Contents

IMPLEMENTATION OF STAIR CLIMBING ROBOT

Abstract

This project involves the design and development of a stair climbing robot consisting of three parts that can climb stairs using prismatic joints while providing support to each other. The robot is assumed to move on a flat surface before encountering the stairs and will be powered by electric motors and controlled by a microcontroller. It is equipped with sensors to detect the stairs and adjust its movements accordingly. The robot can sense the depth of each step and adjust its position to ensure stability. Calculations for the kinematic parameters, such as joint angles and displacements, are performed to enable smooth stair climbing. Design recommendations include ensuring stability, low center of gravity, wide base for the bottom part, and a well-balanced relay mechanism at the top.

1 Objectives

1. The objective of this experiment is to design and implement a stair-climbing robot capable of lifting and maneuvering different sections independently.

2 Introduction

In this project, we present the design and development of a stair climbing robot consisting of three parts that are constructed using SolidWorks. The robot is designed to climb stairs using prismatic joints while providing support to each other. The robot operates on a flat surface before encountering the stairs and is powered by electric motors controlled by a microcontroller. To begin the design process, each part of the robot is constructed in SolidWorks, a computer-aided design (CAD) software, ensuring precise dimensions and structural integrity. Once the parts are designed, they are exported into V-REP (Virtual Robot Experimentation Platform), a robot simulation software, to visualize and test the robot's behavior in a virtual environment. After verifying the design in V-REP, the trajectory planning is carried out to determine the optimal path and movement sequence for the robot to climb the stairs smoothly. The trajectory planning takes into account the dimensions of the stairs and the capabilities of the robot's prismatic joints. It ensures that the robot adjusts its position on each step to maintain stability and avoids any potential collisions or obstacles. In addition to the mechanical design and trajectory planning, the robot's control system is implemented using an Arduino microcontroller. The Arduino is programmed to receive remote control commands and control the electric motors accordingly. This allows for convenient remote operation of the robot during stair climbing tasks. Overall, this project combines mechanical design, simulation, trajectory planning, and microcontroller programming to develop a stair climbing robot capable of autonomously navigating and conquering stairs with stability and efficiency.

3 Back ground

The field of robotics has witnessed significant advancements in recent years, with robots being employed in various domains to perform complex tasks. One area of interest is the development of stair climbing robots, which can navigate stairs and overcome height differences in an environment. Stair climbing robots find applications in areas such as search and rescue operations, household assistance, and industrial automation. The design and development of a stair climbing robot pose several challenges. The robot needs to have the ability to adapt to different stair dimensions and adjust its movements accordingly. It should possess a stable and balanced structure to prevent tipping over during climbing. Additionally, the robot should be capable of sensing the presence of stairs, accurately determining their dimensions, and planning its trajectory for efficient stair climbing. To address these challenges, this project focuses on the design and development of a stair climbing robot with three parts that provide mutual support during climbing. The robot's parts are constructed using SolidWorks, a widely used CAD software, allowing for precise design and structural analysis. The design is then exported to V-REP, a robot simulation platform, to validate the robot's behavior and test its climbing capabilities in a virtual environment. Trajectory planning plays a crucial role in ensuring successful stair climbing. By considering the dimensions of the stairs and the capabilities of the robot's prismatic joints, an optimal path and movement sequence are determined. This planning accounts for maintaining stability, adjusting the robot's position on each step, and avoiding collisions or obstacles. The control system of the robot is implemented using an Arduino microcontroller, which enables remote control functionality. By programming the Arduino, the robot can receive commands and control the electric motors that drive its movements. This allows for convenient operation and control of the robot during stair climbing tasks. Through this project, we aim to demonstrate the feasibility and effectiveness of a stair climbing robot with three parts that can navigate stairs while providing mutual support. By combining mechanical design, simulation, trajectory planning, and microcontroller programming, we can develop a versatile and efficient robot capable of autonomously climbing stairs in various environments.

