Lever Automation in Walk Behind Rollers

Aditi Jha

Abstract

The automation of walk-behind rollers (WBR) aims to enhance operational efficiency, reduce manual effort, and improve precision in construction tasks. This project focuses on automating the lever mechanism of WBRs, traditionally controlled manually, to optimize the adjustment of direction and vibration levers. By integrating a microcontroller , the automation system will allow for precise control of these components. The primary objective is to test and evaluate the performance of this automated system, ensuring the reliability and functionality of the WBR with minimal manual intervention, while setting the groundwork for potential future enhancements in construction equipment automation.

Introduction

Purpose

The purpose of this project is to develop and test an automated system for controlling the lever mechanism of Walk Behind Rollers (WBR).This project will serve as a testbed for evaluating the effectiveness and reliability of automation in construction equipment, with the ultimate goal of enhancing efficiency and functionality in construction tasks. The insights gained will inform future advancements in automated solutions for heavy machinery.

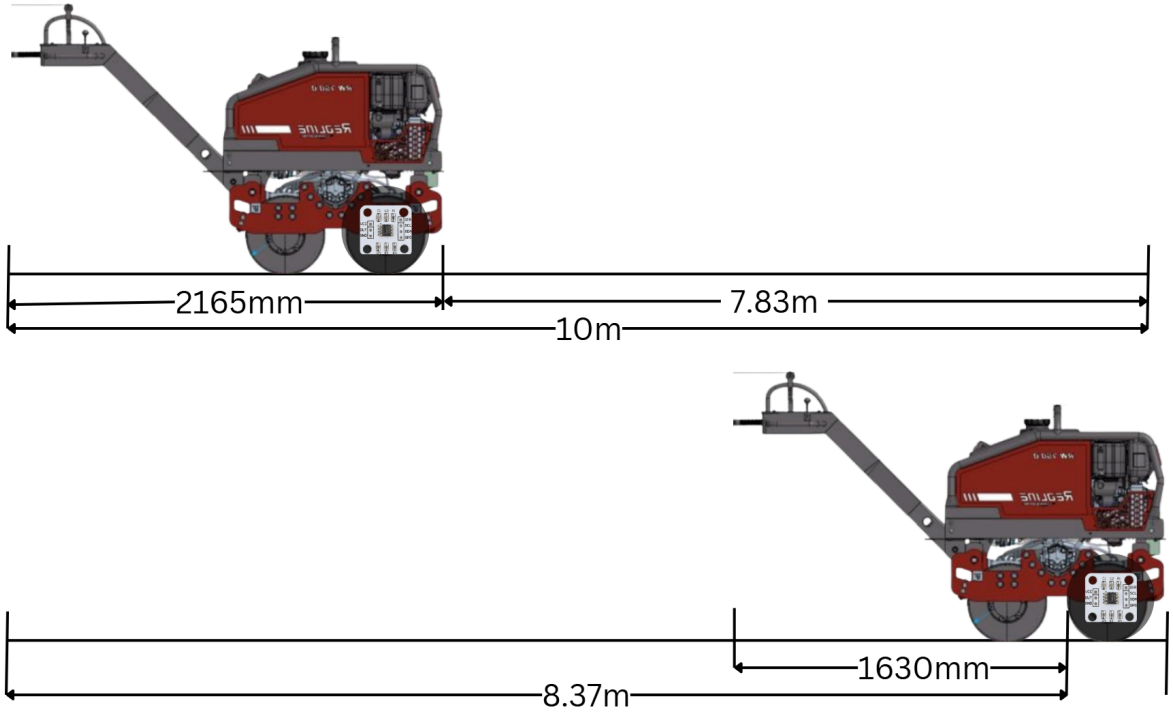
Problem Statement

An automated control system should be designed for the Walk-Behind Roller (WBR) to enhance operational efficiency and reduce manual intervention. The system should be capable of performing the following tasks:

1. **Forward Movement with Neutral Pause:** The system should enable the WBR to move forward to a predefined distance at a controlled speed within the range of 0–4.5 km/h. Once the predefined distance is covered, the system should automatically switch to a neutral state for a predefined duration.
2. **Reverse Movement with Neutral Pause:** After the neutral pause, the system should reverse the WBR back to its starting position, maintaining the same speed range. Upon returning to the starting point, the system should again switch to neutral for the predefined duration, repeating the cycle until the total predefined distance is covered.
3. **Vibration Mode Control:** The system should be capable of switching between two vibration modes based on the direction of movement:
    - **Mode 1** should be activated during forward motion to improve compaction efficiency.
    - **Mode 2** should be activated during reverse motion to ensure effective re-compaction and prevent disruption of already compacted areas.

This automated control system should optimize the WBR's operation by minimizing the need for manual intervention, ensuring consistent performance, and improving efficiency in compaction tasks.

