**Summer 2024**

# A NEW APPROACH FOR LIFTING HEAVY OBJECTS USING AN AUTONOMOUS ROBOT

Student Name 1: Ali Qasim
Student Name 2: Ayaan Farazi

Advisor:
Syedah Zahra Atiq,
Assistant Professor of Practice of Computer Science and Engineering
Ohio State University

Project Supervisor(s):
Areej Aslam,
Senior, Electrical Engineering, LUMS

# A NEW APPROACH FOR LIFTING HEAVY OBJECTS USING AN AUTONOMOUS ROBOT

## Table of Contents

# Abstract

This report examines the development of an autonomous robot capable of automating a tree plantation system. The robot is designed to remove obstacles, such as rocks or boulders, from designated coordinates and then travel to different locations to collect trees. After collecting the trees, the robot returns to plant them at the assigned coordinates. It is also capable of determining the shortest path in the x and y plane for efficient navigation.

Additionally, the robot can revisit previously traversed coordinates to place objects as necessary. Once its tasks are completed, the robot autonomously parks itself.

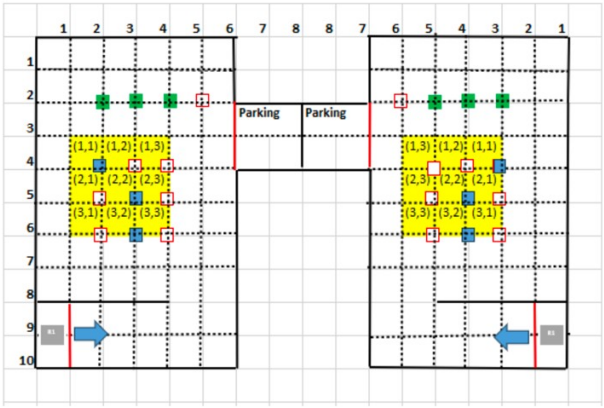
The aim for this research was development of design and simulation of a unique lifting mechanism, that could simultaneously lift multiple objects. Also to develop algorithm for shortest path depending on information of current node, i.e the coordinate.

# Step 1: Components used

| Component | Model |
|---|---|
| Development Board | Arduino mega 2560 |
| Motor driver | LUMS' house made motor driver |
| Battery | Lithium Phosphate (7s) |
| IR sensor | Flying fish |
| Servo | Mg946r 180 degree |
| Battery pack | 1s lithium phosphate |

## Objective:

This project aims to implement an algorithm for determining the shortest path to efficiently remove obstacles, such as boulders and soil, and transport them away from the plantation site. Simultaneously, the robot will bring trees back to the designated locations for planting. This approach minimizes unnecessary back-and-forth movement, streamlining the entire process. This small-scale model can serve as a foundational step towards automating plantation tasks, with potential applications across various industries in the future.
The project is inspired by NERC Pakistan (National Engineering Robotic Contest) and hence uses similar design and parameter constraints of arena to depict plantation sight.




The map illustrates the plantation site, where coordinate (9,1) serves as the starting point. The yellow blocks, consisting of 9 points, represent the potential coordinates where obstacles may be located. The green blocks at coordinates (2,2), (2,3), (2,4) in the left section, and (2,3), (2,4), (2,5) indicate tree positions. The task for the autonomous robot is to remove obstacles from the yellow blocks in either the left or right arena, collect trees from their designated positions, and then place the obstacles where the trees were originally located. Finally, the robot is to move the trees to the coordinates where the obstacles were picked up, and upon completion, autonomously park itself.

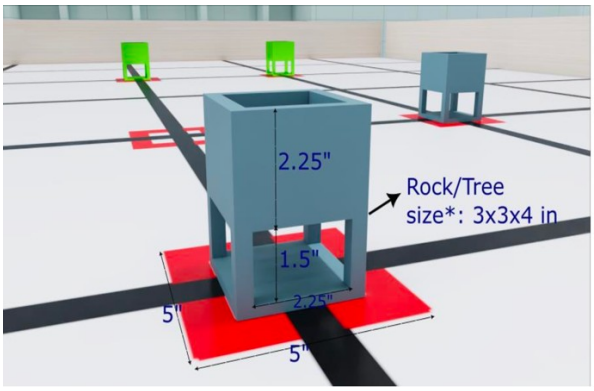



*Figure of tree / rock*

## Structure:

- Designed Robot chassis to be of 12 inches in width since on block on the grid is of 12 inches, did this to ensure better turning of the robot.
- Stimulated robot, its lifting and dropping mechanism on SolidWorks.
- Designed PCB (Printed Circuit board) to include servos and sensors.
- Created prototype of lifting and dropping mechanism, checking if servos could handle the torque required to lift obstacles.
- Tested PCB and assembled the robot.

