Faculty of Computers and Artificial Intelligence

Cairo University

Q1 mini Lite3

Simple Quadruped Robot

Project Report

Names	IDs
Ali Hisham Farouq	20200335
Mohammed Ahmed	20200693
Sahar Hamdi Abdulhafeez	20201089
Rana Abdulsalam	20200752
Hager Maher	20200615
Mario Guirguis	20200405

Abstract

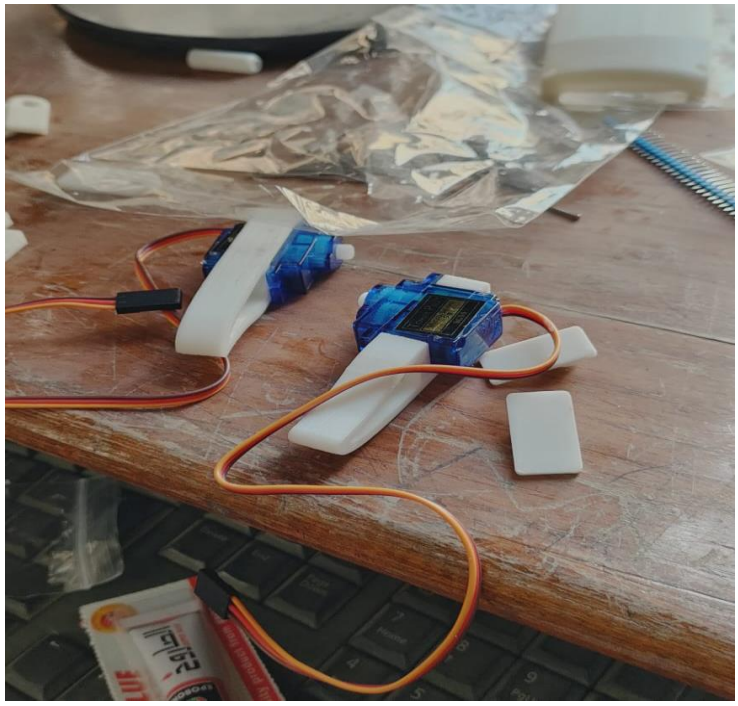
The Q1 Mini Quadruped Robot 2.0, developed by Jason Workshop, represents a significant advancement in the field of versatile and agile robotics. This compact quadrupedal robot, designed with inspiration from spiders, boasts enhanced mobility and stability. The primary goal behind its creation is to address the critical challenges posed by earthquakes and war-torn areas, where traditional means of intervention are often limited.

Through rigorous testing and simulations, the Q1 Mini Quadruped Robot 2.0 has demonstrated exceptional adaptability to challenging terrains, making it an invaluable asset in disaster response and military applications. Its spider-like design allows the robot to traverse uneven surfaces with ease, accessing areas that may be inaccessible to larger, conventional robots. Equipped with advanced sensors and communication systems, the Q1 Mini Quadruped Robot 2.0 has proven effective in remote reconnaissance and data collection tasks.

The results indicate that this innovative robot can play a crucial role in providing real-time information in disaster-stricken regions and enhancing situational awareness in military operations. The Q1 Mini Quadruped Robot 2.0 stands as a testament to the capabilities of bio-inspired robotics in addressing complex challenges, with potential applications extending beyond disaster response and military scenarios to various fields requiring agile and adaptable robotic solutions.

Background

1. **Leg Mechanism:**
    - The robot features four articulated legs, each equipped with 2 high-torque servo motors, enabling precise control of leg movement.
    - Multiple joints in each leg mimic the flexibility of spider limbs, providing adaptability to varied terrains.



2. **Chassis and Frame:**
    - The chassis is constructed from lightweight yet durable materials, ensuring structural integrity while minimizing overall weight.
    - The frame design contributes to the robot's stability and ability to withstand harsh conditions.