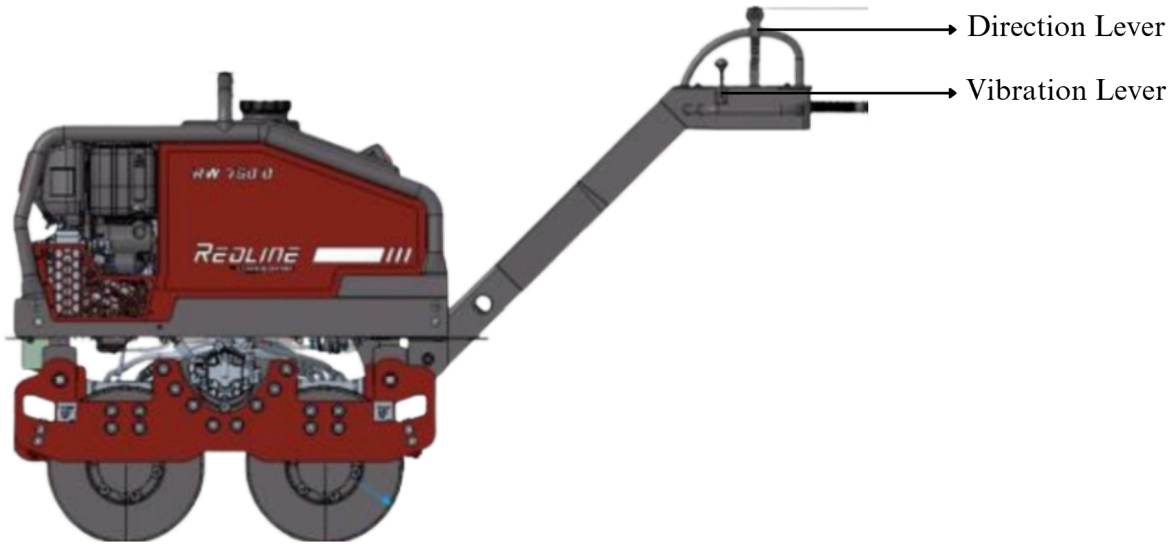
Understandings



Distance to be covered in Forward and Reverse Direction respectively

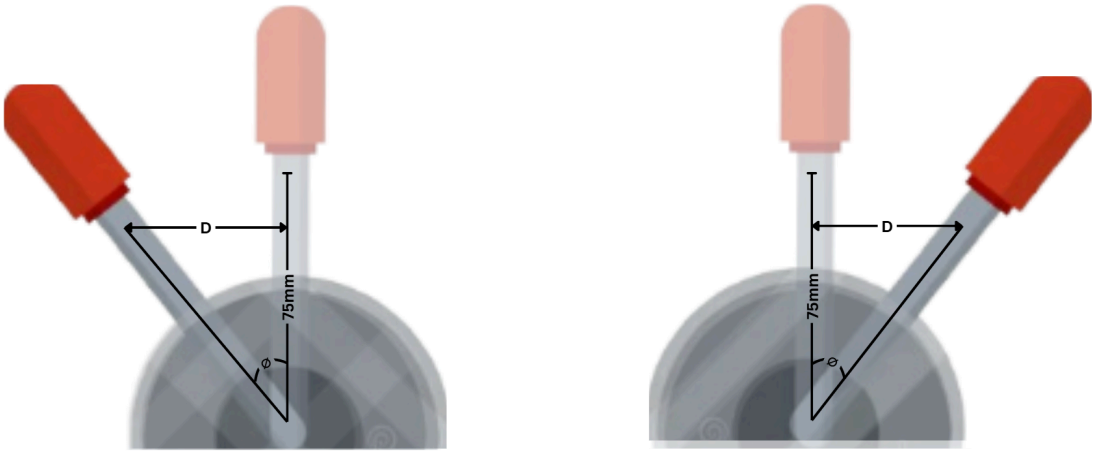
1. The vibration lever moves from the Neutral (N) position to the Forward (F) position, activating Vibration Mode 1 (VBM1).
2. After 2 seconds, the directional lever moves from the Neutral (N) position to the Forward (F) position, initiating forward motion.
3. With the help of an encoder, the system continuously monitors the distance traveled. Once the machine covers a distance of 7.83m, both the vibration lever and directional lever return to the Neutral (N) position.
4. The machine remains stationary in this position for 5 seconds.

5. The vibration lever then moves from the Neutral (N) position to the Reverse (R) position, activating Vibration Mode 2 (VBM2).
6. After 2 seconds, the directional lever moves from the Neutral (N) position to the Reverse (R) position, initiating reverse motion.
7. The encoder continues monitoring the distance, and once the machine travels 8.37m, both the vibration lever and directional lever return to the Neutral (N) position.



Position of Levers in WBR

Direction Lever Dimensions and Movement



Vibration Lever Dimensions and Movement

Proposal of Ideas

In this report, I have developed and will be presenting three potential ideas for the design and implementation of an automated control system for the Walk-Behind Roller (WBR). Each idea offers a unique approach to addressing the core objectives of the system, which include efficient movement and vibration mode control .

The three proposed ideas are:

1. ***YokeDrive: Stepper-Based Automation***

   The YokeDrive system focuses on utilizing high-torque stepper motors coupled with a custom-built Scotch yoke mechanism. YokeDrive is ideal for scenarios requiring fine-tuned movement and precise control over lever positions.

2. ***Actuators: Relay-Driven Automation***

   This approach leverages linear actuators controlled via relays to achieve automated lever movement. The system employs a straightforward design that ensures reliability and ease of implementation. With relays managing actuator operation, this idea provides a cost-effective and durable solution for automating the WBR's levers.

3. ***ScotchDrive: Servo-Enhanced Motion***

   The ScotchDrive concept integrates high-precision servo motors with a Scotch yoke mechanism to achieve automated lever control. By ensuring smooth and responsive operation, ScotchDrive is tailored for scenarios requiring adaptive control and enhanced performance.

APPROACH 1

YokeDrive: Stepper-Based Automation

Description:

The YokeDrive system leverages stepper motors coupled with a Scotch yoke mechanism to automate lever motion in the Walk-Behind Roller (WBR). This design translates the rotary motion of the stepper motor into precise linear motion through the Scotch yoke, ensuring accurate control of lever positioning. The integration of real-time displacement monitoring via a magnetic rotary encoder further enhances the system's precision and reliability.

Key Features:

- **Automated Lever Movement:**

Utilizes stepper motors for precise, incremental rotary motion.

Converts rotary motion to linear motion via a custom-built Scotch yoke mechanism.

- **Real-Time Feedback:**

Incorporates a magnetic rotary encoder to monitor displacement, ensuring accurate position tracking.

- **Control System:**

Powered by an Arduino Mega microcontroller, which interfaces with motor drivers to control the stepper motors.

Optimized for efficient operation and reliable performance.

Components Required:

1. Arduino Mega:

Acts as the central processing unit for the control system.

Executes commands to operate the stepper motors and monitors encoder feedback.

2. 2 x Stepper Motors (High Torque):

Provides precise and powerful rotary motion suitable for converting into linear motion.

Ensures consistent and smooth operation under varying load conditions.

3. 2 x Motor Drivers:

Interfaces between the Arduino and stepper motors to regulate current and voltage.

Protects the motors from overloads and ensures efficient operation.

4. 2 x Scotch Yoke Mechanism:

Custom-built to translate rotary motion into linear motion.

Designed for durability and smooth operation in high-torque applications.

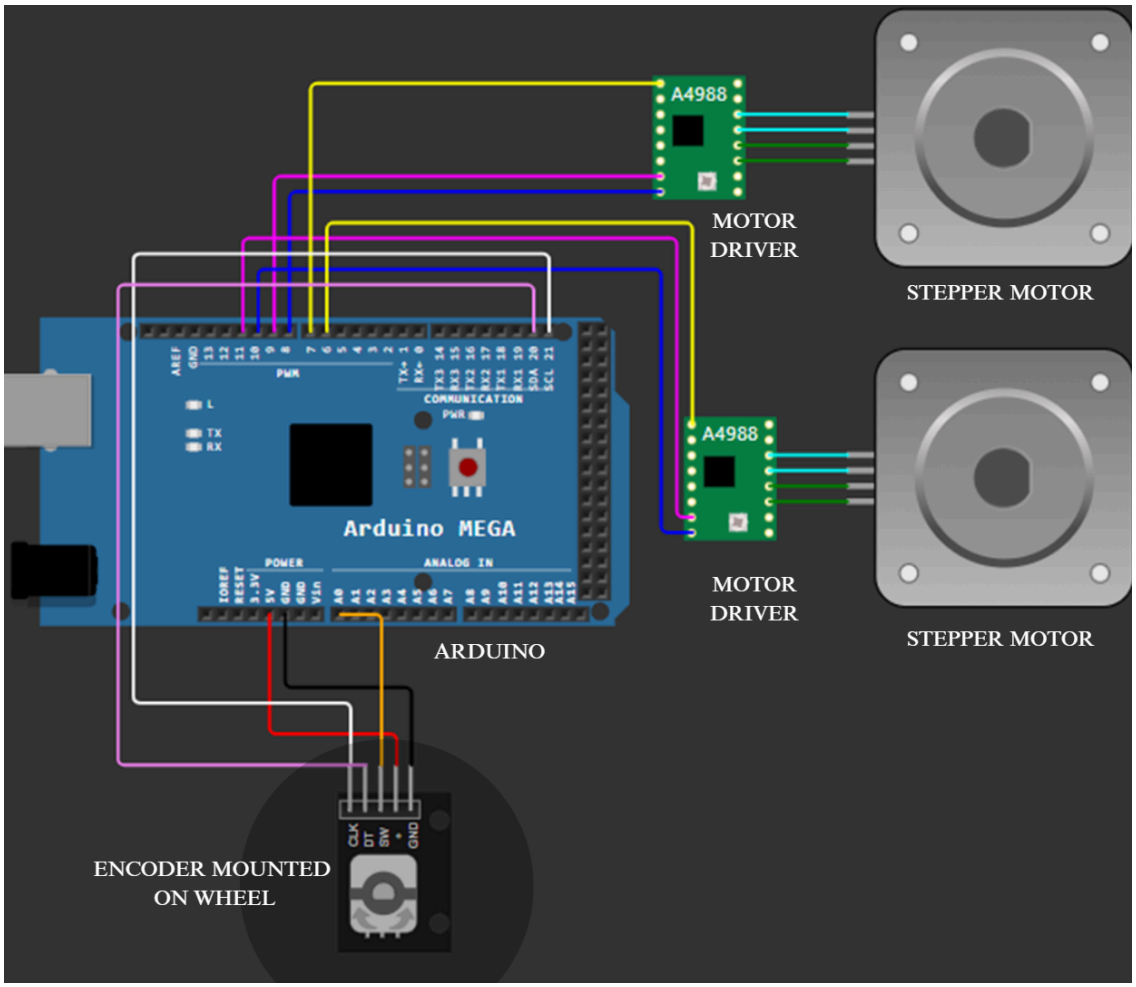
5. Magnetic Rotary Encoder:

Provides real-time feedback on rotary motion.