## Methodology and Problems:

To implement the navigation and pathfinding for the autonomous robot, the following algorithmic structure was employed using IR sensors and an Arduino-based control system. The goal was to achieve precise line-following movement, both forward and reverse, and to execute smooth 90-degree turns, ensuring efficient movement along the grid.

1. **Sensor Setup and Calibration:**

**Line Following**: Two IR sensors were placed at the front of the robot to detect lines for forward movement, and two IR sensors were mounted at the back for reverse movement. Additionally, two IR sensors were installed on the sides of the robot to detect when it was perfectly aligned at the center of a junction. These side sensors read the black line when the robot was positioned in the middle of a crossroad, allowing for accurate turns and rotations.

**PWM Tuning:** Through iterative testing of various Pulse Width Modulation (PWM) values for the left and right motors, the robot's speed and directional control were optimized. The values were fine-tuned to achieve a balanced forward and reverse line following, as well as smooth 90-degree rotations.

2. **Perfecting Turns and Rotations**:
The robot's ability to turn was critical to its success in navigating the grid. A series of experiments were conducted to ensure accurate 90-degree turns in both clockwise and counterclockwise directions. These turns allowed the robot to transition from one path to another at junctions, ensuring efficient travel between coordinates.

3. **Defining Movement Directions**:
The algorithm incorporated four degrees of freedom: North, South, East, and West, based on the robot's orientation. The front of the robot was designated as facing North, and depending on the task, it could turn left or right to face the appropriate direction.

4. **Mapping the Arena**:
In this scenario, the robot navigated within a grid represented by coordinates. The yellow blocks in the arena contained 9 distinct coordinate points, which represented obstacles.

The robot was assigned a starting point at coordinate (4,4) in the left arena and (4,0) in the right arena. For simplicity in the code, these coordinates were renamed as (0,0), representing the robot's initial position.

5. **Pathfinding Logic**:
The task was to develop an algorithm for the shortest path between two coordinates on the grid.

At each point, the robot would assess whether its current x-coordinate was greater than or less than the target x-coordinate. If its x-coordinate was greater, the robot would turn left to face West and move in the x-axis by counting the number of junctions it crossed until it reached the target x-coordinate.

Similarly, if the y-coordinate needed to be adjusted, the robot would evaluate whether it was greater or smaller than the target y-coordinate. Based on this evaluation, the robot would adjust its orientation (North or South) and move forward, crossing junctions until the target y-coordinate was reached.

6. **Rotation and Movement Execution:**
The function `MoveTo` was developed to handle the robot's movement between two coordinates. This function used the fewest number of turns to optimize speed and efficiency during traversal. The robot continuously checked its position and orientation, turning as needed, and moving forward by crossing junctions until the desired location was reached.

7. **Example of Execution**:
For instance, if the robot needed to move from coordinate (0,0) to coordinate (3,1), First the robot would determine if its x-coordinate was larger or smaller than the target x-coordinate. In this case, it would turn left (facing West) since its current x-coordinate was greater than the required x-coordinate. Once aligned with the correct x-coordinate, it would check the y-coordinate. If the y-coordinate was smaller than the target y-coordinate, the robot would turn to face North and move forward until it reached the correct y-coordinate. This process ensured minimal turns and efficient movement across the grid. By following this systematic approach, the robot successfully navigated from point A to point B while avoiding obstacles and minimizing unnecessary turns or backtracking, thereby optimizing overall traversal time. The combination of well-placed sensors, precise motor control, and a robust pathfinding algorithm enabled the robot to perform its tasks with accuracy and efficiency.