4 Theory

4.1 Stair Climbing Mechanism

The stair climbing robot is equipped with a prismatic joint mechanism that allows it to climb stairs. The prismatic joint provides linear motion along a specified axis, enabling the robot to adjust its height as it ascends or descends each step. By utilizing prismatic joints, the robot can maintain stability and provide support to each part during stair climbing.

4.2 Kinematic Calculations

To ensure smooth and efficient stair climbing, kinematic calculations are performed. The joint angle and joint displacement are determined for each part of the robot. The joint angle is calculated using the formula of tangent where angle represents the joint angle, h is the stair height, and w is the stair width.The joint displacement is calculated using the formula d = h/sin(angle), where d represents the joint displacement. These calculations enable the robot to adjust its position and ensure proper climbing motion on each step.

4.3 Trajectory Planning

Trajectory planning is a crucial aspect of the stair climbing robot's operation. It involves determining the optimal path and movement sequence for the robot to climb the stairs smoothly. The trajectory planning takes into account the dimensions of the stairs, the capabilities of the prismatic joints, and the stability requirements. By planning the trajectory, the robot can adjust its position on each step, maintain stability, and avoid collisions or obstacles during the climbing process.

4.4 SolidWorks and V-REP

SolidWorks, a widely used CAD software, is employed for designing the robot's parts. It provides precise dimensions and structural analysis capabilities, ensuring the integrity of the robot's construction. The design is then exported to V-REP, a robot simulation platform, to visualize and test the robot's behavior in a virtual environment. V-REP allows for the validation of the design and enables iterative improvements before physical implementation

4.5 Arduino Microcontroller

The control system of the robot is implemented using an Arduino microcontroller. The microcontroller receives remote control commands and controls the electric motors that drive the robot's movements. By programming the Arduino, the robot can be operated remotely and execute the desired actions during stair climbing tasks. The Arduino enables convenient and efficient control over the robot's actions.

5 Design Criteria

5.1 Stability

The stair climbing robot must have a stable design to prevent tipping or instability during the climbing process. The center of gravity should be kept low, and the base of the robot, particularly the bottom part, should be wide enough to provide a solid foundation.

5.2 Adaptability to Stair Dimensions

The robot should be able to adapt to different stair dimensions. It should be able to sense the height and depth of each step accurately and adjust its movements and position accordingly. The design should accommodate stairs with varying heights and widths.

5.3 Mutual Support

: The three parts of the robot should provide support to each other during stair climbing. The design should ensure that each part can maintain stability and provide stability to the other parts. The distance between the parts should be optimized to ensure proper support and balance.

5.4 Sensors and Perception

The robot should be equipped with sensors to detect the presence of stairs and accurately perceive their dimensions. These sensors can be infrared sensors, ultrasonic sensors, or any other suitable sensing mechanism. The robot should be able to process the sensor data to adjust its movements and position accordingly.

5.5 Trajectory Planning

The design should incorporate trajectory planning algorithms to determine the optimal path and movement sequence for climbing the stairs. The trajectory planning should ensure smooth and efficient climbing while considering stability, adjusting position on each step, and avoiding obstacles.

5.6 Mechanical Robustness

The robot's mechanical design should be robust and capable of withstanding the stresses and strains associated with stair climbing. The materials used should be strong, durable, and lightweight to optimize the robot's performance.

5.7 Remote Control

The robot should have the option for remote control operation. It should be possible to control the robot's movements remotely, allowing for convenient and safe operation during stair climbing tasks.

5.8 Energy Efficiency

The design should aim to optimize energy efficiency by using efficient electric motors and implementing power-saving measures. This will enable longer operation time and reduce the need for frequent battery replacement or recharging.

5.9 Safety

Safety considerations should be incorporated into the design, ensuring that the robot operates without causing harm to users or the environment. The design should include safety mechanisms such as emergency stop buttons or sensors to detect obstacles or obstructions during stair climbing.

5.10 Cost-Effectiveness

The design should be cost-effective, considering the overall budget constraints. The choice of materials, components, and manufacturing processes should be balanced to achieve an optimal cost-to-performance ratio.