Ensures precise displacement monitoring and position accuracy.

6. Power Supply:

Delivers a stable and compatible power source for the stepper motors and other components.



Circuit Diagram

Working:

1. Initialization Phase:

The system initializes by setting the stepper motors to their default (neutral) positions.

2. Command Execution:

Based on the predefined input , the Arduino Mega sends control signals to the motor drivers.

The motor drivers regulate the current and voltage supplied to the stepper motors, ensuring smooth and precise motion.

3. Rotary Motion to Linear Motion Conversion:

The stepper motors rotate incrementally as commanded by the Arduino Mega.

The linear motion from the Scotch yoke is directly linked to the levers of the WBR, causing them to move between Neutral (N), Forward (F), and Reverse (R) positions.

4. Real-Time Monitoring and Feedback:

As the stepper motors rotate, the magnetic rotary encoder continuously monitors the angular displacement.

The encoder sends real-time feedback to the Arduino Mega, ensuring precise tracking of the lever's position.

5. Lever Operation:

For vibration mode control, the stepper motor linked to the vibration lever moves to the Forward (F) position to activate Vibration Mode 1 (VBM1) or to the Reverse (R) position for Vibration Mode 2 (VBM2).

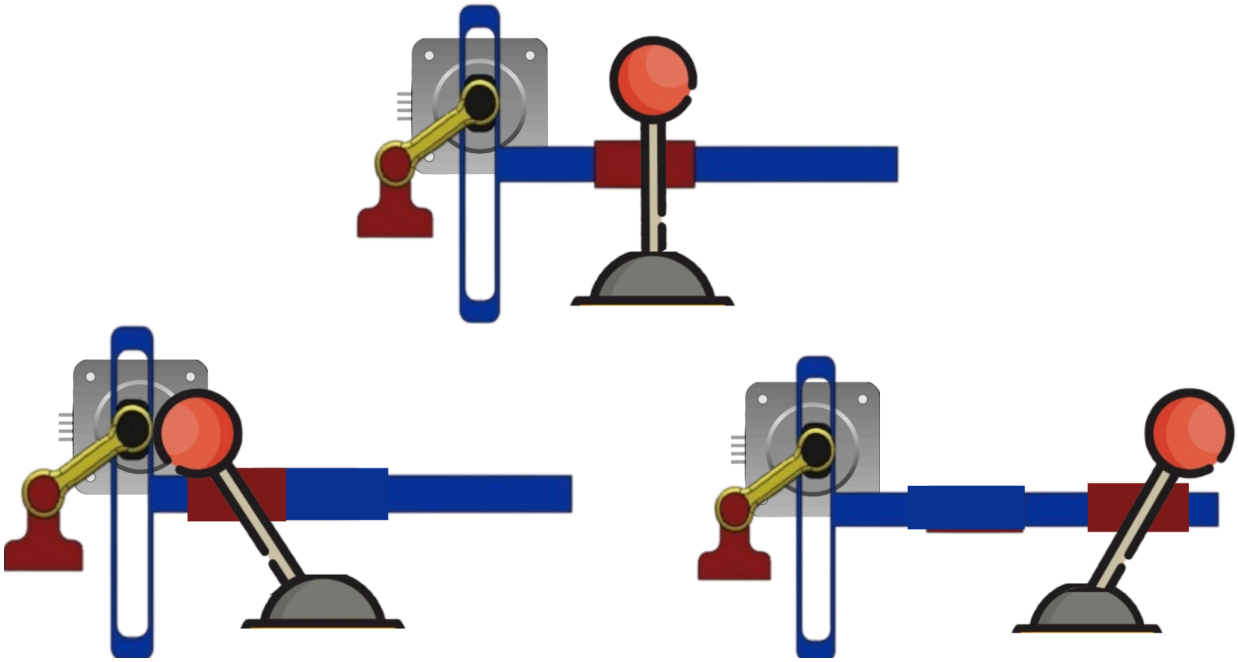
For directional control, the stepper motor linked to the directional lever moves to Forward (F) for forward motion or Reverse (R) for reverse motion.

6. Distance-Based Control:

When the predefined distance (e.g., 7.83m or 8.37m) is reached, the Arduino commands the stepper motors to move the levers back to the Neutral (N) position, halting the machine.

Motion Transfer mechanism:

The stepper motor will be mechanically linked to the Scotch yoke mechanism, where its rotary motion will be converted into linear motion. The Scotch yoke, in turn, will be connected to the lever via a series of precision-engineered links. This configuration ensures efficient transfer of motion from the motor to the lever, enabling accurate and repeatable lever operation.



Neutral, Forward & Reverse Motions

APPROACH 2

Actuators: Relay-Driven Automation

Description:

The Actuators: Relay-Driven Automation system utilizes relays to control the movement of levers in the Walk-Behind Roller (WBR) through actuator-driven mechanisms. Relays are employed to direct the actuators, controlling the movement of the levers between Forward (F), Reverse (R), and Neutral (N) positions. The integration of an encoder ensures precise monitoring of the machine's displacement, providing real-time feedback to track the progress and halt the movement at predefined distances. This design simplifies the control process while maintaining reliable lever movement for vibration mode switching and distance-based control.

Key Features:

- **Automated Lever Movement:**

Relays control the high-power actuators that drive the lever's movement.

Actuators provide linear motion in response to relay signals, moving the lever between the predefined positions: Neutral (N), Forward (F), and Reverse (R).

- **Real-Time Monitoring with Encoder:**

An encoder is used to monitor the position and displacement of the machine.

It provides feedback to the microcontroller (Arduino Mega) for precise distance-based control, ensuring the machine stops after reaching the defined travel distance (e.g., 7.83m or 8.37m).

- **Simplified Control System:**

The system utilizes basic relay-driven control, reducing complexity while maintaining reliable performance.

A microcontroller (Arduino) manages the relay switches, automating the WBR's motion based on predefined commands.

- **Vibration Mode Control:**

Two distinct vibration modes are controlled through relay and actuator movements:

Vibration Mode 1 (VBM1): Achieved when the lever is in the Forward (F) position.

Vibration Mode 2 (VBM2): Activated when the lever is in the Reverse (R) position.

Components Required:

1. Arduino Mega:

Acts as the central controller, commanding the relays based on the input signals.

Manages the automation of lever positions, vibration modes, and directionality, and receives real-time feedback from the encoder to monitor displacement.

2. Relays (4-Channel Relay Module):

Control the activation of actuators that adjust the position of the levers.

Ensures safe and reliable switching between positions (Neutral, Forward, Reverse) without mechanical wear.

3. Actuators:

Convert the relay signals into linear motion, driving the levers of the WBR.

