

# Quadruped Project Report

---

Author:

Divyansh Garg

# Table of contents

---

Abstract	3
Introduction	4
Mechanical structure	6
Electrical components	6-7
Working	8
Circuit	9
Algorithm	10-11
Codes	12-32
Images	33
Conclusion	34

# Abstract

---

Nowadays, design, development, and motion planning of a mobile robot explore research areas in the field of robotics. Mobile robots have an extensive area of applications in various fields like space exploration, military application, industrial use, and many more. Hence, the design and development of a mobile robot is a crucial part of the above application. Among all the mobile robot, the quadrupedal robot is a legged robot, which is superior to wheeled and tracked robot due to its potential to explore in all the terrain like the human and animal. In this paper, the survey concentrates on various design and development approaches for the quadrupedal robot, and environment perception techniques are discussed. Four-legged robots can have very sophisticated locomotion patterns and provide means of navigating on surfaces where it seems impossible for wheeled robots. Part of the earth's landmass may be inaccessible by humans due to their geographical locations, environmental hazards, etc. The objective of this project is to develop a reliable solution that enables the implementation of stable and fast static/dynamic walking on even and uneven terrain. The robot captures/mimics the mobility, autonomy and speed of four-legged living creatures

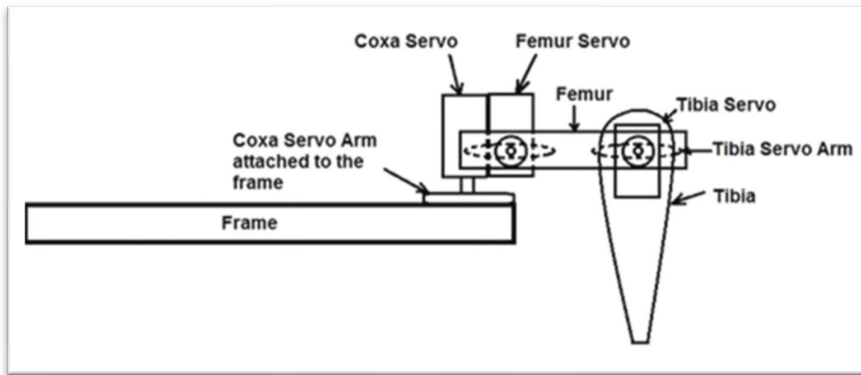
# Introduction

---

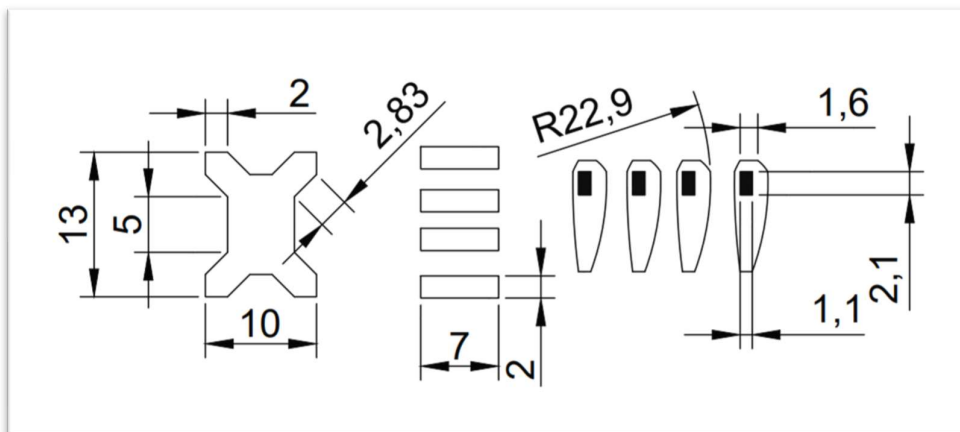
In the last three decades, the mobile robot has made much attention because of exploring in the complex environment, space, rescue operation, accomplishing a task without human effort, etc. The mobile robot can be broadly classified into three categories; wheeled robot, tracked robot, and legged robot . The robot locomotion system is an essential characteristic of mobile design, and it depends not only on working space but also on a technical measure like manoeuvrability, controllability, terrain condition, efficiency, and stability. Each system has its own advantages and disadvantages . The Development of terrestrial locomotion of legged robot has been continuously grown over the few decades because of more advantages than wheel robot vehicles.

# Mechanical Structure

The mechanical structure of quadruped usually consists of legs and a main frame. Legs are divided into three parts coxa, Femur and Tibia. The mechanical frame is a structural frame or chassis that supports the mechanical components and provides attachments point for motors, sensors, and other electrical components.



The structure of the foot varies depending on the species, that provides support, absorbs shock, during locomotion. It maintains stability and balance while moving by having low centre of gravity and wide stance.