6 Material and Apparatus

Since the project has been simulated using SolidWorks and V-REP, the material and apparatus used are virtual in nature. Here are the simulated materials and apparatus employed in the project:

6.1 SolidWorks

SolidWorks is a computer-aided design (CAD) software used for designing and modeling the robot's parts. It provides a virtual environment for constructing and analyzing the robot's structure, ensuring precise dimensions and structural integrity.

6.2 V-REP (Virtual Robot Experimentation Platform)

V-REP is a robot simulation platform used to validate the behavior and functionality of the stair climbing robot. It provides a virtual environment where the robot's design and trajectory planning can be tested and optimized before physical implementation.

6.3 Computer System

A computer system equipped with the necessary hardware and software is required to run SolidWorks and V-REP. The system should meet the minimum requirements specified by the software vendors for smooth operation

7 Procedure

1. Assemble the robot chassis by securing the motors onto the 3D printed mount between two extrusion pieces.
2. Attach the couplers to prevent the motors from pulling apart and ensure stability.
3. Place a plate with a bearing above the couplers to restrict separation.
4. Mount the V-wheels on a sliding piece, coupled to the ball nut, to create a linear sliding axis.
5. Connect the worm gear driven gearboxes to the drive wheels to enable differential drive for improved carpet grip.

6. Attach wheels to the front forks, allowing the robot to perform skid steering without drag issues.
7. Implement the rack and pinion assembly to move the axis, enabling the robot to reach over multiple steps simultaneously.
8. Set up the belt-driven axis to move the two 1-kilogram lead weights.
9. Position the weights according to the desired weight distribution, making the robot either front-heavy or back-heavy.
10. Test the robot's lifting capability by lifting the front section and driving it forward.
11. Position the lifted section in the desired location and move the mass to the middle.
12. Pick up the middle section and drive on with the next two sections.
13. Move the mass to the back section and pick it up accordingly.
14. Repeat the previous steps until all sections are successfully maneuvered.
15. Document and analyze the robot's performance, including its ability to climb stairs, lift and transport sections, and maintain stability.
16. Summarize the findings, observations, and any improvements that can be made for future iterations

8 Work plan

8.1 Introduction

1. Provide an overview of the project and its objectives.
2. Explain the importance of designing a stair-climbing robot for various applications.

8.2 SolidWorks Design

1. Use SolidWorks (or any other suitable CAD software) to design the robot chassis, extrusion, motor mounts, and other components.
2. Create 3D models of the robot and its individual parts.
3. Ensure accurate measurements, proper fit, and structural integrity of the design.

8.3 Importing the Design to V-REP

1. Export the SolidWorks model in a compatible file format (e.g., .STL, .OBJ, or .STEP).
2. Import the exported design into V-REP (Virtual Robot Experimentation Platform).
3. Set up the virtual environment in V-REP, including the floor, stairs, and other objects as needed.
4. Verify that the imported model aligns with the virtual environment and accurately represents the physical robot design.

8.4 Importing the Design to V-REP

1. Define the desired trajectory for the robot to climb stairs and lift/transport sections.
2. Utilize V-REP's simulation capabilities to plan and visualize the trajectory
3. Implement appropriate algorithms for path planning, obstacle avoidance, and motion control.
4. Fine-tune the trajectory based on the robot's kinematics and desired performance.

8.5 Simulation Testing

1. Conduct simulation tests in V-REP to evaluate the robot's ability to climb stairs and perform lifting/transportation tasks
2. Record the robot's behavior, motion, and any observed issues or limitations.
3. Analyze the simulation results and iterate on the trajectory planning if necessary.
4. Document the simulation setup, parameters, and observations

8.6 Data Analysis and Discussion

1. Analyze the simulation data to evaluate the robot's performance and capabilities.
2. Discuss the effectiveness of the trajectory planning in achieving the desired tasks.
3. Identify any challenges or limitations encountered during the simulation testing.
4. Discuss possible improvements or modifications to enhance the robot's performance.