Move the levers from Neutral to Forward and Reverse positions, based on the commanded input.

4. 2 x Clevis Mounts:

Used to link the actuators to the levers.

Provides a secure and adjustable connection for the actuators, allowing them to precisely control the lever movement.

Ensures that the actuators are properly aligned with the levers to transmit the desired motion.

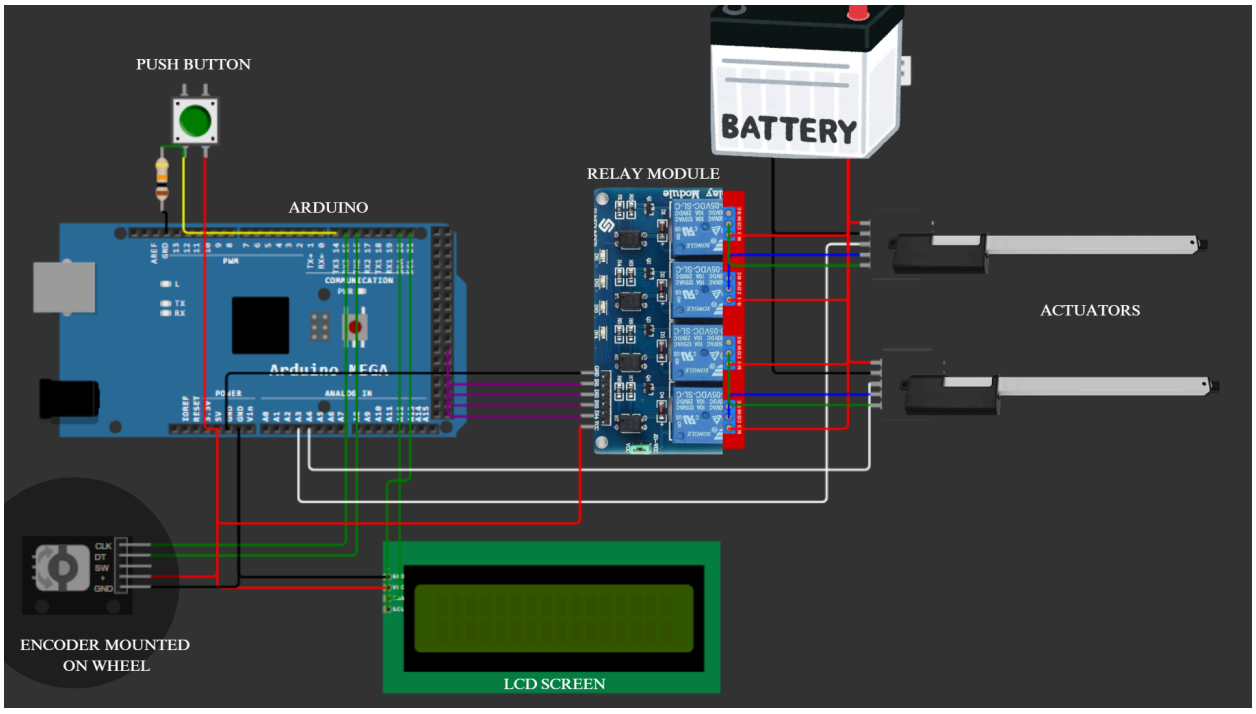
5. Magnetic Rotary Encoder:

Provides real-time feedback on the machine's displacement.

Tracks the machine's position and sends data to the Arduino Mega, which commands the actuators to stop when the predefined distance is achieved.

6. Power Supply:

Provides stable power to the relays, actuators, and Arduino.



Circuit Diagram

Working:

1. Initialization Phase:

The Arduino Mega initializes the system by setting the actuators and relay modules to their default positions (Neutral for levers).

The connected LCD displays the system status, such as "System Ready."

2. Command Execution:

Based on predefined user inputs via the keypad, the Arduino sends control signals to the relay module.

The relay module switches power to the linear actuators, controlling their movement.

3. Linear Actuator Operation:

The actuators extend or retract as per the relay signals. This movement directly translates into the operation of the levers for both vibration and directional control.

Actuators linked to the vibration lever set it to Neutral (N), Forward (F), or Reverse (R), toggling between vibration modes VBM1 and VBM2.

Similarly, actuators linked to the directional lever control the machine's forward and reverse motions.

4. Real-Time Monitoring and Feedback:

The magnetic rotary encoder attached to the wheel monitors its angular displacement.

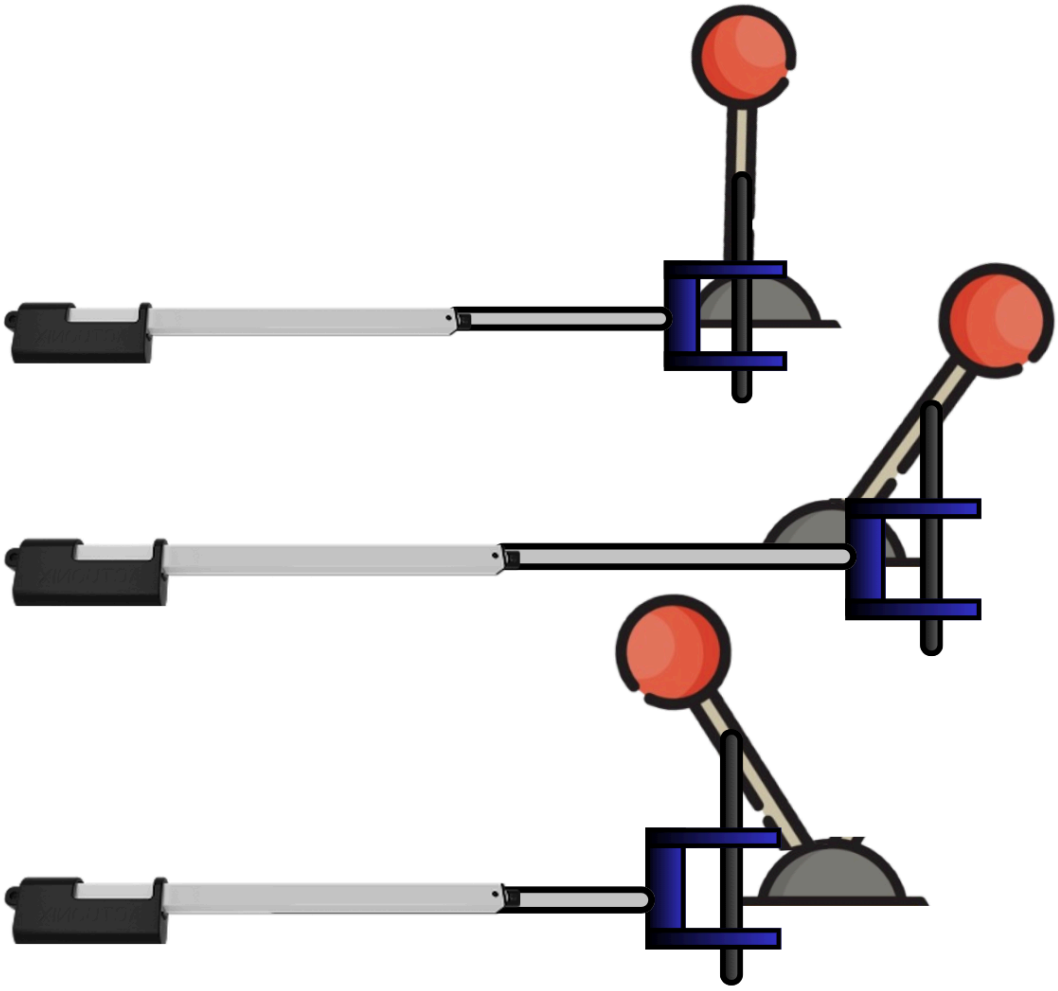
5. Distance-Based Control:

The rotary encoder mounted on the wheel calculates the distance traveled by the machine.