# Electrical Components

---

## **Arduino Nano:**

The Arduino Nano is a compact and versatile microcontroller board based on the ATmega328P microcontroller chip. Due to its small size, ease of use, and versatility, the Arduino Nano is frequently utilised in quadruped robots. Its compact form factor makes it possible to integrate it inside the body of the robot in confined locations, and its many digital and analogue I/O pins make it possible to control the robot's many actuators, sensors, and motors. For prototyping and developing quadruped robots, the Arduino Nano is a preferred option due to its extensive libraries, robust community support, and compatibility with a wide range of sensors and peripherals. This allows for rapid iteration and experimentation in the design and functionality of the robots.

## **SG-90 Servo Motor:**

In all 12 servo motors are used 3 for each leg. Because of its small size, lightweight construction, and affordable price, the SG-90 servo motor is frequently utilised in quadruped robots. This makes it an excellent choice for applications requiring several motors. Its precision control, which has a 180-degree rotation

range, enables the robot to position its limbs precisely, which promotes fluid and synchronised movement. Furthermore, the SG-90's simplicity of integration and programming into quadruped robot designs are facilitated by its interoperability with microcontrollers such as Arduino.

### **power supply:**

12volt 2 ampere power supply is used to give power. Each motor consumes 4.8v and 100-200 mA based on the load. The input voltage of Arduino nano varies from 6v-20v. In particular operation it consumes a voltage of around 7.6v and a current of around 22mA.

### **Capacitor:**

24 microfarad capacitor is used for short term energy storage and to provide a constant and stable and backup power supply to the Arduino nano disabling its auto reset function and allowing the program to run fully

### **Light emitting Diode:**

L.E.D is used as power on and off indicator for the quadruped showing whether the robot is operating or shut down.