# Code:

```
#define PWMR 3
#define dirR1 17
#define dirR2 16
#define PWML 2
#define dirL1 15
#define dirL2 14

#define Fleft 18
#define Fright 19
#define Mleft  20
#define Mright 21
#define Bleft  22
#define Bright 23
#define Bx     8

bool a=0;
bool b=0;
bool d=0;
bool e=0;
bool l=0;
bool m=0;
bool u=0;//bx
bool N=1;
bool S=0;
bool W=0;
bool E=0;

int count0=0;
int count1=0; //take to 00
int count2=0; //take to 00
int count3=0;
unsigned long time;




void Forward(){
  digitalWrite(dirR2,HIGH);
  digitalWrite(dirR1,LOW);
  analogWrite(PWMR, speedr+5);

  digitalWrite(dirL1,LOW);
  digitalWrite(dirL2,HIGH);
  analogWrite(PWML, speedr-15);
  }
void Stop(){
  digitalWrite(dirR2,HIGH);
  digitalWrite(dirR1,HIGH);
  analogWrite(PWMR, 255);

  digitalWrite(dirL1,HIGH);
  digitalWrite(dirL2,HIGH);
```

```
  analogWrite(PWML, 255);
  }
void Reverse(){
  digitalWrite(dirR2,LOW);
  digitalWrite(dirR1,HIGH);
  analogWrite(PWMR, speedr+5);

  digitalWrite(dirL1,HIGH);
  digitalWrite(dirL2,LOW);
  analogWrite(PWML, speedr-18);
  }
void SharpLeft(){
  digitalWrite(dirL1,HIGH);
  digitalWrite(dirL2,LOW);
  analogWrite(PWML, 55-5);
  digitalWrite(dirR1,LOW);
  digitalWrite(dirR2,HIGH);
  analogWrite(PWMR, 45);
  }
void SharpLeft2(){
  digitalWrite(dirL1,HIGH);
  digitalWrite(dirL2,LOW);
  analogWrite(PWML, 0);

  digitalWrite(dirR1,LOW);
  digitalWrite(dirR2,HIGH);
  analogWrite(PWMR, 55);
  }
void SharpRight(){
  digitalWrite(dirR1,HIGH);
  digitalWrite(dirR2,LOW);
  analogWrite(PWMR, 45);

  digitalWrite(dirL2,HIGH);
  digitalWrite(dirL1,LOW);
  analogWrite(PWML, 55);
  }
void SharpRight2(){
  digitalWrite(dirR1,HIGH);
  digitalWrite(dirR2,LOW);
  analogWrite(PWMR, 0);

  digitalWrite(dirL2,HIGH);
  digitalWrite(dirL1,LOW);
  analogWrite(PWML, 45);
  }
void SoftRight(){
  digitalWrite(dirL2,HIGH);
  digitalWrite(dirL1,LOW);
  analogWrite(PWML, 55-18);

  digitalWrite(dirR2,HIGH);
  digitalWrite(dirR1,LOW);
  analogWrite(PWMR, 0);
  }
void SoftLeft(){
  digitalWrite(dirR2,HIGH);
  digitalWrite(dirR1,LOW);
  analogWrite(PWMR, 50);

  digitalWrite(dirL2,HIGH);
  digitalWrite(dirL1,LOW);
  analogWrite(PWML, 0);

  }
void REVSoftLeft(){
  digitalWrite(dirL2,LOW);
  digitalWrite(dirL1,HIGH);
  analogWrite(PWML, 55-18);

  digitalWrite(dirR2,HIGH);
  digitalWrite(dirR1,LOW);
  analogWrite(PWMR, 0);
  }
void REVSoftRight(){
  digitalWrite(dirR2,LOW);
  digitalWrite(dirR1,HIGH);
  analogWrite(PWMR, 55);
```

```
  digitalWrite(dirL2,HIGH);
  digitalWrite(dirL1,LOW);
  analogWrite(PWML, 0);

  }
void READl(int PINnum){
  a=digitalRead(PINnum);
  int zeros=0;
  int ones=0;
  unsigned long time=millis();
  int reqMilli=1;
  while(millis()<time+reqMilli)
  {
    a=digitalRead(PINnum);
    if(!a)
    {
      zeros++;
    }
    else
    {
      ones++;
    }


  }
  if(zeros==ones)
  {
    READl(PINnum);
  }
  else if(zeros>ones)
  {
    a=0;
  }
  else if(ones>zeros)
  {
    a=1;
  }
}                   //middle left sensor a
void READFl(int PINnum) {
  d=digitalRead(PINnum);
  int zeros=0;
  int ones=0;
  unsigned long time=millis();
  int reqMilli=1;
  while(millis()<time+reqMilli)
  {
    d=digitalRead(PINnum);
    if(!d)
    {
      zeros++;
    }
    else
    {
      ones++;
    }


  }
  if(zeros==ones)
  {
    READl(PINnum);
  }
  else if(zeros>ones)
  {
    d=0;
  }
  else if(ones>zeros)
  {
    d=1;
  }
}                   //front left sensor d
void READBl(int PINnum){
  l=digitalRead(PINnum);
  int zeros=0;
  int ones=0;
  unsigned long time=millis();
  int reqMilli=1;
  while(millis()<time+reqMilli)
