5. 3D fillament [Link]
6. Nylon string
7. Screws
8. Electrical parts (Resistors, Capacitors, Cables, Headers)

Code

All the relevant code can be found on the github repository.[e] All code was self-written, and ChatGPT was used as an additional tool to help with optimization and debugging of the code.

e. https://github.com/clonder/roboRetriever