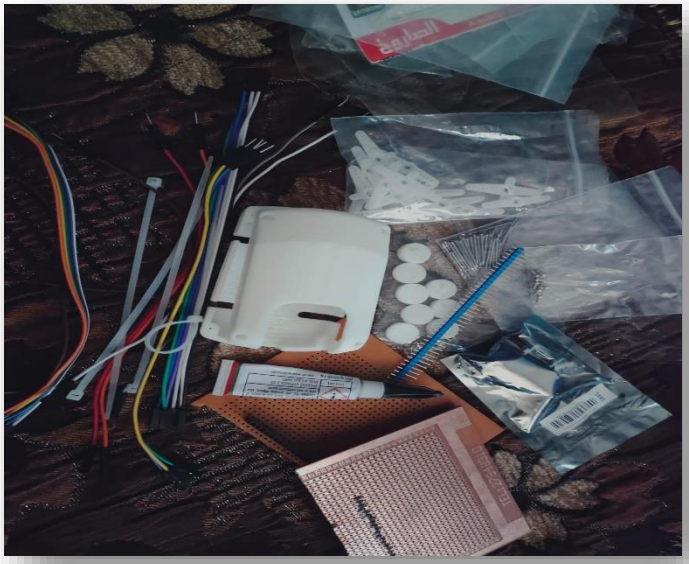
3. **Communication Module:**
    - A communication module facilitates remote operation and data exchange.

    - **ESP32:**
        - **Usage:** The ESP32 module serves as a communication bridge between the robot and external

devices or control systems. It enables wireless communication, allowing operators to remotely control the robot and receive real-time data.

- **Functionality:** The ESP32 module can facilitate tasks such as sending commands for movement, receiving sensor data, and establishing a communication link for mission-specific requirements.

- **Driver:**
  - Connect the output pins from the robot's ESP32 to the input pins of the drive system for effective coordination.
  - Ensure proper power and ground connections between the microcontroller and the drive system.



- **Jump Wires:**
  - We use it to connect between Driver and ESP32.
  - flexibility during connections and modifications.

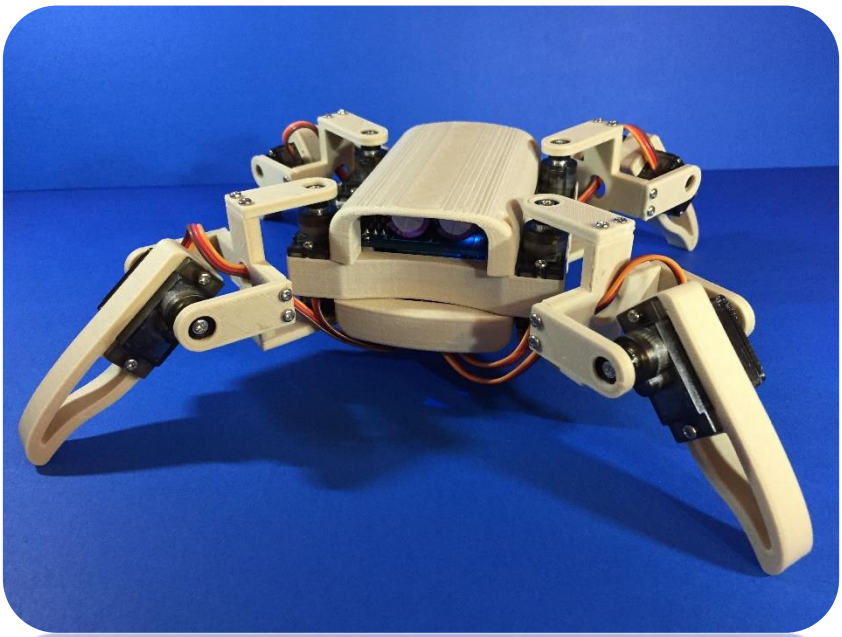
We Connected:

➔ Ground of ESP with the Ground of Driver.
➔ 5v from Driver with vcc (v+).
➔ SDA with pin 21.
➔ SCI with pin 22.