8.7 Conclusion

(a) Summarize the findings from the SolidWorks design and V-REP simulation.

(b) Highlight the strengths and weaknesses of the robot's design and trajectory planning.

(c) Discuss the potential real-world applications and future development possibilities.

(d) Conclude the lab report by emphasizing the achievements and the significance of the project.

9 Discussion

The implementation of a stair-climbing robot involved several stages, including the design phase using SolidWorks, simulation in V-REP, and coding the remote control functionality in Arduino. Each stage contributed to the overall development and evaluation of the robot's performance. Let's discuss the key aspects of the implementation:

9.1 SolidWorks Design

The use of SolidWorks allowed for precise and detailed design of the robot's chassis, extrusion, motor mounts, and other components. By creating 3D models, we could visualize the physical structure and ensure proper fit and functionality. The design stage provided a solid foundation for further development and testing.

9.2 V-REP Simulation

Importing the SolidWorks design into V-REP facilitated the simulation of the stair-climbing robot. The virtual environment enabled us to assess the robot's performance in realistic scenarios without the need for physical hardware. By defining the desired trajectory and utilizing V-REP's simulation capabilities, we could plan and visualize the robot's movements, climb stairs, and lift/transport sections.

9.3 Trajectory Planning

The trajectory planning stage was crucial for mapping out the robot's path and ensuring efficient and safe stair climbing. Through the use of appropriate algorithms, we implemented path planning, obstacle avoidance, and motion control strategies. The simulation tests allowed us to refine the trajectory and assess the robot's ability to navigate stairs and complete lifting tasks.

9.4 Remote Control Implementation

To enable wireless control, we utilized Arduino for coding the remote control functionality. By integrating the Arduino Mega microcontroller and Nrf24l01 radio chip, we established communication between the custom remote control and the robot. The remote control code enabled us to send commands wirelessly, facilitating the control and operation of the robot during the simulation.

9.5 Overall Performance and Future Considerations

The implementation stages provided valuable insights into the stair-climbing robot's performance. Through simulation, we evaluated the robot's ability to climb stairs, lift and transport sections, and maintain stability. The trajectory planning and remote control implementation demonstrated the effectiveness of our approach in achieving the desired tasks. While the simulation results provided a good basis for evaluation, it is important to note that the real-world implementation and hardware integration may present additional challenges. Factors such as weight distribution, traction, and the physical dynamics of the robot can significantly impact its performance. Future considerations should include testing and refining the design using physical components, as well as optimizing the control algorithms based on real-world constraints.

10 Conclusion

In conclusion, the implementation of the stair-climbing robot through SolidWorks design, V-REP simulation, and remote control coding in Arduino allowed for a comprehensive evaluation of the robot's performance. The combination of these stages provided valuable insights for further development and highlighted the potential for real-world applications of the robot in various scenarios. In conclusion, the implementation of the stair-climbing robot using SolidWorks design, V-REP simulation, and remote control coding in Arduino has been a significant step towards achieving the project objectives. Through the use of SolidWorks, we were able to design a precise and functional robot chassis, motor mounts, and other components, ensuring a solid foundation for the project. The V-REP simulation provided a realistic virtual environment where we could test and evaluate the robot's performance in climbing stairs and performing lifting tasks. The trajectory planning algorithms allowed us to plan efficient paths and avoid obstacles, showcasing the robot's capabilities. The remote control implementation using Arduino enabled wireless communication and control, enhancing the robot's usability and ease of operation.

