Mechatronics Final Project
Firefighting Robot

Team 1

Kaitlyn Dohn, Jacob Munoz, Annamalai Muthupalaniappan, Ryan Schmidt

UD Mechanical Engineering

Instructor: Adam Wickenheiser

TABLE OF CONTENTS:

INTRODUCTION:

The goal of this project is to design an autonomous vehicle capable of finding and putting out building "fires" in a contained arena. The arena is an 8ft by 8ft square with randomly placed buildings, some of which are "on fire" and others that act as obstacles. The buildings are boxes with an opening on one side from which an infrared pulse is emitted. Fires are put out by the robot breaking a break beam sensor across the roof via a ladder attachment.

DESIGN DETAILS:

1. **Manufacturing:**

The final design features a rectangular 2-layer acrylic chassis supported by 4 wheels: 2 TT motor-driven wheels on the left and right sides and 2 ball bearings in the front and back to stabilize the vehicle and facilitate smooth rotational movement. The lower layer of the chassis is connected to the wheels and secures the majority of the circuitry, including the Arduino Mega, breadboard, AA battery holder, 9V battery, and the QTI sensors used for line detection [1]. The second layer of the chassis is raised by hex standoffs attached in the four corners. The top layer encloses all the wires and provides support for the ladder mechanism, which is a wooden dowel connected to a continuous rotation servo motor. This layer also supports the IR infrared sensors which are mounted at the approximate height of the IR sensors within the buildings. The non-autonomous version of the design features an NRF24 wireless radio that receives data from a separate Arduino and NRF24 radio combination controlled manually with a joystick [2].

The chassis was designed in SolidWorks and laser cut out of ¼ inch and ⅛ inch acrylic in the FabLab at Spencer Lab (Figure 1). Screw holes were measured and added for each of the components as well as some larger holes to reduce mass, increase airflow through the electronics, and for wire management and organized flow through layers. For ease of designing and ordering via the FabLab, two of the same design were printed for the two layers - one ¼ inch for the base layer and one ⅛ inch for the upper layer.

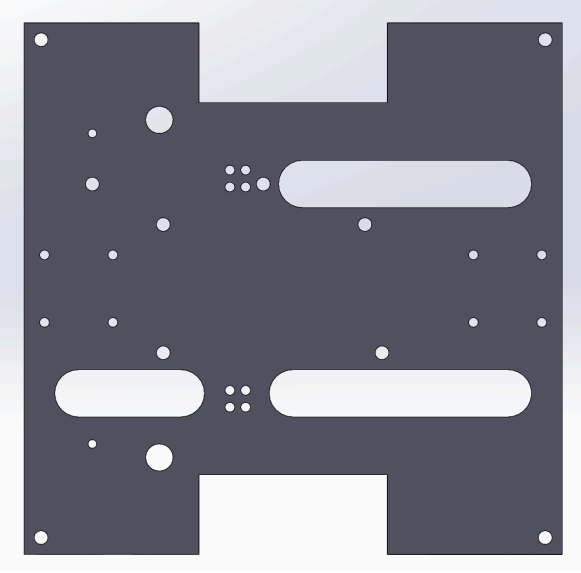

Figure 1: Screenshot from SolidWorks of custom chassis.

While the ball bearing wheels could be directly screwed into the chassis due to existing screw holes in their bases, additional mounts for the DC motors were necessary. These were custom-designed in SolidWorks and 3D printed out of PLA at the FabLab (Figure 2). This mount features 4 holes for screws to attach to the base as well as 2 wider holes to attach the motor to the mount.

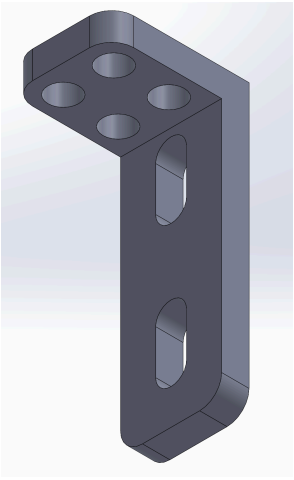

Figure 2: Screenshot from SolidWorks of custom DC motor mount.

The final design can be seen in Figure 3 and Figure 4. Figure 3 shows the front and side views with callouts to features. Figure 4 shows a top view of the vehicle. For the sake of organization, an Arduino Mega shield was utilized. The wires were securely fastened to the shield to prevent them from becoming disconnected. To control the two DC motors, an L298N driver controller (Table A.1) was employed. A distance sensor was incorporated into the autonomous code to detect nearby buildings. This sensor can change direction as it is mounted on a mini servo motor. The total cost of designing this vehicle was $244.46, with only $91.53 of the $300 budget spent.