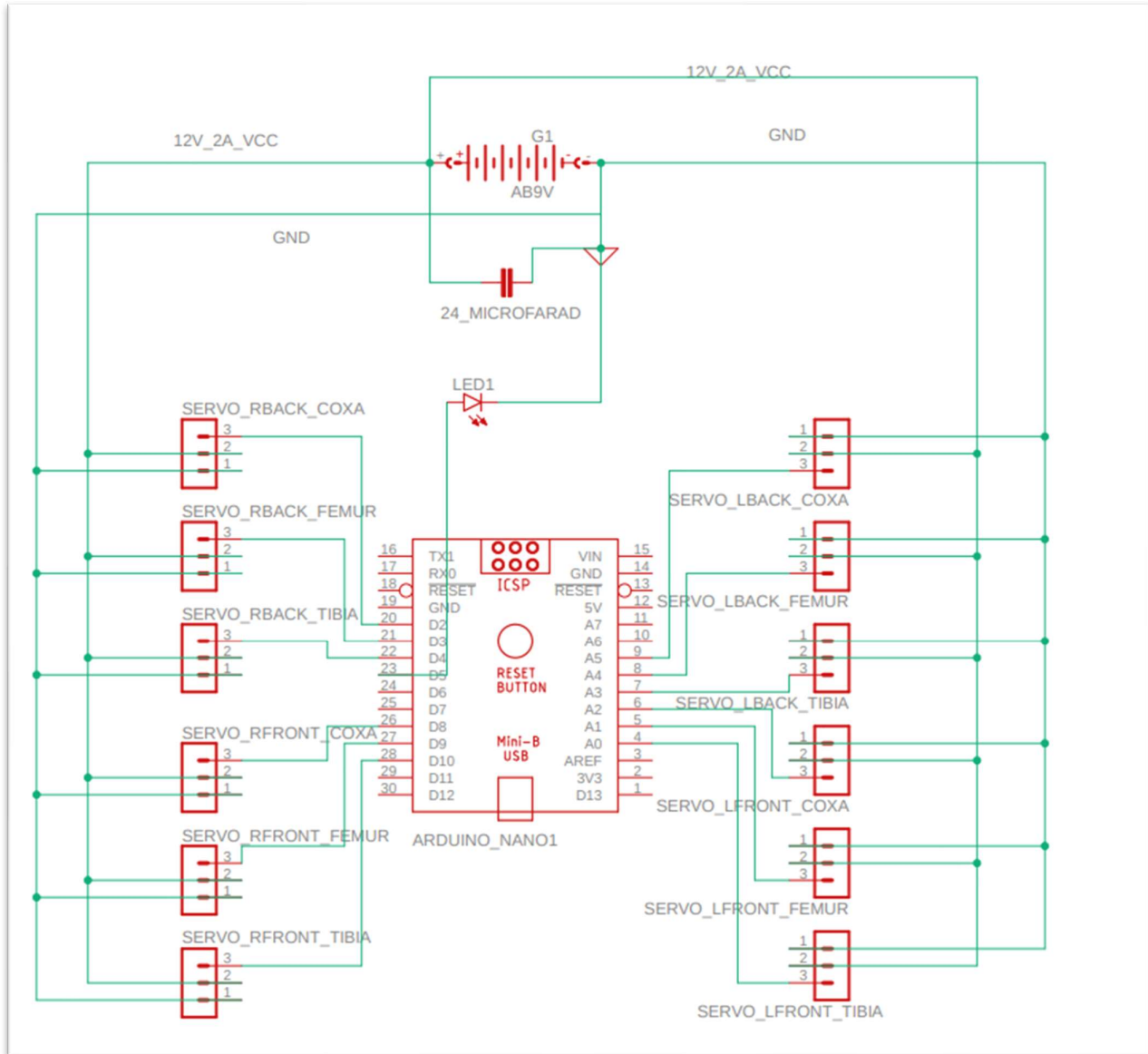
# Working Principle

---

Each limb on the robot consists of three parts called Tibia , femur and coxa. Each of the part is controlled by a servo motor with Arduino as their brain. With proper codes a specific part of a specific limb is moved as er desired condition. The robot's control system generates sequence of leg moments known as gaits. These gaits are designed to achieve desired locomotion patterns, such as walking, climbing etc. Actuators, typically electric motors or servos, execute these leg movements based on commands from the control system. In addition to locomotion, quadruped robots may incorporate navigation algorithms to plan paths through complex environments and avoid obstacle.



# Circuit



# Algorithm

---

## **Initialization:**

The setup function initializes parameters, sets up serial communication, attaches servos, and starts the servo service. The servo\_attach function attaches servo motors to their respective pins, while the servo\_detach function detaches them.

## **Movement Functions:**


Several functions are defined for various movements such as sitting, standing, turning left/right, stepping forward/backward, hand waving, and hand shaking. These functions adjust the positions of the leg end-points to achieve the desired movement.

## **Servo Service:**

The servo\_service function is a timer interrupt service routine that continuously updates the positions of servo motors based on the expected positions of leg end-points. It calculates the required servo angles based on the current and expected coordinates of the leg end-points using inverse kinematics.

## **Coordinate Transformation:**

The cartesian\_to\_polar function converts Cartesian coordinates (x, y, z) to polar coordinates (alpha, beta, gamma), which represent the servo angles required to position the leg end-



points. The `polar_to_servo` function then maps these angles to the corresponding servo positions.

### **Main Loop:**

The loop function repeatedly executes a sequence of predefined movements, such as standing, stepping, turning, and hand gestures, with delays in between each movement.

# Codes

---

## Leg Initialisation:

```
#include <Servo.h>

Servo servo[4][3];

const int servo_pin[4][3] = { {2, 3, 4}, {A5, A4, A3}, {8, 9, 10}, {A0, A2, A1} };

void setup()
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            servo[i][j].attach(servo_pin[i][j]);
            delay(20);
        }
    }
}

void loop(void)
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            servo[i][j].write(90);
            delay(20);
        }
    }
}
```

```
}  
}
```

## Quadruped movement:

```
#include <Servo.h>  
#include <FlexiTimer2.h>  
  
Servo servo[4][3];  
  
const int servo_pin[4][3] = { {2, 3, 4}, {A5,A4, A3}, {8, 9, 10}, {A0, A2, A1} };  
  
const float length_a = 55;  
const float length_b = 77.5;  
const float length_c = 27.5;  
const float length_side = 71;  
const float z_absolute = -28;  
const float z_default = -50, z_up = -30, z_boot = z_absolute;  
const float x_default = 62, x_offset = 0;  
const float y_start = 0, y_step = 40;  
volatile float site_now[4][3];  
volatile float site_expect[4][3];  
float temp_speed[4][3];  
float move_speed;  
float speed_multiple = 1;  
const float spot_turn_speed = 4;  
const float leg_move_speed = 8;  
const float body_move_speed = 3;  
const float stand_seat_speed = 1;  
volatile int rest_counter;  
const float KEEP = 255;  
const float pi = 3.1415926;
```

```

const float temp_a = sqrt(pow(2 * x_default + length_side, 2) + pow(y_step,
2));
const float temp_b = 2 * (y_start + y_step) + length_side;
const float temp_c = sqrt(pow(2 * x_default + length_side, 2) + pow(2 * y_start
+ y_step + length_side, 2));
const float temp_alpha = acos((pow(temp_a, 2) + pow(temp_b, 2) -
pow(temp_c, 2)) / 2 / temp_a / temp_b);
const float turn_x1 = (temp_a - length_side) / 2;
const float turn_y1 = y_start + y_step / 2;
const float turn_x0 = turn_x1 - temp_b * cos(temp_alpha);
const float turn_y0 = temp_b * sin(temp_alpha) - turn_y1 - length_side;
void setup()
{
  Serial.begin(115200);
  Serial.println("Robot starts initialization");
  set_site(0, x_default - x_offset, y_start + y_step, z_boot);
  set_site(1, x_default - x_offset, y_start + y_step, z_boot);
  set_site(2, x_default + x_offset, y_start, z_boot);
  set_site(3, x_default + x_offset, y_start, z_boot);
  for (int i = 0; i < 4; i++)
  {
    for (int j = 0; j < 3; j++)
    {
      site_now[i][j] = site_expect[i][j];
    }
  }
  FlexiTimer2::set(20, servo_service);
  FlexiTimer2::start();
  Serial.println("Servo service started");
}