When the preset distance (e.g., 7.83m or 8.37m) is reached, the Arduino signals the relay module to retract the actuators, returning the levers to the Neutral (N) position and halting the machine.

Motion Transfer mechanism:

The linear actuator is securely mounted using clevis mounts, which provide a pivoting connection for smooth motion transfer. The actuator's shaft is linked to the clevis, which is attached to the lever arm, ensuring efficient force transmission. As the actuator extends or retracts, the clevis mount pivots to accommodate the lever's motion, moving it between Neutral, Forward, and Reverse positions. This setup ensures precise, repeatable lever operation while minimizing mechanical stress and enhancing system reliability through accurate and controlled motion.



Neutral, Forward & Reverse Motions

Lever Stroke Length Calculations:

Below are the stroke lengths and calculations used in the system:

1. Direction Lever (DL)

The Direction Lever (DL) is responsible for controlling the forward and reverse movements of the system. The forward and reverse stroke lengths are calculated using the formula:

$$\text{Stroke Length} = \text{Length} \times \cos(\theta)$$

Where:

- **Length** is the maximum stroke length of the lever (in mm).
- **θ** is the angle at which the lever moves, measured in degrees from the horizontal axis.

For the Direction Lever, we have the following stroke lengths:

- *Forward Stroke (FD):*

  $FD = 100 \times \cos(35.94°)$
- *Reverse Stroke (RD):*

  $RD = 100 \times \cos(-29.73^\circ)$

Neutral Position of the Direction Lever:

$$\text{Neutral Position (DL\_Neutral)} = (FD + RD)/2$$

This neutral position ensures that the lever is not at zero mm (midway between forward and reverse), allowing for smooth transitions in both directions.

2. Vibration Lever (VBL)

The Vibration Lever (VBL) controls a different function, but the same principle is used for calculating its forward and reverse stroke lengths. Using the same cosine formula:

- *Forward Stroke (FD):*

  $FD=60\times\cos(43.97°)$
- *Reverse Stroke (RD):*

  $RD=60\times\cos(-43.97\circ)$

Neutral Position of the Vibration Lever:

$$\text{Neutral Position (VBL\_Neutral)} = (FD+RD)/2$$

This ensures the vibration lever also has a non-zero neutral position, facilitating both push and pull movements.

Code Constants and Variables:

The constants and variables used for the lever stroke length calculations and the operation of the system are defined in the code. These constants can be adjusted to match the specific design or functional requirements of the system.

- **Angle Constants**: The angles used in the calculations for both levers (35.94°, -29.73° for the direction lever, and 43.97°, -43.97° for the vibration lever) can be modified in the code depending on the desired stroke length and functional behavior of the system.
- **Stroke Length Constants**: The maximum stroke length (100 mm for the direction lever and 60 mm for the vibration lever) can be changed as needed.

- **Neutral Positions**: The neutral positions for both levers are dynamically calculated using the forward and reverse stroke lengths. This value ensures that the lever is positioned in a neutral state between forward and reverse movements.

Sequence of Operation:

1. Relay 3 HIGH (VBM1) for 2 seconds
2. Relay 1 HIGH (Forward direction) for covering the forward distance
3. Relay 3 LOW (Neutral position)
4. Relay 1 LOW (Neutral position)
5. Pause for 5 seconds
6. Relay 4 HIGH (VBM2) for 2 seconds
7. Relay 2 HIGH (Reverse direction) for covering the reverse distance
8. Relay 4 LOW (Neutral position)
9. Relay 2 LOW (Neutral position)

CODE:

```
// Pin Assignments
#define RELAY1_PIN 22  // Forward direction of DL
#define RELAY2_PIN 23  // Reverse direction of DL
#define RELAY3_PIN 24  // Forward direction of VBL (VBM1)
#define RELAY4_PIN 25  // Reverse direction of VBL (VBM2)
#define ENCODER_PIN A0 // Encoder for distance tracking
// Constants
const float SPEED = 4.5 * 1000 / 3600;  // Convert speed from km/h to m/s
const float DISTANCE1 = 7.87;          // First forward distance in meters
const float DISTANCE_REVERSE = 8.37;    // Reverse distance in meters
const float L2 = 2.0;                  // Adjust based on requirement
// Stroke Length Calculation for Direction Lever (DL) and Vibration Lever (VBL)
const float DL_FD = 100 * cos(35.94 * PI / 180); // Forward stroke length for DL
const float DL_RD = 100 * cos(-29.73 * PI / 180); // Reverse stroke length for DL
const float VBL_FD = 60 * cos(43.97 * PI / 180);  // Forward stroke length for VBL
const float VBL_RD = 60 * cos(-43.97 * PI / 180); // Reverse stroke length for VBL
// Neutral Positions for both levers
const float DL_Neutral = (DL_FD + DL_RD) / 2; // Neutral position for DL
const float VBL_Neutral = (VBL_FD + VBL_RD) / 2; // Neutral position for VBL
// Variables
volatile float distanceTravelled = 0.0;
unsigned long lastTime = 0;
```

```
// LCD setup (I2C, 16x2)
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
// Function to simulate encoder pulse counting
void encoderInterrupt() {
  static int pulseCount = 0;
  float wheelCircumference = 0.3; // Adjust as per your wheel diameter
  int pulsesPerRevolution = 360;  // Encoder ticks per revolution
pulseCount++;
  distanceTravelled = (pulseCount * wheelCircumference) / pulsesPerRevolution;
}
void setup() {
  // Setup relay pins
  pinMode(RELAY1_PIN, OUTPUT);
  pinMode(RELAY2_PIN, OUTPUT);
  pinMode(RELAY3_PIN, OUTPUT);
  pinMode(RELAY4_PIN, OUTPUT);
 // Initialize relays to LOW (neutral position)
  digitalWrite(RELAY1_PIN, LOW);
  digitalWrite(RELAY2_PIN, LOW);
  digitalWrite(RELAY3_PIN, LOW);
  digitalWrite(RELAY4_PIN, LOW);
```

```
// Setup encoder pin and interrupt
  pinMode(ENCODER_PIN, INPUT);
  attachInterrupt(digitalPinToInterrupt(ENCODER_PIN), encoderInterrupt, RISING);
// Initialize serial monitor
  Serial.begin(9600);
// Initialize LCD
  lcd.begin();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("System Ready");
  delay(2000); // Display "System Ready" for 2 seconds
  lcd.clear();
}
void loop() {
  // Step 1: Machine ON, initialize
  delay(2000); // Simulate start delay
 // Step 2: Relay 3 HIGH (VBM1), wait 2 seconds
  digitalWrite(RELAY3_PIN, HIGH);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("VBM1 ON");
  delay(2000);
```

```
  // Step 3: Relay 1 HIGH (Forward), track distance
  distanceTravelled = 0.0; // Reset distance
  digitalWrite(RELAY1_PIN, HIGH);
 lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Moving Forward");
while (distanceTravelled < DISTANCE1) {
    // Wait until the machine covers 7.87m
    lcd.setCursor(0, 1);
  lcd.print("Dist: ");
    lcd.print(distanceTravelled);
    lcd.print(" m");    delay(100); // Small delay to allow encoder to count pulses
  }
 // Turn OFF Relay 3 and Relay 1 (Neutral position)
  digitalWrite(RELAY3_PIN, LOW);
  digitalWrite(RELAY1_PIN, LOW);
// Step 4: Pause for 5 seconds
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Pause");
  delay(5000);
// Step 5: Relay 4 HIGH (VBM2), wait 2 seconds
  digitalWrite(RELAY4_PIN, HIGH);
```

```
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("VBM2 ON");
  delay(2000);
// Step 6: Relay 2 HIGH (Reverse), track distance
  distanceTravelled = 0.0; // Reset distance
  digitalWrite(RELAY2_PIN, HIGH);
lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Moving Backward");
while (distanceTravelled < DISTANCE_REVERSE) {
    // Wait until the machine covers 8.37m
    lcd.setCursor(0, 1);
    lcd.print("Dist: ");
    lcd.print(distanceTravelled);
    lcd.print(" m");
    delay(100); // Small delay to allow encoder to count pulses
  }
// Turn OFF Relay 4 and Relay 2 (Neutral position)
  digitalWrite(RELAY4_PIN, LOW);
  digitalWrite(RELAY2_PIN, LOW);
 // Loop ends, reset system or add further steps if needed
  lcd.clear();
```

```
  lcd.setCursor(0, 0);
  lcd.print("Operation Complete");
  delay(2000); // Display "Operation Complete" for 2 seconds
  lcd.clear();
  // Optionally print stroke lengths and neutral positions for debugging
  Serial.print("DL Forward Stroke: ");
  Serial.println(DL_FD);
  Serial.print("DL Reverse Stroke: ");
  Serial.println(DL_RD);
  Serial.print("DL Neutral Position: ");
  Serial.println(DL_Neutral);

  Serial.print("VBL Forward Stroke: ");
  Serial.println(VBL_FD);
  Serial.print("VBL Reverse Stroke: ");
  Serial.println(VBL_RD);
  Serial.print("VBL Neutral Position: ");
  Serial.println(VBL_Neutral);
}
```