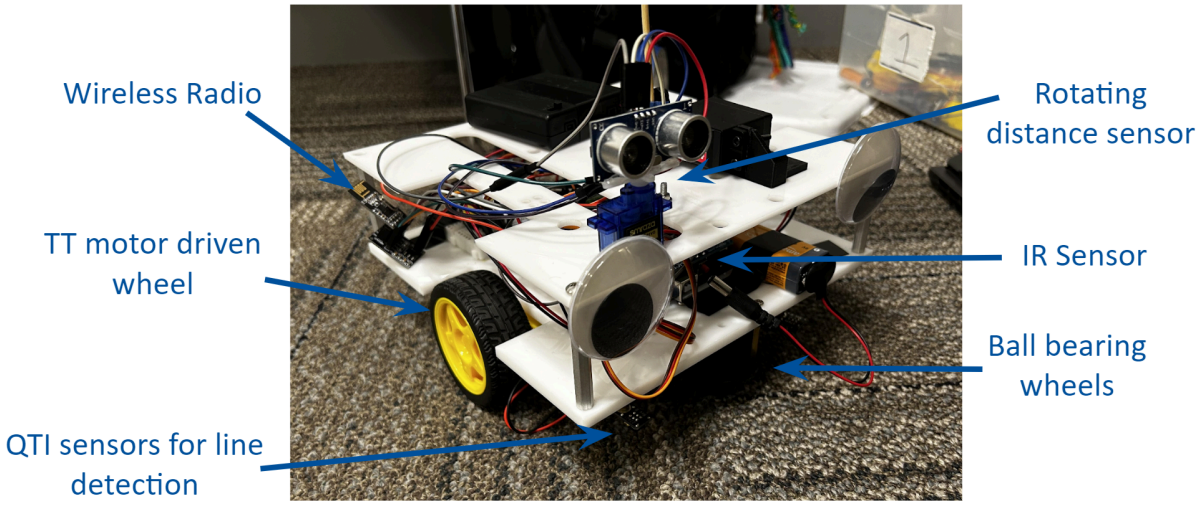

Figure 3: Picture of prototype at showcase.

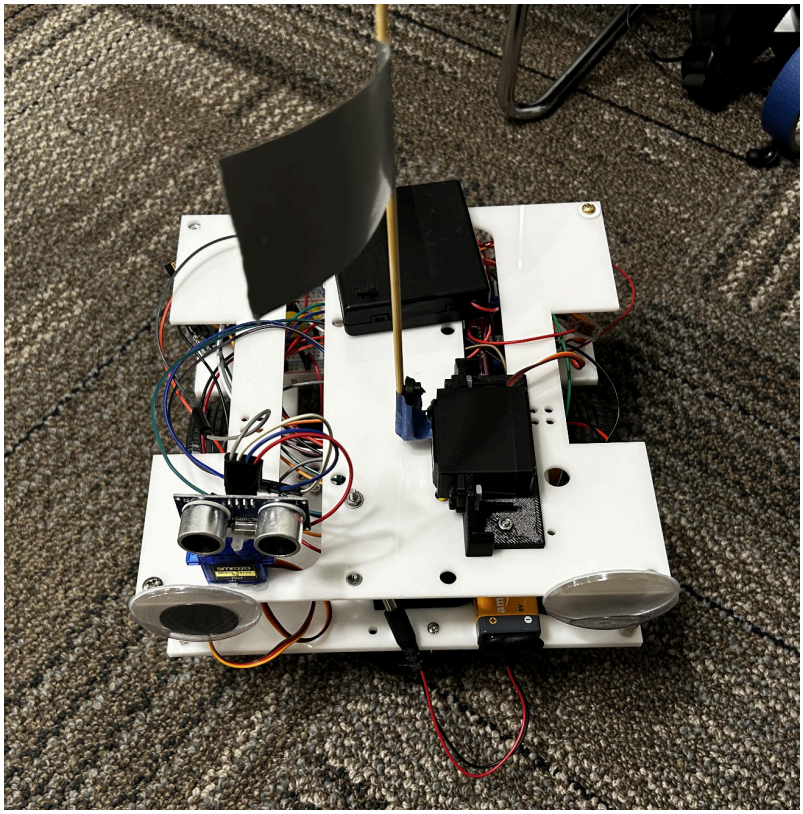

Figure 4: Picture of prototype at showcase.

2. Programming:

The utilization of this vehicle relies on the press of a button. Once the button is pressed, the radios begin transmitting, sensors begin running, and the vehicle becomes able to move. The non-autonomous version uses a joystick to communicate commands to the vehicle. The primary commands are up, down, left, and right. Pressing down on the joystick will cause the ladder to be lowered, and releasing the click will cause the ladder to be returned to its original position. The joystick is connected to a second Arduino Mega with a transmitter code file (Appendix B).