```

```
servo_attach();  
Serial.println("Servos initialized");  
Serial.println("Robot initialization Complete");  
}  
void servo_attach(void)  
{  
  for (int i = 0; i < 4; i++)  
  {  
    for (int j = 0; j < 3; j++)  
    {  
      servo[i][j].attach(servo_pin[i][j]);  
      delay(100);  
    }  
  }  
}  
void servo_detach(void)  
{  
  for (int i = 0; i < 4; i++)  
  {  
    for (int j = 0; j < 3; j++)  
    {  
      servo[i][j].detach();  
      delay(100);  
    }  
  }  
}  
void loop()  
{
```

```
Serial.println("Stand");
stand();
delay(2000);
Serial.println("Step forward");
step_forward(5);
delay(2000);
Serial.println("Step back");
step_back(5);
delay(2000);
Serial.println("Turn left");
turn_left(5);
delay(2000);
Serial.println("Turn right");
turn_right(5);
delay(2000);
Serial.println("Hand wave");
hand_wave(3);
delay(2000);
Serial.println("Hand wave");
hand_shake(3);
delay(2000);
Serial.println("Sit");
sit();
delay(5000);
}

void sit(void)
{
    move_speed = stand_seat_speed;
```



```

for (int leg = 0; leg < 4; leg++)
{
    set_site(leg, KEEP, KEEP, z_boot);
}
wait_all_reach();
}

void stand(void)
{
    move_speed = stand_seat_speed;
    for (int leg = 0; leg < 4; leg++)
    {
        set_site(leg, KEEP, KEEP, z_default);
    }
    wait_all_reach();
}

void turn_left(unsigned int step)
{
    move_speed = spot_turn_speed;
    while (step-- > 0)
    {
        if (site_now[3][1] == y_start)
        {
            //leg 3&1 move
            set_site(3, x_default + x_offset, y_start, z_up);
            wait_all_reach();
            set_site(0, turn_x1 - x_offset, turn_y1, z_default);
            set_site(1, turn_x0 - x_offset, turn_y0, z_default);
            set_site(2, turn_x1 + x_offset, turn_y1, z_default);

```

```
set_site(3, turn_x0 + x_offset, turn_y0, z_up);
wait_all_reach();
set_site(3, turn_x0 + x_offset, turn_y0, z_default);
wait_all_reach();
set_site(0, turn_x1 + x_offset, turn_y1, z_default);
set_site(1, turn_x0 + x_offset, turn_y0, z_default);
set_site(2, turn_x1 - x_offset, turn_y1, z_default);
set_site(3, turn_x0 - x_offset, turn_y0, z_default);
wait_all_reach();
set_site(1, turn_x0 + x_offset, turn_y0, z_up);
wait_all_reach();
set_site(0, x_default + x_offset, y_start, z_default);
set_site(1, x_default + x_offset, y_start, z_up);
set_site(2, x_default - x_offset, y_start + y_step, z_default);
set_site(3, x_default - x_offset, y_start + y_step, z_default);
wait_all_reach();
set_site(1, x_default + x_offset, y_start, z_default);
wait_all_reach();
}
else
{
    set_site(0, x_default + x_offset, y_start, z_up);
    wait_all_reach();
    set_site(0, turn_x0 + x_offset, turn_y0, z_up);
    set_site(1, turn_x1 + x_offset, turn_y1, z_default);
    set_site(2, turn_x0 - x_offset, turn_y0, z_default);
    set_site(3, turn_x1 - x_offset, turn_y1, z_default);
    wait_all_reach();
}
```

```

    set_site(0, turn_x0 + x_offset, turn_y0, z_default);
    wait_all_reach();
    set_site(0, turn_x0 - x_offset, turn_y0, z_default);
    set_site(1, turn_x1 - x_offset, turn_y1, z_default);
    set_site(2, turn_x0 + x_offset, turn_y0, z_default);
    set_site(3, turn_x1 + x_offset, turn_y1, z_default);
    wait_all_reach();
    set_site(2, turn_x0 + x_offset, turn_y0, z_up);
    wait_all_reach();
    set_site(0, x_default - x_offset, y_start + y_step, z_default);
    set_site(1, x_default - x_offset, y_start + y_step, z_default);
    set_site(2, x_default + x_offset, y_start, z_up);
    set_site(3, x_default + x_offset, y_start, z_default);
    wait_all_reach();
    set_site(2, x_default + x_offset, y_start, z_default);
    wait_all_reach();
}
}
}

void turn_right(unsigned int step)
{
    move_speed = spot_turn_speed;
    while (step-- > 0)
    {
        if (site_now[2][1] == y_start)
        {
            //leg 2&0 move
            set_site(2, x_default + x_offset, y_start, z_up);