```

```
  {
   l=digitalRead(PINnum);
   if(!l)
   {
    zeros++;
   }
   else
   {
    ones++;
   }

  }
  if(zeros==ones)
  {
   READl(PINnum);
  }
  else if(zeros>ones)
  {
   l=0;
  }
  else if(ones>zeros)
  {
   l=1;
  }
}                 //back left sensor l
void READFr(int PINnum) {
  e=digitalRead(PINnum);
  int zeros=0;
  int ones=0;
  unsigned long time=millis();
  int reqMilli=1;
  while(millis()<time+reqMilli)
  {
   e=digitalRead(PINnum);
   if(!e)
   {
    zeros++;
   }
   else
   {
    ones++;
   }

  }
  if(zeros==ones)
  {
   READl(PINnum);
  }
  else if(zeros>ones)
  {
   e=0;
  }
  else if(ones>zeros)
  {
   e=1;
  }
}                 //front right sensor e
void READr(int PINnum) {
  b=digitalRead(PINnum);
  int zeros=0;
  int ones=0;
  unsigned long time=millis();
  int reqMilli=1;
  while(millis()<time+reqMilli)
  {
   b=digitalRead(PINnum);
   if(!b)
   {
    zeros++;
   }
   else
   {
    ones++;
   }

  }
  if(zeros==ones)
```

```
  {
   READl(PINnum);
  }
  else if(zeros>ones)
  {
   b=0;
  }
  else if(ones>zeros)
  {
   b=1;
  }
}                  //middle right sensor b
void READBr(int PINnum){
  m=digitalRead(PINnum);
  int zeros=0;
  int ones=0;
  unsigned long time=millis();
  int reqMilli=1;
  while(millis()<time+reqMilli)
  {
   m=digitalRead(PINnum);
   if(!m)
   {
    zeros++;
   }
   else
   {
    ones++;
   }


  }
  if(zeros==ones)
  {
   READl(PINnum);
  }
  else if(zeros>ones)
  {
   m=0;
  }
  else if(ones>zeros)
  {
   m=1;
  }
}                  //back right sensor m
void READBx(int PINnum){
  u=digitalRead(PINnum);
  int zeros=0;
  int ones=0;
  unsigned long time=millis();
  int reqMilli=1;
  while(millis()<time+reqMilli)
  {
   u=digitalRead(PINnum);
   if(!u)
   {
    zeros++;
   }
   else
   {
    ones++;
   }


  }
  if(zeros==ones)
  {
   READl(PINnum);
  }
  else if(zeros>ones)
  {
   u=0;
  }
  else if(ones>zeros)
  {
   u=1;
  }
}                  //back x sensor u
void LineFollow(){
```

```
  READFl(Fleft);
  READFr(Fright);
  if((e)&&(!d))
  {
    //softRight
    SharpRight2();
  }
  //left detects black
  else if ((d)&&(!e))
  {
    //soft left
    SharpLeft2();
  }
  //both on white, a rare case
  else if ((d)&&(e))///////
  {
    //sharp right so the bot finds the line while spinning
    Forward();
  }
  //both on black, means an intersection
  else if ((!d)&&(!e))
  {
    //softleft if u want to follow left side
    Forward();
    //SoftRight();

  }
 }
void REVLineFollow(){
  READBl(Bleft);
  READBr(Bright);
  if((m)&&(!l))
  {
    //softRight
    REVSoftLeft();
  }
  //left detects black
  else if ((l)&&(!m))
  {
    //soft left
    REVSoftRight();
  }
  //both on white, a rare case
  else if ((l)&&(m))///////
  {
    //sharp right so the bot finds the line while spinning
    Reverse();
  }
  //both on black, means an intersection
  else if ((!m)&&(!l))
  {
    //softleft if u want to follow left side
    Reverse();
  }
 }
void rotateleft(){
  Stop();
  delay(200);


  SharpLeft();
  delay(500);

  READFl(Fleft);
  while(!d)
  {
    READFl(Fleft);
  }
  READFl(Fleft);
  while(d)
  {
    READFl(Fleft);
  }
  Stop();
  delay(100);


 }
```

```
void rotateright(){


  Stop();
  delay(200);


  SharpRight();
  delay(500);

  READFr(Fright);
  while(!e)
  {
    READFr(Fright);
  }
  READFr(Fright);
  while(e)
  {
    READFr(Fright);
  }
  Stop();
  delay(100);


 }
void rotateleftR(){
  Stop();
  delay(200);
  Forward();
  delay(200);
  // READl(Mleft);
  //   while(!a)
  //   {
  //     LineFollow();
  //     READl(Mleft);