The line detection relies on QTI sensors which compare the reflectivity values of the ground below the vehicle. When “white” values are sensed, the vehicle translates this as driving over a line. The angle detection also uses the QTI sensors, but compares the values between the two sensors to determine the angle at which the robot has rotated from a straight position. Sections of the line detection function can be seen below.

```
void detectLines(){
  //line detection
  digitalWrite(10, LOW);  //blue       set LED pins to LOW
  digitalWrite(11, LOW);  //red
  digitalWrite(12, LOW);  //green
  sense1=RCTime(8); //Check QTI sensor values
  sense2=RCTime(9);
  if (sense1>minForWhite && t<checkTime){ //if sensor 1 detects black, and a line has not recently been crossed, update the relative time t1
    t1=t_true;
  }
  if (sense2>minForWhite && t<checkTime){ //if sensor 2 detects black, and a line has not recently been crossed, update the relative time t2
    t2=t_true;
  }
  if (sense1<minForWhite){ //if white is detected
    if (t>checkTime){ //if a line has not recently been crossed
      whiteEnded=false; //the line is identified
      t_new=t+t_new; //the time since crossing the line is updated
      lineCount=1; //a line is counted
    } else {
      if (whiteEnded){//if the end of the line has been already reached, this is a second line
        lineCount=2; //count two lines
        t_new=t+t_new; //time since crossing the line is updated
      } else {
        t_new=t+t_new; //time since crossing the line is updated
      }
    }
  } else { //if black is detected
    t_white_start=t_true; //time of starting to cross the line is updated
    whiteEnded=true; //a line has ended
    if (t>checkTime){ //if done searching for type of line
      if (lineCount==3){ //if out of bounds
        digitalWrite(12, HIGH); //light green LED
      }
      if (lineCount==2){ //if double line
        digitalWrite(10, HIGH); //light blue LED
      }
      if (lineCount==1) { //if single line
        digitalWrite(11,HIGH); //light red LED
      }
      lineCount=0;//set lines to zero
    }
  }
  if (t_white>maxTimeForWhite){//if the line has not ended after a maximum time, turn on the green out of bounds LED and set the lines to 3
      digitalWrite(12, HIGH);
      lineCount=3;
  }
}
```

Figure 5. Code snippet showing how the QTI sensor output was used to distinguish between single, double, and out-of-bounds lines.

In addition, our group worked to create autonomous code to allow the vehicle to explore the grid and operate without the use of a controller. This code was designed to have the robot find a path through the entire grid, avoiding hitting boxes using the distance sensors, and instead checking each box on each accessible side for the IR signal, and activating the ladder if it found it. In this way, the vehicle could activate all of the brake beams on the grid, regardless of the box positions, configurations, or orientations.

3. Innovations

This design has two key innovations. One innovation can be found on the physical prototype and is the rotating distance sensor. This distance sensor is used to determine the locations of boxes within the grid. The micro servo on which the sensor is attached rotates so the robot can change the surrounding squares in the grid for boxes. This system is specifically implemented such that the servo rotates on 90 degree intervals to check if there are boxes in the squares directly in the spaces to the right, left, and front of the current square the robot is located in the grid. If the distance sensor reads a value below a certain threshold, then it determines there

is a box and it should check to see if an IR signal is being emitted. If the value is not below this threshold for any of these three squares, the robot determines there are no boxes surrounding the current grid space. This subsystem is useful in automation as the vehicle is able to more quickly locate potential fires.

The second innovation is the simulation used to test automation code. We were able to make functional simulated code that could take any input grid and simulate how the vehicle would traverse and explore that space, activating all of the break beam sensors in the simulated grid. This code accounted for the orientation and movement capabilities of the vehicle, as well as the movement capabilities and dependent orientations of the IR and distance sensors. The code progresses in steps, outputting the current grid into the serial to be viewed, with a one-second gap in between "frames" of the simulation. Each grid cell has a number with a corresponding meaning so that both the vehicle and the user looking at the simulation could know what the vehicle should be doing at any given time based on the grid state. The simulation code is entirely functional, and can explore grids given to it; testing every box fully.

DISCUSSION:

The physical prototype is merited by its stability and ability to turn smoothly from the ball bearings. The wires are confined within the two layers but are still accessible by removing the top layer. While there are many wires and a number of components, the wiring is organized using an Arduino Mega shield. The robot is able to consistently detect single and double lines on the ground as well as the IR sensor within buildings. Related to these sensors, the code is easy to change in order to adjust sensor values for environment changes (i.e. changes in lighting or dirtying of mat).