```

```
wait_all_reach();
set_site(0, turn_x0 - x_offset, turn_y0, z_default);
set_site(1, turn_x1 - x_offset, turn_y1, z_default);
set_site(2, turn_x0 + x_offset, turn_y0, z_up);
set_site(3, turn_x1 + x_offset, turn_y1, z_default);
wait_all_reach();
set_site(2, turn_x0 + x_offset, turn_y0, z_default);
wait_all_reach();
set_site(0, turn_x0 + x_offset, turn_y0, z_default);
set_site(1, turn_x1 + x_offset, turn_y1, z_default);
set_site(2, turn_x0 - x_offset, turn_y0, z_default);
set_site(3, turn_x1 - x_offset, turn_y1, z_default);
wait_all_reach();
set_site(0, turn_x0 + x_offset, turn_y0, z_up);
wait_all_reach();
```

```
set_site(0, x_default + x_offset, y_start, z_up);
set_site(1, x_default + x_offset, y_start, z_default);
set_site(2, x_default - x_offset, y_start + y_step, z_default);
set_site(3, x_default - x_offset, y_start + y_step, z_default);
wait_all_reach();
set_site(0, x_default + x_offset, y_start, z_default);
wait_all_reach();
```

```
}
```

```
else
```

```
{
```

```
set_site(1, x_default + x_offset, y_start, z_up);
wait_all_reach();
```

```

    set_site(0, turn_x1 + x_offset, turn_y1, z_default);
    set_site(1, turn_x0 + x_offset, turn_y0, z_up);
    set_site(2, turn_x1 - x_offset, turn_y1, z_default);
    set_site(3, turn_x0 - x_offset, turn_y0, z_default);
    wait_all_reach();
    set_site(1, turn_x0 + x_offset, turn_y0, z_default);
    wait_all_reach();
    set_site(0, turn_x1 - x_offset, turn_y1, z_default);
    set_site(1, turn_x0 - x_offset, turn_y0, z_default);
    set_site(2, turn_x1 + x_offset, turn_y1, z_default);
    set_site(3, turn_x0 + x_offset, turn_y0, z_default);
    wait_all_reach();
    set_site(3, turn_x0 + x_offset, turn_y0, z_up);
    wait_all_reach();
    set_site(0, x_default - x_offset, y_start + y_step, z_default);
    set_site(1, x_default - x_offset, y_start + y_step, z_default);
    set_site(2, x_default + x_offset, y_start, z_default);
    set_site(3, x_default + x_offset, y_start, z_up);
    wait_all_reach();
    set_site(3, x_default + x_offset, y_start, z_default);
    wait_all_reach();
}
}
}

void step_forward(unsigned int step)
{
    move_speed = leg_move_speed;
    while (step-- > 0)

```

```
{
  if (site_now[2][1] == y_start)
  {
    set_site(2, x_default + x_offset, y_start, z_up);
    wait_all_reach();
    set_site(2, x_default + x_offset, y_start + 2 * y_step, z_up);
    wait_all_reach();
    set_site(2, x_default + x_offset, y_start + 2 * y_step, z_default);
    wait_all_reach();
    move_speed = body_move_speed;
    set_site(0, x_default + x_offset, y_start, z_default);
    set_site(1, x_default + x_offset, y_start + 2 * y_step, z_default);
    set_site(2, x_default - x_offset, y_start + y_step, z_default);
    set_site(3, x_default - x_offset, y_start + y_step, z_default);
    wait_all_reach();
    move_speed = leg_move_speed;
    set_site(1, x_default + x_offset, y_start + 2 * y_step, z_up);
    wait_all_reach();
    set_site(1, x_default + x_offset, y_start, z_up);
    wait_all_reach();
    set_site(1, x_default + x_offset, y_start, z_default);
    wait_all_reach();
  }
  else
  {
    set_site(0, x_default + x_offset, y_start, z_up);
    wait_all_reach();
    set_site(0, x_default + x_offset, y_start + 2 * y_step, z_up);
```

```

wait_all_reach();
set_site(0, x_default + x_offset, y_start + 2 * y_step, z_default);
wait_all_reach();
move_speed = body_move_speed;
set_site(0, x_default - x_offset, y_start + y_step, z_default);
set_site(1, x_default - x_offset, y_start + y_step, z_default);
set_site(2, x_default + x_offset, y_start, z_default);
set_site(3, x_default + x_offset, y_start + 2 * y_step, z_default);
wait_all_reach();
move_speed = leg_move_speed;
set_site(3, x_default + x_offset, y_start + 2 * y_step, z_up);
wait_all_reach();
set_site(3, x_default + x_offset, y_start, z_up);
wait_all_reach();
set_site(3, x_default + x_offset, y_start, z_default);
wait_all_reach();
}
}
}
void step_back(unsigned int step)
{
    move_speed = leg_move_speed;
    while (step-- > 0)
    {
        if (site_now[3][1] == y_start)
        {
            set_site(3, x_default + x_offset, y_start, z_up);
            wait_all_reach();