APPROACH 3

ScotchDrive: Servo-Enhanced Motion

Description:

In the WBR project, the ScotchDrive mechanism is implemented to control the Directional Lever (DL) and Vibration Lever (VB) with precision. A servo motor generates controlled rotary motion, which is converted into linear motion using a Scotch yoke assembly. This linear motion is then mechanically linked to the levers through precision-engineered connections.This servo-enhanced setup ensures precise, repeatable operation of both levers, optimizing performance while maintaining mechanical simplicity and reliability.

Key Features:

- **Precision Lever Control:**

The servo-enhanced Scotch yoke mechanism ensures accurate and repeatable positioning of the Directional Lever (DL) and Vibration Lever (VB), enabling smooth transitions between Neutral, Forward, and Reverse positions.

- **Efficient Motion Conversion:**

The rotary motion of the servo is efficiently converted into linear motion via the Scotch yoke, minimizing energy loss and ensuring consistent lever operation with reduced mechanical complexity.

- **Programmable Flexibility:**

The servo motor's programmability allows for real-time adjustments to lever positions, providing adaptability for varied operational requirements and ensuring optimal machine performance.

Components Required:

1. Arduino Mega:

Acts as the central processing unit for the control system.

Executes commands to operate the stepper motors and monitors encoder feedback.

2. 2 x Stepper Motors (High Torque):

High-torque servo motors for precise motion control of the Directional and Vibration Levers.

3. 2 x Scotch Yoke Mechanism:

Custom-built to translate rotary motion into linear motion.

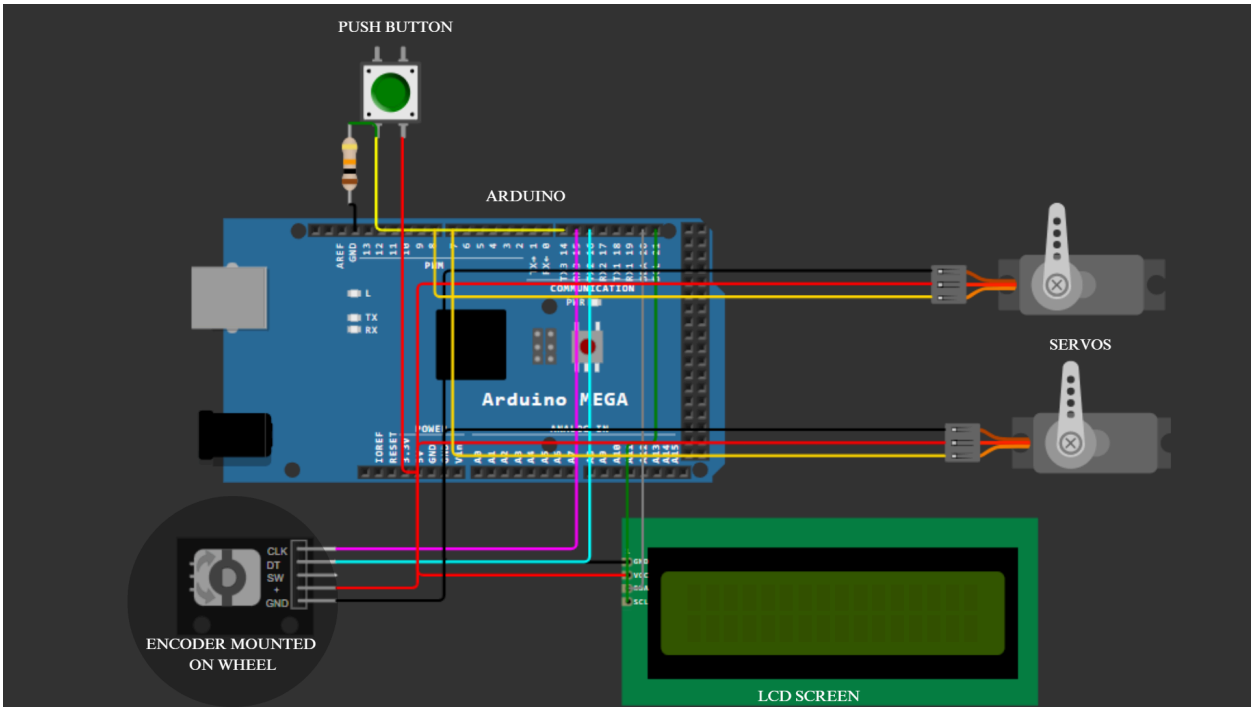
Designed for durability and smooth operation in high-torque applications.

4. Magnetic Rotary Encoder:

Provides real-time feedback on rotary motion.

5. Power Supply:

Delivers a stable and compatible power source for the stepper motors and other components.



Circuit Diagram

Working:

1. Initialization Phase:

The Arduino Mega initializes the system by setting the servo motors to their default positions via the Scotch yoke mechanism, positioning the levers (DL and VB) in Neutral (N).

The connected LCD displays the system status, such as "System Ready."

2. Command Execution:

Based on predefined user inputs via the keypad, the Arduino sends control signals to the servo motors.

The servo motors rotate as commanded, driving the Scotch yoke mechanism to translate rotary motion into linear motion.

3. Scotch Yoke Mechanism Operation:

The Scotch yoke mechanism converts the servo's rotary motion into linear motion, which is transmitted to the levers through precision-engineered linkages.

The DL lever is moved to Forward (F) for forward motion or Reverse (R) for reverse motion.

The VB lever toggles between Forward (F) for Vibration Mode 1 (VBM1) or Reverse (R) for Vibration Mode 2 (VBM2).