  //   }
  SharpLeft();
  delay(500);

  READBr(Bright);
  while(!m)
  {
    READBr(Bright);
  }
  READBr(Bright);
  while(m)
  {
    READBr(Bright);
  }
  Stop();
  delay(100);


 }
void rotaterightR(){
  Stop();
  delay(200);
  Forward();
  delay(100);

  SharpRight();
  delay(500);

  READBl(Bleft);
  while(!l)
  {
    READBl(Bleft);
  }
  READBl(Bleft);
  while(l)
  {
    READBl(Bleft);
  }
  Stop();
  delay(100);


 }
void TestMotors(){
```

```
   Forward();
  delay(5000);
  Stop();
  delay(2000);
  Reverse();
  delay(5000);
  Stop();
  delay(2000);
  REVSoftLeft();
  delay(5000);
  Stop();
  delay(2000);
  REVSoftRight();
  delay(5000);
 }
void startFFFF(){
  READI(Mleft);
  if(a)
  {
    count3++;
    time = millis();
    if(count3!=2)
    {
      while(millis()<time+300)
      {
        LineFollow();
      }
    }
    if(!(count3%2))
    {

    }
  }
  else
  {
    LineFollow();
  }
 }
void startFFF(){
  READI(Mleft);
  if(a)
  {
    count2++;
    time = millis();
    if(count2!=3)
    {
      while(millis()<time+300)
      {
        LineFollow();
      }
    }
    if(!(count2%3))
    {
      rotateleft();
      time = millis();
      while(millis()<time+600)
      {
        LineFollow();
      }
    }
  }
  else
  {
    LineFollow();
  }
 }
void startFF(){

  READI(Mleft);
  if(a)
  {
    count1++;
    time = millis();
    if(count1!=1)
    {
        while(millis()<time+300)
       {
```

```
        LineFollow();
       }
    }

    if(!(count1%1))
    {
     rotateleftR();
     time = millis();
     while(millis()<time+600)
     {
      LineFollow();
     }
    }
  }
  else
  {
   LineFollow();
  }
 }
void startF(){
  READI(Mleft);
  if(a)
  {
   count0++;
   time = millis();
   if(count0!=2)
   {
    while(millis()<time+300)
    {
     LineFollow();
    }
   }
   if(!(count0%2))
   {
    rotateright();
   }
  }
  else
  {
   LineFollow();
  }
 }
void start(){

  while(count0!=2){
   startF();
  }
  time = millis();
  while(millis()<time+2000)
  {
   REVLineFollow();
  }
  delay(500);
  for(int i=0; i<10;i++)
  {MapSense();}
  delay(5000);
  for (int i=0;i<10;i++){
   Serial.print(combo[i]);
  }


  while(count1!=1){
   startFF();
  }
  while(count2!=3){
   startFFF();
  }
  while(count3!=2){
   startFFFF();
  }

}
int differenceH(int x1,int x2){
  return abs(x1-x2);
}
int differenceV(int y1,int y2){
  return abs(y1-y2);
}
```

```
void MovetoPos(int x1,int y1,int x2,int y2)//current pos to new pos
{
  int distH=differenceH(x1,x2);
  int distV=differenceV(y1,y2);
  if(distH != 0)
  {
    if(x2>x1) //face w
    {
      W=1;
      if(N)
      {
        N=0;
        rotateleft();
      }
      else if (S)
      {
        S=0;
        rotateright();
      }
      else if(E)
      {
        E=0;
        rotateright();
        delay(500);
        rotateright();
      }
    }
    if(x1>x2)
    {
      E=1;
      if(N)
      {
        N=0;
        rotateright();
      }
      else if (S)
      {
        S=0;
        rotateleft();
      }
      else if(W)
      {
        W=0;
        rotateright();
        delay(500);
        rotateright();
      }
    }
    READBl(Bleft);
    READBr(Bright);
    while(l ==0 || m==0)
    {
     READBl(Bleft);
     READBr(Bright);
     LineFollow();
    }

    while(distH != 0)
    {

     LineFollow();
     READl(Mleft);
     if(a)
     {
       distH--;
       if(distH != 0)
       {
         READBl(Bleft);
         READBr(Bright);
         while(l ==0 || m==0)
         {
           READBl(Bleft);
           READBr(Bright);
           LineFollow();