11 Appendixes

11.1 Code snippet

11.2 Trajectory planning

```
-- Initialize handles for robot components
local frontSliderHandle = sim.getObjectHandle("front_slider")
local midSliderHandle = sim.getObjectHandle("mid_slider")
local backSliderHandle = sim.getObjectHandle("back_slider")
local gravityShifterHandle = sim.getObjectHandle("gravity_shifter")

-- Calculate target positions and orientations
local stairHeight = 7
local stairWidth = 11
local firstPartLength = 12
local secondPartLength = 8
local distanceBetweenParts = 29.82
local bottomPartJointAngle = 32.47
local bottomPartJointDisplacement = 8.34

-- Set initial positions and orientations
local initialFrontSliderPos = sim.getObjectPosition(frontSliderHandle, -1)
local initialMidSliderPos = sim.getObjectPosition(midSliderHandle, -1)
local initialBackSliderPos = sim.getObjectPosition(backSliderHandle, -1)
local initialGravityShifterPos = sim.getObjectPosition(gravityShifterHandle, -1)

-- Implement trajectory planning loop
local numSteps = 5 -- Number of steps to climb
local stepHeight = stairHeight / numSteps -- Height of each step

for i = 1, numSteps do
    -- Adjust target positions and orientations for each part
 local targetFrontSliderPos =

{initialFrontSliderPos[1],
 initialFrontSliderPos[2],
 initialFrontSliderPos[3] + (i *
 stepHeight)}

    -- Set target positions for each part
    sim.setObjectPosition(frontSliderHandle, -1, targetFrontSliderPos)
    sim.setObjectPosition(midSliderHandle, -1, targetMidSliderPos)

    sim.setObjectPosition(backSliderHandle, -1, targetBackSliderPos)

    -- Wait for stabilization between steps
    local stabilizationTime = 1 -- Adjust as needed
    sim.wait(stabilizationTime)
end
```