```

```
set_site(3, x_default + x_offset, y_start + 2 * y_step, z_up);
wait_all_reach();
set_site(3, x_default + x_offset, y_start + 2 * y_step, z_default);
wait_all_reach();
move_speed = body_move_speed;
set_site(0, x_default + x_offset, y_start + 2 * y_step, z_default);
set_site(1, x_default + x_offset, y_start, z_default);
set_site(2, x_default - x_offset, y_start + y_step, z_default);
set_site(3, x_default - x_offset, y_start + y_step, z_default);
wait_all_reach();
move_speed = leg_move_speed;
set_site(0, x_default + x_offset, y_start + 2 * y_step, z_up);
wait_all_reach();
set_site(0, x_default + x_offset, y_start, z_up);
wait_all_reach();
set_site(0, x_default + x_offset, y_start, z_default);
wait_all_reach();
}
else
{
    set_site(1, x_default + x_offset, y_start, z_up);
    wait_all_reach();
    set_site(1, x_default + x_offset, y_start + 2 * y_step, z_up);
    wait_all_reach();
    set_site(1, x_default + x_offset, y_start + 2 * y_step, z_default);
    wait_all_reach();
    move_speed = body_move_speed;
    set_site(0, x_default - x_offset, y_start + y_step, z_default);
```



```

    set_site(1, x_default - x_offset, y_start + y_step, z_default);
    set_site(2, x_default + x_offset, y_start + 2 * y_step, z_default);
    set_site(3, x_default + x_offset, y_start, z_default);
    wait_all_reach();
    move_speed = leg_move_speed;
    set_site(2, x_default + x_offset, y_start + 2 * y_step, z_up);
    wait_all_reach();
    set_site(2, x_default + x_offset, y_start, z_up);
    wait_all_reach();
    set_site(2, x_default + x_offset, y_start, z_default);
    wait_all_reach();
}
}
}
void body_left(int i)
{
    set_site(0, site_now[0][0] + i, KEEP, KEEP);
    set_site(1, site_now[1][0] + i, KEEP, KEEP);
    set_site(2, site_now[2][0] - i, KEEP, KEEP);
    set_site(3, site_now[3][0] - i, KEEP, KEEP);
    wait_all_reach();
}
void body_right(int i)
{
    set_site(0, site_now[0][0] - i, KEEP, KEEP);
    set_site(1, site_now[1][0] - i, KEEP, KEEP);
    set_site(2, site_now[2][0] + i, KEEP, KEEP);
    set_site(3, site_now[3][0] + i, KEEP, KEEP);
}

```

```
    wait_all_reach();
}
void hand_wave(int i)
{
    float x_tmp;
    float y_tmp;
    float z_tmp;
    move_speed = 1;
    if (site_now[3][1] == y_start)
    {
        body_right(15);
        x_tmp = site_now[2][0];
        y_tmp = site_now[2][1];
        z_tmp = site_now[2][2];
        move_speed = body_move_speed;
        for (int j = 0; j < i; j++)
        {
            set_site(2, turn_x1, turn_y1, 50);
            wait_all_reach();
            set_site(2, turn_x0, turn_y0, 50);
            wait_all_reach();
        }
        set_site(2, x_tmp, y_tmp, z_tmp);
        wait_all_reach();
        move_speed = 1;
        body_left(15);
    }
    else
```

```

{
    body_left(15);
    x_tmp = site_now[0][0];
    y_tmp = site_now[0][1];
    z_tmp = site_now[0][2];
    move_speed = body_move_speed;
    for (int j = 0; j < i; j++)
    {
        set_site(0, turn_x1, turn_y1, 50);
        wait_all_reach();
        set_site(0, turn_x0, turn_y0, 50);
        wait_all_reach();
    }
    set_site(0, x_tmp, y_tmp, z_tmp);
    wait_all_reach();
    move_speed = 1;
    body_right(15);
}
}

void hand_shake(int i)
{
    float x_tmp;
    float y_tmp;
    float z_tmp;
    move_speed = 1;
    if (site_now[3][1] == y_start)
    {
        body_right(15);
    }
}