         }
       }
     }
    }
```

```
}
//============================
if(distV != 0)
{
 if(y2>y1) //face w
 {
  N=1;
  if(E)
  {
   E=0;
   rotateleft();
  }
  else if (W)
  {
   W=0;
   rotateright();
  }
  else if(S)
  {
   S=0;
   rotateright();
   delay(500);
   rotateright();
  }
 }
 if(y1>y2)
 {
  S=1;
  if(W)
  {
   W=0;
   rotateleft();
  }
  else if (E)
  {
   E=0;
   rotateright();
  }
  else if(N)
  {
   N=0;
   rotateright();
   delay(500);
   rotateright();
  }
 }
 READBl(Bleft);
 READBr(Bright);
 while(l ==0 || m==0)
 {
  READBl(Bleft);
  READBr(Bright);
  LineFollow();
 }
 while(distV != 0)
 {
  LineFollow();
  READl(Mleft);
  if(a)
  {
   distV--;
   if(distV != 0)
   {
    READBl(Bleft);
    READBr(Bright);
    while(l ==0 || m==0)
    {
     READBl(Bleft);
     READBr(Bright);
     LineFollow();
    }
   }
  }
 }
}
READl(Mleft);
while(!a)
{
```

```
    REVLineFollow();
  }
  Stop();
  delay(100);
}
void MovetoPosR(int x1,int y1,int x2,int y2)//current pos to new pos
{
  int distH=differenceH(x1,x2);
  int distV=differenceV(y1,y2);
  if(distH != 0)
  {
    if(x2>x1) //face E
    {
      E=1;
      if(N)
      {
        N=0;
        rotaterightR();
      }
      else if (S)
      {
        S=0;
        rotateleftR();
      }
      else if(W)
      {
        W=0;
        rotaterightR();
        delay(500);
        rotaterightR();
      }
    }
    if(x1>x2)
    {
      W=1;
      if(N)
      {
        N=0;
        rotateleftR();
      }
      else if (S)
      {
        S=0;
        rotaterightR();
      }
      else if(E)
      {
        E=0;
        rotaterightR();
        delay(500);
        rotaterightR();
      }
    }
    READFl(Fleft);
    READFr(Fright);
    while(d ==0 || e==0)
    {
      READFl(Fleft);
      READFr(Fright);
      REVLineFollow();
    }
    while(distH != 0)
    {

      REVLineFollow();
      READl(Mleft);
      if(a)
      {
        distH--;
        if(distH != 0)
        {
          READFl(Fleft);
          READFr(Fright);
          while(d ==0 || e==0)
          {
            READFl(Fleft);
            READFr(Fright);
            REVLineFollow();
```

```
      }
     }
    }
   }
}
//============================
if(distV != 0)
{
 if(y2>y1) //face s
 {
  S=1;
  if(E)
  {
   E=0;
   rotaterightR();
  }
  else if (W)
  {
   W=0;
   rotateleftR();
  }
  else if(N)
  {
   N=0;
   rotaterightR();
   delay(500);
   rotaterightR();
  }
 }
 if(y1>y2)
 {
  N=1;
  if(W)
  {
   W=0;
   rotaterightR();
  }
  else if (E)
  {
   E=0;
   rotateleftR();
  }
  else if(S)
  {
   S=0;
   rotaterightR();
   delay(500);
   rotaterightR();
  }
 }
 READFl(Fleft);
 READFr(Fright);
 while(d ==0 || e==0)
 {
  READFl(Fleft);
  READFr(Fright);
  REVLineFollow();
 }
 while(distV != 0)
 {
  REVLineFollow();
  READl(Mleft);
  if(a)
  {
   distV--;
   if(distV != 0)
   {
    READFl(Fleft);
    READFr(Fright);
    while(d ==0 || e==0)
    {
     READFl(Fleft);
     READFr(Fright);
     REVLineFollow();
    }
   }
  }
 }
```

```
  }
  READI(Mleft);
  while(!a)
  {
    LineFollow();
  }
  Stop();
  delay(100);
}



void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);


  pinMode(PWML, OUTPUT);
  pinMode(dirL1, OUTPUT);
  pinMode(dirL2, OUTPUT);

  pinMode(PWMR, OUTPUT);
  pinMode(dirR1, OUTPUT);
  pinMode(dirR2, OUTPUT);


  pinMode(Mleft,INPUT);
  pinMode(Mright,INPUT);
  pinMode(Fleft,INPUT);
  pinMode(Fright,INPUT);
  pinMode(Bleft,INPUT);
  pinMode(Bright,INPUT);


}

void loop() {
  start();
  proj(combo);
  Stop();

  while(1)
  {
  }
}
```