12 Remote control code

```
// Radio
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

RF24 radio(7, 8); // CE, CSN
const byte addresses[][6] = {"00001", "00002"};

//**************remote control****************
struct RECEIVE_DATA_STRUCTURE{
  //put your variable definitions here for the data you want to send
  //THIS MUST BE EXACTLY THE SAME ON THE OTHER ARDUINO

    int16_t RLR;
    int16_t RFB;
    bool sw1;
    bool sw2;
    bool sw3;
    bool sw4;
    bool sw5;
    bool sw6;
    bool sw7;
    bool sw8;


};

RECEIVE_DATA_STRUCTURE mydata_remote;

int RLR = 0;
int RFB = 0;
int RFBa = 0;

int drive1;
int drive2;
int drive1a;
int drive2a;

unsigned long currentMillis;
long previousMillis = 0;    // set up timers
long interval = 10;        // time constant for timer

unsigned long remoteMillis;   // safety timer
int remoteState;

void setup() {

    // initialize serial communication
    Serial.begin(115200);

    radio.begin();
    radio.openWritingPipe(addresses[0]); // 00002
    radio.openReadingPipe(1, addresses[1]); // 00001
    radio.setPALevel(RF24_PA_MIN);
    radio.startListening();

    pinMode(2, OUTPUT);   // wheel PWMs
    pinMode(3, OUTPUT);
    pinMode(4, OUTPUT);
    pinMode(5, OUTPUT);

    pinMode(6, OUTPUT);   // back linear axis
    pinMode(9, OUTPUT);

    pinMode(10, OUTPUT);  // front linear axis
    pinMode(11, OUTPUT);

    pinMode(22, OUTPUT);  // vertical linear axis
    pinMode(24, OUTPUT);
    pinMode(26, OUTPUT);
    pinMode(28, OUTPUT);


}   // end of setup

// ******************** MAIN LOOP ******************************
```

```
void loop() {


        currentMillis = millis();
        if (currentMillis - previousMillis >= 10) {  // start timed event

            previousMillis = currentMillis;


            // check for radio data
            if (radio.available()) {
                    radio.read(&mydata_remote, sizeof(RECEIVE_DATA_STRUCTURE));
                    remoteMillis = currentMillis;
            }

            // is the remote disconnected for too long ?
            if (currentMillis - remoteMillis > 500) {
              remoteState = 0;
              mydata_remote.RFB = 512;
              mydata_remote.RLR = 512;
              mydata_remote.sw1 = 1;
              mydata_remote.sw2 = 1;
              mydata_remote.sw3 = 1;
              mydata_remote.sw4 = 1;
              mydata_remote.sw5 = 1;
              mydata_remote.sw6 = 1;
              mydata_remote.sw7 = 1;
              mydata_remote.sw8 = 1;
              Serial.println("no data");
            }
            else {
              remoteState = 1;
            }


            // threshold remote data
            // some are reversed based on stick wiring in remote
            RFB = thresholdStick(mydata_remote.RFB)/1.8;
            RLR = thresholdStick(mydata_remote.RLR)/1.8;

            drive1 = RFB + RLR;
            drive2 = RFB - RLR;

            drive1 = constrain(drive1, -255,255);
```

```
drive2 = constrain(drive2, -255,255);

// drive wheels
//left wheel

if (drive1 > 0) {
  analogWrite(3, drive1);
  analogWrite(2, 0);
}

else if (drive1 < 0) {
  drive1a = abs(drive1);
  analogWrite(2, drive1a);
  analogWrite(3, 0);
}

else {
  analogWrite(2, 0);
  analogWrite(3, 0);
}

//right wheel

if (drive2 > 0) {
  analogWrite(4, drive2);
  analogWrite(5, 0);
}

else if (drive2 < 0) {
  drive2a = abs(drive2);
  analogWrite(5, drive2a);
  analogWrite(4, 0);
}

else {
  analogWrite(4, 0);
  analogWrite(5, 0);
}

// rear linear axis

if (mydata_remote.sw4 == 0) {
  analogWrite(6, 75);
  analogWrite(9, 0);
}
```

```
else if (mydata_remote.sw3 == 0) {
  analogWrite(9, 75);
  analogWrite(6, 0);
}

else {
  analogWrite(9, 0);
  analogWrite(6, 0);
}

// front axis linear

if (mydata_remote.sw5 == 0) {
  analogWrite(11, 30);
  analogWrite(10, 0);
}

else if (mydata_remote.sw6 == 0) {
  analogWrite(10, 30);
  analogWrite(11, 0);
}

else {
  analogWrite(10, 0);
  analogWrite(11, 0);
}

// front linear axis

if (mydata_remote.sw7 == 0) {
  digitalWrite(26, HIGH);
  digitalWrite(28, 0);
}

else if (mydata_remote.sw8 == 0) {
  digitalWrite(28, HIGH);
  digitalWrite(26, 0);
}

else {
  digitalWrite(26, 0);
  digitalWrite(28, 0);
}
```

```
// back linear axis

if (mydata_remote.sw1 == 0) {
  digitalWrite(24, HIGH);
  digitalWrite(22, 0);
}

else if (mydata_remote.sw2 == 0) {
  digitalWrite(22, HIGH);
  digitalWrite(24, 0);
}

else {
  digitalWrite(22, 0);
  digitalWrite(24, 0);
}

/*

Serial.print(RFB);
Serial.print(" , ");
Serial.print(RLR);
Serial.print(" , ");
Serial.print(mydata_remote.sw1);
Serial.print(" , ");
Serial.print(mydata_remote.sw2);
Serial.print(" , ");
Serial.print(mydata_remote.sw3);
Serial.print(" , ");
Serial.print(mydata_remote.sw4);
Serial.print(" , ");
Serial.print(mydata_remote.sw5);
Serial.print(" , ");
Serial.print(mydata_remote.sw6);
Serial.print(" , ");
Serial.print(mydata_remote.sw7);
Serial.print(" , ");
Serial.println(mydata_remote.sw8);

*/
```

```
        }     // end of timed loop


}       // end  of main loop
int thresholdStick (int pos) {

    // get zero centre position
    pos = pos - 512;

    // threshold value for control sticks
    if (pos > 50) {
      pos = pos -50;
    }
    else if (pos < -50) {
      pos = pos +50;
    }
    else {
      pos = 0;
    }

    return pos;
}



// motion filter to filter motions and compliance

float filter(float prevValue, float currentValue, int filter) {
  float lengthFiltered =  (prevValue + (currentValue * filter)) / (filter + 1);
  return lengthFiltered;
}
```