```

```
x_tmp = site_now[2][0];
y_tmp = site_now[2][1];
z_tmp = site_now[2][2];
move_speed = body_move_speed;
for (int j = 0; j < i; j++)
{
    set_site(2, x_default - 30, y_start + 2 * y_step, 55);
    wait_all_reach();
    set_site(2, x_default - 30, y_start + 2 * y_step, 10);
    wait_all_reach();
}
set_site(2, x_tmp, y_tmp, z_tmp);
wait_all_reach();
move_speed = 1;
body_left(15);
}
else
{
    body_left(15);
    x_tmp = site_now[0][0];
    y_tmp = site_now[0][1];
    z_tmp = site_now[0][2];
    move_speed = body_move_speed;
    for (int j = 0; j < i; j++)
    {
        set_site(0, x_default - 30, y_start + 2 * y_step, 55);
        wait_all_reach();
        set_site(0, x_default - 30, y_start + 2 * y_step, 10);
```

```

    wait_all_reach();
}
set_site(0, x_tmp, y_tmp, z_tmp);
wait_all_reach();
move_speed = 1;
body_right(15);
}
}
void servo_service(void)
{
    sei();
    static float alpha, beta, gamma;

    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (abs(site_now[i][j] - site_expect[i][j]) >= abs(temp_speed[i][j]))
                site_now[i][j] += temp_speed[i][j];
            else
                site_now[i][j] = site_expect[i][j];
        }
        cartesian_to_polar(alpha, beta, gamma, site_now[i][0], site_now[i][1],
site_now[i][2]);
        polar_to_servo(i, alpha, beta, gamma);
    }
    rest_counter++;
}

```

```

void set_site(int leg, float x, float y, float z)
{
    float length_x = 0, length_y = 0, length_z = 0;

    if (x != KEEP)
        length_x = x - site_now[leg][0];
    if (y != KEEP)
        length_y = y - site_now[leg][1];
    if (z != KEEP)
        length_z = z - site_now[leg][2];
    float length = sqrt(pow(length_x, 2) + pow(length_y, 2) + pow(length_z, 2));
    temp_speed[leg][0] = length_x / length * move_speed * speed_multiple;
    temp_speed[leg][1] = length_y / length * move_speed * speed_multiple;
    temp_speed[leg][2] = length_z / length * move_speed * speed_multiple;
    if (x != KEEP)
        site_expect[leg][0] = x;
    if (y != KEEP)
        site_expect[leg][1] = y;
    if (z != KEEP)
        site_expect[leg][2] = z;
}

void wait_reach(int leg)
{
    while (1)
        if (site_now[leg][0] == site_expect[leg][0])
            if (site_now[leg][1] == site_expect[leg][1])
                if (site_now[leg][2] == site_expect[leg][2])
                    break;
}