4. Real-Time Monitoring and Feedback:

A magnetic rotary encoder mounted on the wheel continuously monitors the machine's travel distance.

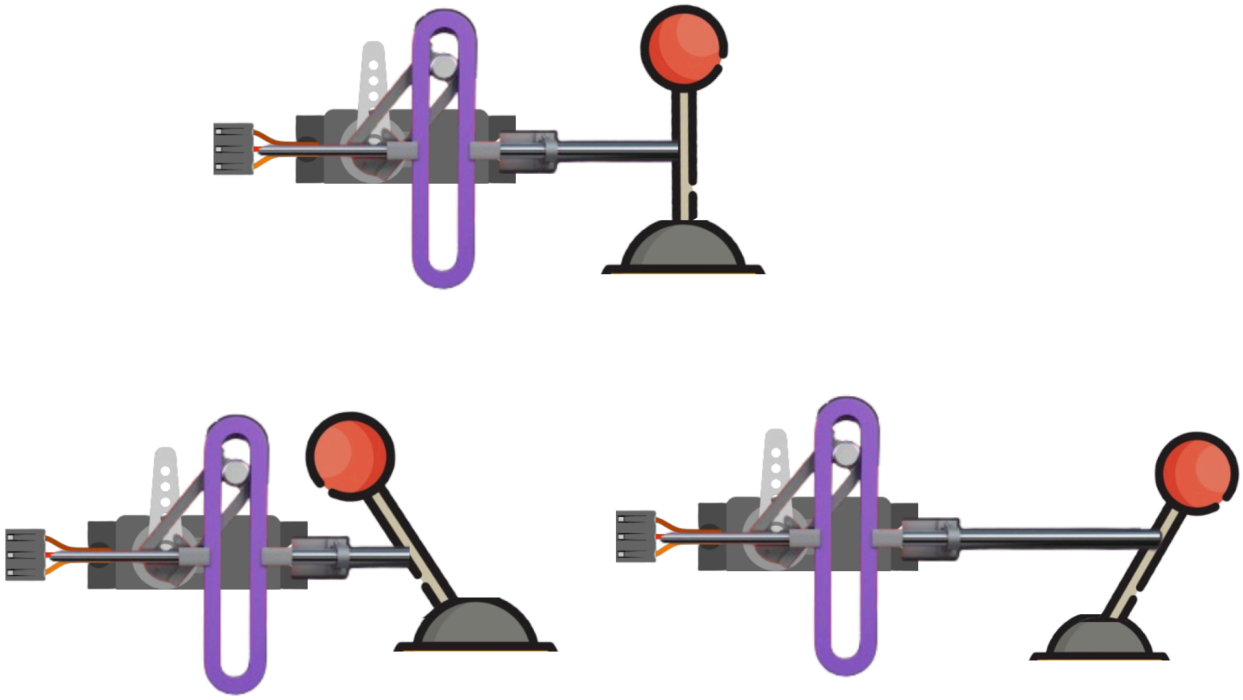
Feedback from the encoder ensures accurate tracking and distance measurement.

5. Distance-Based Control:

When the machine travels the preset distance (e.g., 7.83m or 8.37m), the Arduino commands the servo motors to return the levers to the Neutral (N) position via the Scotch yoke mechanism.

Motion Transfer Mechanism:

The servo motor will be linked to the Scotch yoke mechanism, where its rotary motion is converted into precise linear motion. The Scotch yoke assembly, connected to the servo, will transfer this linear motion through a set of precision-engineered linkages. These linkages connect the yoke to the Directional Lever (DL) and Vibration Lever (VB), ensuring smooth, accurate, and repeatable lever movement. This setup enables efficient and controlled motion transfer, allowing the levers to be positioned accurately for directional and vibration control of the machine.



Neutral, Forward & Reverse Motions

CODE

```
#include <Servo.h>

#include <Wire.h>

#include <LiquidCrystal_I2C.h>

// Define the pins for the servos

#define DL_servo_pin 9      // Direction Lever Servo Pin

#define VB_servo_pin 10     // Vibration Lever Servo Pin

#define encoder_pin_A 2     // Encoder A Pin

#define encoder_pin_B 3     // Encoder B Pin

#define button_pin 8        // Push Button Pin

// Create Servo objects

Servo DL_servo;          // Direction Lever Servo

Servo VB_servo;          // Vibration Lever Servo

// Set up LCD display (Address 0x27, 16 columns, 2 rows)

LiquidCrystal_I2C lcd(0x27, 16, 2);

// Encoder variables

volatile int encoder_pos = 0;    // Encoder position

int last_encoder_pos = 0;        // Last encoder position to track distance

long distance_travelled = 0;     // To store the distance

// Button state variables

bool last_button_state = LOW;

bool button_state = LOW;

bool system_running = false;    // Track system state
```

```
// Define servo positions
int neutral_pos = 90;
int forward_pos = 180;
int reverse_pos = -180;
int vibration_pos_1 = 180;
int vibration_pos_2 = -180;
void setup() {
  // Initialize the servos
  DL_servo.attach(DL_servo_pin);
  VB_servo.attach(VB_servo_pin);
  // Set initial servo positions
  DL_servo.write(neutral_pos);
  VB_servo.write(neutral_pos);
 // Setup Encoder
  pinMode(encoder_pin_A, INPUT);
  pinMode(encoder_pin_B, INPUT);
  attachInterrupt(digitalPinToInterrupt(encoder_pin_A), encoder_ISR, RISING);
// Setup Push Button
  pinMode(button_pin, INPUT);
// Initialize LCD
  lcd.begin();
  lcd.backlight();
  lcd.setCursor(0, 0);
```

```
  lcd.print("System Ready");
  delay(1000);
}
void loop() {
  // Read button state
  button_state = digitalRead(button_pin);
  // Toggle system start/stop with button
  if (button_state == HIGH && last_button_state == LOW) {
    system_running = !system_running;
    if (system_running) {
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Operation Start");
      start_operation();
    } else {
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("System Stopped");
      DL_servo.write(neutral_pos);
      VB_servo.write(neutral_pos);
    }
    delay(500);  // Debounce delay
  }
```

```
  last_button_state = button_state;
 // Update LCD with distance traveled
  if (system_running) {
    distance_travelled = encoder_pos / 7.83;  // Assuming encoder counts correspond to 7.83m of
distance
    lcd.setCursor(0, 1);
    lcd.print("Dist: ");
    lcd.print(distance_travelled);
    lcd.print(" m");
  }
}void encoder_ISR() {
  int MSB = digitalRead(encoder_pin_A);  // Most significant bit
  int LSB = digitalRead(encoder_pin_B);  // Least significant bit

  int encoded = (MSB << 1) | LSB;       // Combine MSB and LSB
  int sum = (encoder_pos << 2) | encoded;  // Add to current position

  // Update encoder position
  if (sum == 0b1101 || sum == 0b0110 || sum == 0b1010 || sum == 0b0101) {
    encoder_pos++;   // Clockwise rotation
  } else {
    encoder_pos--;   // Counter-clockwise rotation
  }
```

```
}
void start_operation() {
  // Move Direction Lever to Forward (180°) and Vibration Mode 1 (180°)
  DL_servo.write(forward_pos);
  VB_servo.write(vibration_pos_1);
  delay(500);  // Wait for servos to reach positions
 // Monitor distance using encoder, stop when target distance is reached
  while (encoder_pos / 7.83 < 7.83) {  // Example: 7.83 meters
    // Continue until the target distance is reached
  }
  // Reset servos to Neutral (90°)
  DL_servo.write(neutral_pos);
  VB_servo.write(neutral_pos);
  delay(500);
  // Move Direction Lever to Reverse (-180°) and Vibration Mode 2 (-180°)
  DL_servo.write(reverse_pos);
  VB_servo.write(vibration_pos_2);
  delay(500);  // Wait for servos to reach positions
// Monitor distance again in reverse direction
  while (encoder_pos / 7.83 < 8.37) {  // Example: 8.37 meters in reverse
    // Continue until the target distance is reached
  }
```

```
  // Reset servos to Neutral (90°)
  DL_servo.write(neutral_pos);
  VB_servo.write(neutral_pos);
  delay(500);
// Operation Complete
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Operation Complete");
  delay(1000);
}
```