# Expansion

The prototype designed for lifting is a three-arm mechanism controlled by servo motors. It consists of a total of five servo motors, with the middle arm functioning like a crane, operating at a 90-degree angle. The left and right arms can move along both the x and y axes. Initially, all three arms are positioned along the y+ axis, with the left arm aligned along the x- axis and the right arm along the x+ axis, ensuring no interference between them. Upon receiving the command, the middle arm moves downward to perform the lift and then returns to its original position. Afterward, either the left or right arm moves to the x0 position and then shifts downward along the y- axis to perform the pick-up before returning to its original position. The final arm follows the same procedure.

Although this model is innovative and convenient, its design, when scaled up, could result in a heavy and bulky lifting mechanism. This necessitates further research into making the structure lighter and optimizing the motors or redesigning the system to better fit space constraints. Despite these challenges, the mechanism offers a new approach to making lifting machinery more efficient and autonomous.

# A NEW APPROACH FOR LIFTING HEAVY OBJECTS USING AN AUTONOMOUS ROBOT

**Sub-report by:** **Ali Qasim**

## Software Report

As the student handling the software side of our robotic project, my role focused on developing the codebase that powered the robot's operations, ensuring smooth interaction between the software and hardware components. Here's a summary of my workflow, learning outcomes, and the overall value I brought to the project.

## Languages Used: C++

## Software Development Process

### 1. Code Structure and Logic Design

My first step was to establish a structured, modular codebase that could handle various functions independently. I split the code into sections for motor control, sensor reading, and decision-making algorithms. This allowed for easier troubleshooting and improved scalability.

### 2. Sensor Integration and Data Processing

A significant part of my work involved integrating sensors with the system. For example, if our robot used ultrasonic sensors for distance measurement, I coded algorithms to handle data processing from these sensors. The algorithms helped the robot to identify obstacles, adjust speed, and navigate accordingly.

### 3. Motor Control and Actuation

I developed the control logic for actuating motors, using functions that translated sensor data into actionable movements. This code ensured that the robot could move in a coordinated manner, reacting promptly to environmental changes.

### 4. Testing and Debugging

Throughout the project, I performed rigorous testing and debugging to ensure all components worked in sync. I collaborated closely with the hardware lead, troubleshooting issues at the intersection of hardware and software, and made adjustments to refine the robot's response.

## Learning Outcomes

This project was a valuable learning experience that enhanced my understanding of embedded systems and real-time programming. I developed stronger skills in debugging and optimizing code to ensure efficient memory usage and fast processing times, both of which are critical for embedded software. Additionally, working with hardware components taught me the intricacies of software-hardware integration, an essential skill for robotics and engineering.