Proposed Idea

1- Design:

- The chassis and structural design of the robot are lightweight yet durable, allowing it to withstand harsh conditions while remaining agile.
- The combination of materials and design contributes to the overall stability and robustness of the robot.
- The integration of these elements in the Q1 Mini Quadruped Robot 2.0's mechanism enables it to emulate the agility and adaptability of spiders, making it well-suited for navigating challenging environments encountered in disaster response and military applications.



2- Mechanism:

- The Q1 Mini Quadruped Robot 2.0 employs a sophisticated and bio-inspired mechanism to achieve its agility and stability, drawing inspiration from the biomechanics of spiders. The primary components of its mechanism include:

- **Leg Configuration:**
  - The robot features four legs, each with multiple joints that mimic the flexibility and range of motion found in arachnid limbs.
  - These joints are actuated by powerful motors, allowing for precise and coordinated movement.

- **Spider-Inspired Gait:**
  - The robot utilizes a spider-inspired walking gait, enabling it to navigate diverse terrains with efficiency.
  - This gait involves alternating tripod and tetrapod configurations, providing stability while moving and adapting to changes in the environment.

- **Communication System:**
  - The robot is equipped with a communication system that facilitates remote control and data transmission.
  - This system enables operators to guide the robot in various tasks, such as reconnaissance, and receive real-time information from its sensors.

Software

We used the Arduino IDE to write our code using the C++ programming language. Here are the important parts of our code and their explanation:

1- We downloaded the important libraries and included them in the IDE:

- **DabbleESP32.h →** The Application we used to control the robot motion.
- **Adafruit →** is the type of our Driver.

```
#include <DabbleESP32.h>
#define CUSTOM_SETTINGS
#define INCLUDE_GAMEPAD_MODULE

#include <ESP32Servo.h>
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
```

2- Setup() Function → sets up serial communication, initializes the PWM module for controlling servo motors, starts Dabble communication, and introduces a short delay for stability. This initialization code typically runs once when the microcontroller starts or is reset.

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200); //Baud rate
  pwm.begin();
  pwm.setPWMFreq(60); //Servo motors work at 60hz refresh rate
  Dabble.begin("ESP32ServoController");
  delay(100);

}
```

3- Loop() Function → main execution loop of an Arduino sketch for an ESP32 microcontroller:

- **Dabble.processInput():** This line processes any input received from the Dabble app, which is a mobile application used to control and communicate with the ESP32 wirelessly.

- The following series of if and else if statements check the state of various buttons on the GamePad module of the Dabble app:

  - **GamePad.isUpPressed():** If the "Up" button is pressed, the forward() function is called, and then there is a delay of 100 milliseconds.
  - **GamePad.isDownPressed():** If the "Down" button is pressed, the backward() function is called.
  - **GamePad.isRightPressed():** If the "Right" button is pressed, the right() function is called.
  - **GamePad.isLeftPressed():** If the "Left" button is pressed, the left() function is called.
  - **GamePad.isCirclePressed():** If the "Circle" button is pressed, the dance() function is called three times (dancing sequence).
  - **GamePad.isSquarePressed():** If the "Square" button is pressed, the hi() function is called, followed by a delay.
  - **GamePad.isTrianglePressed():** If the "Triangle" button is pressed, the slep() function is called, followed by a delay.

- **GamePad.isCrossPressed():** If the "Cross" button is pressed, the wake() function is called, followed by a delay.

```
void loop() {
  Dabble.processInput();
  if (GamePad.isUpPressed()) {
        forward();
        delay(100);
  }else if (GamePad.isDownPressed()) {
        backward();
        delay(100);
  }else if(GamePad.isRightPressed()){
        right();
        delay(100);
  }else if(GamePad.isLeftPressed()){
        left();
        delay(100);
  }else if(GamePad.isCirclePressed()){
        dance();
        dance();
        dance();
        delay(100);
  }else if(GamePad.isSquarePressed()){
        hi();
        delay(100);
  }else if(GamePad.isTrianglePressed()){
        slep();
        delay(100);
  }else if(GamePad.isCrossPressed()){
        wake();
        delay(100);
  }
```

4- Setangle() Function → designed to simplify the process of setting the angle of a servo motor connected to the Adafruit_PWMServoDriver by providing the servo number and the desired angle in degrees. The function calculates the corresponding pulse width value using the map function and then sets the servo's pulse width using the setPWM method of the PWM driver.