Comparison of Three Approaches: A Detailed Analysis

1. ***YokeDrive: Stepper-Based Automation***

Component Criticality Chart

Components	Failure Scenario (EFFECT)	Failure Mode (FUNCTIONALITY)	Failure Mode (SAFETY)
ARDUINO	System fails to execute any logic; entire operation halts.	Loss of control over input/output; program crash or error.	FAIL SECURE
STEPPER MOTOR DRIVER (Directional Lever)	Loss of directional movement; inability to steer.	Motor stalls or moves erratically; loss of precision.	NOT FAIL SECURE
STEPPER MOTOR DRIVER (Vibrational Lever)	Vibration control fails; uneven operation of the machine.	Erratic or no response to signals; vibration pattern failure.	FAIL SECURE
STEPPER MOTOR (Directional Lever)	Inability to move the directional lever.	Motor stalls, reduces torque, or skips steps.	NOT FAIL SECURE
STEPPER MOTOR (Vibrational Lever)	Vibration lever becomes non-functional.	Loss of synchronization and becomes non functional	FAIL SECURE
SCOTCH YOKE MECHANISM (For Directional Lever)	Failure to convert motor rotation into linear motion; lever remains static.	Misalignment or breakage in the yoke or pin.	FAIL SECURE
SCOTCH YOKE MECHANISM (For Vibrational Lever)	Loss of vibration control; uneven machine operation.	Wear and tear causing movement failure or inefficiency.	FAIL SECURE
MAGNETIC ROTARY ENCODER (Real time Displacement Monitoring)	Loss of position monitoring; feedback becomes unreliable.	Incorrect displacement data sent to Arduino; system errors.	NOT FAIL SECURE

PRICE LIST		
COMPONENT	QUANTITY	PRICE
Arduino Mega	1	1370
TB6600 Stepper Motor Driver	2	1262
Nema 23 Stepper Motor	2	2278
AMS AS5600 Magnetic Encoder	1	350
LCD with I2C Module	1	187
Push Button Button	1	56
TOTAL(Rs.)		**5,503**

2. Actuators: Relay-Driven Automation

Component Criticality Chart

Components	Failure Scenario (EFFECT)	Failure Mode (FUNCTIONALITY)	Failure Mode (SAFETY)
ARDUINO	System fails to execute any logic; entire operation halts.	Loss of control over input/output; program crash or error.	FAIL SECURE
4 CHANNEL RELAY MODULE	Relays fail to switch; inability to control actuators	Stuck in open or closed state; intermittent switching.	NOT FAIL SECURE
LINEAR ACTUATOR (Directional Lever)	Inability to move the directional lever; operation stalls.	Motor stalls; reduced stroke length or no motion.	NOT FAIL SECURE
LiINEAR ACTUATOR (Vibrational Lever)	Loss of vibration control; lever does not function as intended.	Inconsistent or no motion; reduced efficiency.	FAIL SECURE

CLEVIS MOUNTING (Directional lever)	Misalignment or detachment; directional actuator becomes ineffective.	Mechanical wear or failure; lever unable to stay in position.	NOT FAIL SECURE
CLEVIS MOUNTING (Vibrational lever)	Misalignment or detachment; vibrational actuator becomes ineffective.	ineffective. Mechanical wear or failure; vibration compromised.	FAIL SECURE
MAGNETIC ROTARY ENCODER (Real time Displacement Monitoring)	Loss of position monitoring; feedback becomes unreliable.	Incorrect displacement data sent to Arduino; system errors.	NOT FAIL SECURE

PRICE LIST		
COMPONENT	QUANTITY	PRICE
Arduino Mega	1	1370
Actuonix L16-100-150-6R	2	20250
HiLetgo 4-Channel 5V Relay Module	1	136
AMS AS5600 Magnetic Encoder	1	350
LCD with I2C Module	1	187
Push Button Button	1	56
TOTAL(Rs.)		**22,859**

3. *ScotchDrive: Servo-Enhanced Motion*

Component Criticality Chart

Components	Failure Scenario (EFFECT)	Failure Mode (FUNCTIONALITY)	Failure Mode (SAFETY)
ARDUINO	System fails to execute any logic; entire operation halts.	Loss of control over input/output; program crash or error.	FAIL SECURE
SERVO MOTOR (Directional Lever)	Inability to move the directional lever; loss of control in direction.	Motor stalls or moves erratically; reduced precision.	NOT FAIL SECURE
SERVO MOTOR (Vibrational Lever)	Vibration control fails; uneven machine operation or no vibration.	Erratic movement or complete failure; motor fails to respond.	FAIL SECURE
SCOTCH YOKE MECHANISM (Directional Lever)	Failure to convert rotary motion into linear motion; lever remains static.	Mechanical misalignment or pin breakage.	NOT FAIL SECURE
SCOTCH YOKE MECHANISM (Vibrational Lever)	Loss of vibration control; uneven operation of the machine.	Wear and tear causing motion failure or inefficiency.	FAIL SECURE
MAGNETIC ROTARY ENCODER (Real time Displacement Monitoring)	Loss of position monitoring; feedback becomes unreliable.	Incorrect displacement data sent to Arduino; system errors.	NOT FAIL SECURE

PRICE LIST		
COMPONENT	QUANTITY	PRICE
Arduino Mega	1	1370
OT5320M 7.4V 20kg.cm 180° Copper Metal Gear Digital Servo Motor	2	2782
AMS AS5600 Magnetic Encoder	1	350
LCD with I2C Module	1	187
Push Button Button	1	56
TOTAL(Rs.)		**4,745**

Conclusion

YokeDrive: Stepper-Based Automation

The YokeDrive system emphasizes precision with high-torque stepper motors coupled with a custom-built Scotch yoke mechanism. It is best suited for applications requiring finely tuned movement and precise control over lever positions.

Actuators: Relay-Driven Automation

This approach relies on linear actuators controlled via relays to automate lever movement. The simplicity of the design ensures reliability and ease of implementation. It offers a cost-effective and robust solution for automating the WBR's levers, making it suitable for scenarios where budget and durability are priorities.

ScotchDrive: Servo-Enhanced Motion

ScotchDrive integrates precision servo motors with a Scotch yoke mechanism for smooth and responsive lever operation. Its adaptive control capabilities and enhanced performance make it ideal for tasks requiring a higher degree of automation and versatility.

Comparison

A detailed component criticality chart highlights the advantages and challenges of each approach. While YokeDrive and ScotchDrive (Ideas 1 and 3) are the most feasible options, ScotchDrive stands out as a less complex yet effective solution compared to YokeDrive. Its balance of adaptability, performance, and ease of implementation makes it a compelling choice for the WBR's automation needs.

.

References

- https://www.youtube.com/user/mcwhorpj
- https://www.youtube.com/c/MichaelReeves
- https://robu.in/