```

```

}

void wait_all_reach(void)
{
    for (int i = 0; i < 4; i++)
        wait_reach(i);
}

void cartesian_to_polar(volatile float &alpha, volatile float &beta, volatile float
&gamma, volatile float x, volatile float y, volatile float z)
{
    float v, w;
    w = (x >= 0 ? 1 : -1) * (sqrt(pow(x, 2) + pow(y, 2)));
    v = w - length_c;
    alpha = atan2(z, v) + acos((pow(length_a, 2) - pow(length_b, 2) + pow(v, 2) +
pow(z, 2)) / 2 / length_a / sqrt(pow(v, 2) + pow(z, 2)));
    beta = acos((pow(length_a, 2) + pow(length_b, 2) - pow(v, 2) - pow(z, 2)) / 2 /
length_a / length_b);
    gamma = (w >= 0) ? atan2(y, x) : atan2(-y, -x);
    alpha = alpha / pi * 180;
    beta = beta / pi * 180;
    gamma = gamma / pi * 180;
}

void polar_to_servo(int leg, float alpha, float beta, float gamma)
{
    if (leg == 0)
    {
        alpha = 90 - alpha;
        beta = beta;
        gamma += 90;
    }
}

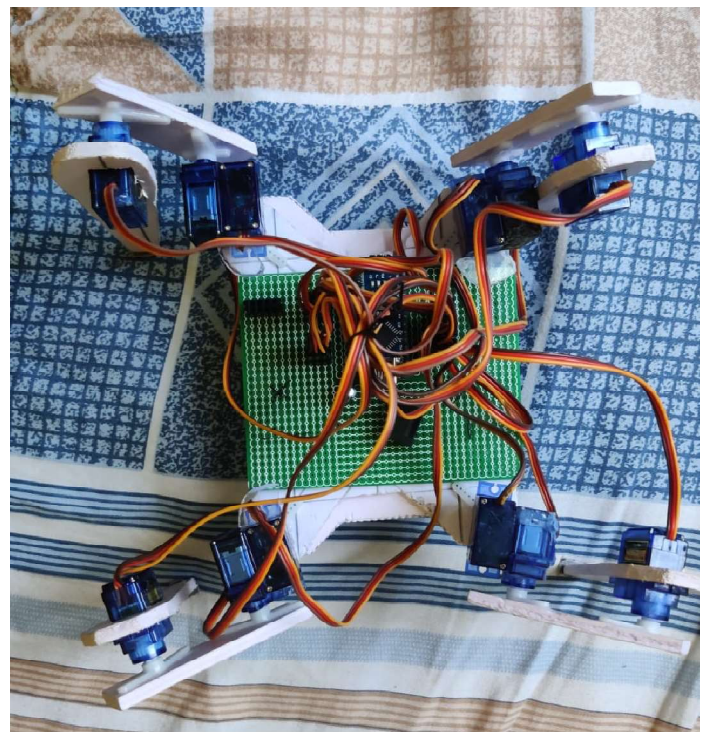
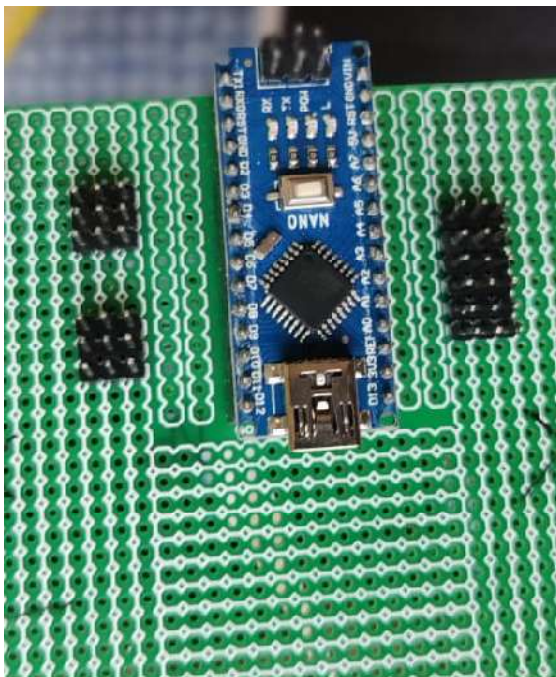
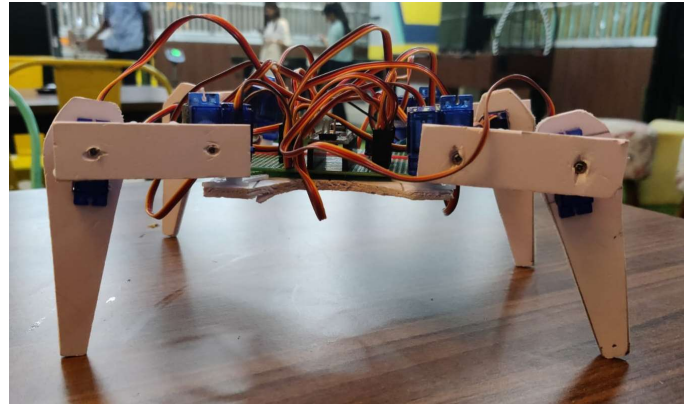
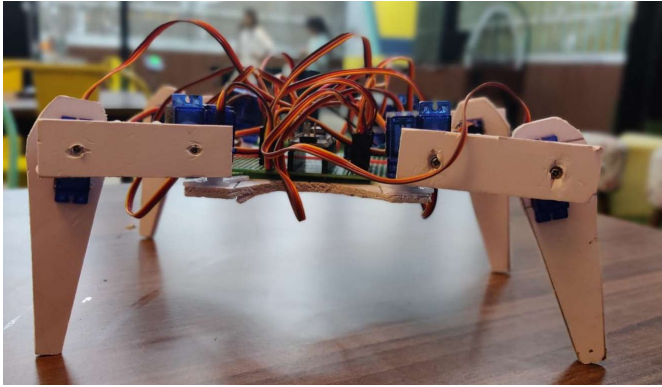
```

```
else if (leg == 1)
{
    alpha += 90;
    beta = 180 - beta;
    gamma = 90 - gamma;
}
else if (leg == 2)
{
    alpha += 90;
    beta = 180 - beta;
    gamma = 90 - gamma;
}
else if (leg == 3)
{
    alpha = 90 - alpha;
    beta = beta;
    gamma += 90;
}
servo[leg][0].write(alpha);
servo[leg][1].write(beta);
servo[leg][2].write(gamma);
}
```



# Images

---



# Conclusion

---

Quadrupeds are excellent tool for providing situational awareness, mapping debris etc. to areas where it was previously extremely difficult to do so. The quadruped is not cost effective but its manoeuvrability and efficiency while navigating through obstacles is far, its usage in monitoring harsh environment like one found in nuclear power plants or for gathering tracing and locating makes it far more advance robot in its field