```
void setangle(int servonum , int angle)
{

  int pulse = map(angle , 0, 180, 150, 600);

  pwm.setPWM(servonum, 0, pulse);
}
```

5- Forward() Function → For Forward Motion.

```
void forward() {



  setangle(12, 40); //leg up
  setangle(3, 140); //leg up
  delay(100);
  setangle(13, 90); // move leg                                  //leg2,3
  setangle(2, 0); // move leg
  delay(100);
  setangle(12, 0);// leg back on ground
  setangle(3, 180);// leg back on ground
  delay(100);

  setangle(0, 40);
  setangle(15, 50);
  delay(100);
  setangle(1, 180);    //leg1,4
  setangle(14, 60);
  delay(100);
  setangle(0, 0);
  setangle(15, 90);
  delay(100);
```

6- Backward() Function → for Backward Motion.

```
}
void backward() {



 setangle(12, 40);
  setangle(3, 140);

                                  //leg2,3
  setangle(2, 90);
  delay(100);
  setangle(13, 0);
  delay(100);

  setangle(3, 180);
  delay(100);
  setangle(12, 0);
  delay(100);

  setangle(0, 40);
  delay(100);
  setangle(15, 50);
  delay(100);
  setangle(1, 90);    //leg1,4
  setangle(14, 180);
  delay(100);
  setangle(0, 0);
  setangle(15, 90);
  delay(100);
```

7- Hi(), Sleep(), and Wake() Functions →

```
}
void hi(){
  setangle(12,180);
  delay(100);
  setangle(13,90);
  delay(100);
  setangle(13,0);
  delay(100);
   setangle(13,90);
  delay(100);
  setangle(13,0);
  delay(100);
   setangle(13,90);
  delay(100);
  setangle(13,0);
  delay(100);
  setangle(12,0);
  setangle(13,45);

}
void slep(){

  setangle(12, 90);
  setangle(0, 90);
  delay(100);
  setangle(3, 90);
  setangle(15, 0);
  delay(100);
}
void wake(){
  setangle(12, 0);
  setangle(0, 0);
  delay(100);
  setangle(3, 180);
  setangle(15, 90);
}
```

8- Right() Function → Right Motion

```
void right() {


  setangle(12,4);
  setangle(3,140);
  delay(100);
  setangle(13,90);
  setangle(2,0);           //leg 3,2
  delay(100);
  setangle(12,0);
  setangle(3,180);
  delay(100);

  setangle(0,40);
  setangle(15,50);
  delay(100);             //leg 1,4
  setangle(1,90);
  setangle(14,180);
  delay(100);
  setangle(0,0);
  setangle(15,90);
  delay(100);
```

9- Left() Function → For Left Motion.

```
void left(){
  setangle(3,140);
  setangle(12,40);
  delay(100);
  setangle(2,90);
  setangle(13,0);
  delay(100);
  setangle(3,180);
  setangle(12,0);
  delay(100);


  setangle(0,40);
  setangle(15,50);
  delay(100);
  setangle(1,180);
  setangle(14,60);
  delay(100);
  setangle(0,0);
  setangle(15 ,90);
  delay(100);



  setangle(2, 45);
  setangle(14, 115);
  setangle(1, 135);
  setangle(13, 45);
  delay(100);
}
```

Application

We used a Mobile App called Dabble to control robot's Motion and Communicate with ESP32 Wirelessly.

https://play.google.com/store/apps/details?id=io.dabbleapp