## Value Added

Through my contributions, the project gained a robust and responsive software foundation that empowered the robot's functionalities. My modular approach to code structure allowed for straightforward troubleshooting, which made it easier to isolate and resolve issues as they arose. This not only improved the reliability of our system but also streamlined the collaboration process, as my teammate and I could efficiently test, debug, and implement changes.

## Appendix A: Robot Code and Explanations

This document contains the code segments used in the autonomous robot project, along with explanations for each block of code. These explanations clarify the purpose and functionality of each segment, making it easier to understand the robot's operations and the logic implemented.

### Define Constants and Global Variables

This section defines the constants and global variables used throughout the code, including the pin numbers for motor and sensor connections as well as boolean flags for directional control. This setup simplifies access to these variables and makes modifications easier if any connections change.

Code:

```
#define PWMR 3
#define dirR1 17
#define dirR2 16
#define PWML 2
#define dirL1 15
#define dirL2 14

#define Fleft 18
#define Fright 19
#define Mleft  20
#define Mright 21
#define Bleft  22
#define Bright 23
#define Bx     8

bool a=0; bool b=0; bool d=0; bool e=0; bool l=0; bool m=0; bool u=0;  // Sensor Flags
bool N=1; bool S=0; bool W=0; bool E=0;  // Direction Flags

int count0=0; int count1=0; int count2=0; int count3=0;  // Counters
unsigned long time;  // Time variable for delays
```

## Motor Control Functions

This section contains functions to control the robot's motors for different movements: forward, stop, reverse, sharp turns, and soft turns. These functions make the main loop easier to read and allow for reusable motor control instructions.

Code:

```
void Forward() {
  digitalWrite(dirR2,HIGH); digitalWrite(dirR1,LOW); analogWrite(PWMR, speedr+5);
  digitalWrite(dirL1,LOW); digitalWrite(dirL2,HIGH); analogWrite(PWML, speedr-15);
}

void Stop() {
  digitalWrite(dirR2,HIGH); digitalWrite(dirR1,HIGH); analogWrite(PWMR, 255);
  digitalWrite(dirL1,HIGH); digitalWrite(dirL2,HIGH); analogWrite(PWML, 255);
}

// Similar functions exist for Reverse, SharpLeft, SharpRight, etc.
```

## Sensor Reading Functions

These functions read the values from different sensors placed around the robot. Each function monitors a specific sensor and updates the boolean flags based on the readings. This enables the robot to interpret its environment and make decisions accordingly.

Code:

```
void READl(int PINnum) { /* Code to read the middle left sensor */ }
void READFl(int PINnum) { /* Code to read the front left sensor */ }
void READBl(int PINnum) { /* Code to read the back left sensor */ }
// Additional functions for other sensors are structured similarly.
```

## Line Following and Obstacle Avoidance

The `LineFollow` function uses readings from front and back sensors to adjust the robot's path along a designated line. The `REVLineFollow` function is similar but enables reverse line following, helping the robot move efficiently across the grid. These functions help the robot navigate the environment.

Code:

```
void LineFollow() { /* Code to control the robot based on line detection */ }
void REVLineFollow() { /* Code for reverse line following */ }
```

## Movement and Turning Functions

These functions help the robot perform specific movements such as rotating left or right. Each function manages the timing and execution of turns based on sensor readings to ensure accurate positioning.

Code:

```
void rotateleft() { /* Code to execute a left turn */ }
void rotateright() { /* Code for right turn */ }
// Functions like rotateleftR() and rotaterightR() handle reverse turns.
```

## Setup and Main Loop

The setup function initializes the pin modes for each component, setting pins as INPUT or OUTPUT as
required. The main loop initiates the robot's tasks, calling movement and sensor functions to operate the
robot based on its environment.

Code:

```
void setup() {
  Serial.begin(9600);
  pinMode(PWML, OUTPUT); pinMode(dirL1, OUTPUT); pinMode(dirL2, OUTPUT);
  pinMode(PWMR, OUTPUT); pinMode(dirR1, OUTPUT); pinMode(dirR2, OUTPUT);
  pinMode(Mleft, INPUT); pinMode(Mright, INPUT); pinMode(Fleft, INPUT);
  pinMode(Fright, INPUT); pinMode(Bleft, INPUT); pinMode(Bright, INPUT);
}

void loop() {
  start(); proj(combo); Stop();
  while(1) { /* Infinite loop after tasks complete */ }
}
```