Despite being able to complete tasks to meet milestones, the vehicle struggles in a number of areas. Firstly, the DC motors are unreliable and inconsistent. Because the two motors do not rotate at the same speed, there were issues with driving perfectly straight as well as with speed control. The vehicle slips sometimes and has trouble driving smoothly when the flatness of the mat changes. These issues have been mostly mitigated but are still weak points in the design. The continuous servo used for the ladder also causes difficulty as this motor moves based on speed values that are not consistent between motors or in both directions. Getting the servo to rotate the same amount forward and then backward for the ladder to touch the top of a building is an ongoing challenge as it is always off in one direction by a small amount that adds up over time.

Although we were able to get the manual control to work well, we had some trouble with the autonomous side of things. As described above, a simulation was used to test automation code. While the code mirrors reality and was designed to be able to be inserted into our main manual code, the interfacing between the simulation and physical device failed with the device not being able to perform nearly any of the actions that it was supposed to do. While the combined code would compile with no syntax errors, there were clearly multiple interfacing errors within the 1500+ lines of code that we had after combining, which made it next to

impossible to troubleshoot in the time that we had to do so, especially since the device was failing to give physical or serial feedback as to what the root issues might be.

CONCLUSION:

The project has a number of constraints, one of which is a maximum size of 20cm x 20cm (width x length). If this constraint was not in place or was modified, it would be possible to make the vehicle the size of the grid. If the vehicle was the size of the grid, it would be easier to track position, keep straight motion, and detect changes in direction. By fitting between the gridlines, the vehicle would be more secure and precise in its grid location. A larger vehicle in the length and width directions would also allow for more components to be added on this plane leading to a possible decrease in the overall height of the vehicle. During the competition, the robot was able to successfully complete the task manually via a joystick. With some more time and testing, we are confident that autonomous control can be achieved with our current setup.

APPENDICES

APPENDIX A:

This appendix provides details about the Bill of Materials (Table A.1) and budget. The budget accounts for all purchased parts and must not exceed $300. The BOM is the full price of producing this device and includes items that are both purchased and borrowed. Out of the given budget, $91.53 was spent. The total cost of designing this vehicle was $244.46.

Table A.1: Bill of Materials. Items in bottom gray rows are included in the budget.

Part	Link	No. Req	Units Req	Cost/Unit	Cost
AA battery holder	battery holder	1	1	$8.39	$8.39
NRF24 Wireless Radio	radio	2	1	$13.99	$13.99
NRF24 Breakout Adapter	radio adapter	2	1	$5.99	$5.99
QTI Sensor	QTI	2	2	$9.99	$19.98
Continuous Rotation Servo	servo (cont)	1	1	$10.69	$10.69
Elegoo Mega	Elegoo	2	2	$29.99	$59.98
Breadboards	breadboards	2	1	$9.99	$9.99
Micro Servo	micro servo	1	1	$6.99	$6.99
HC-SR04 Ultrasonic sensor	distance sensor	1	1	$6.99	$6.99
Wooden Dowel	dowel rod	1	1	$5.99	$5.99
IR Sensor	IR sensor	1	1	$3.95	$3.95
Arduino Shield	shield	1	1	$32.00	$32.00
Driver Controller	driver	1	1	$11.49	$11.49
TT Motors	motors	2	1	$14.99	$14.99
Ball Bearing Wheels	bearings	2	1	$8.99	$8.99
Chassis (wood)	FabLab	1	60	$0.05	$3.00
Chassis (acrylic)	FabLab	4	36	$0.10	$14.40
Joystick	joystick	1	1	$6.66	$6.66
Total					$244.46

APPENDIX B:

This appendix includes the final version of the code used to run the robot.

1. **Manual Transmitter:**

https://drive.google.com/file/d/1LlLSt4B8oPYtKmbaHfgwn2VQ00mjjQYj/view?usp=sharing

```
// including libraries
#include <SPI.h>
#include "printf.h"
#include "RF24.h"

#define CE_PIN 7  // Defining the radio pins
#define CSN_PIN 8

// Instantiating an object for the nRF24L01 transceiver
RF24 radio(CE_PIN, CSN_PIN);

// Let these addresses be used for the pair
uint8_t address[][6] = { "1Node", "2Node" };
bool radioNumber = 0;

int joy1_x = A0;
int joy1_y = A1;
int joy_b = 10;

// defining joystick values
int y_pos = 512;
int x_pos = 512;

// Declaring an array for motor control (4 motors + direction)
uint16_t motorcontrol[3];

void setup()
{
  // setup serial monitor
  Serial.begin(115200);
  radio.begin();
  if (!radio.begin()) {
    Serial.println("Radio hardware is not responding!!");
    while (1) {} // Infinite loop to halt further execution
```

```
}
  // setup radio transmitter
  radio.setPALevel(RF24_PA_LOW);
  radio.setPayloadSize(sizeof(motorcontrol));
  radio.openWritingPipe(address[radioNumber]);
  pinMode(joy1_x,INPUT);
  pinMode(joy1_y,INPUT);
  pinMode(joy_b,INPUT);
  digitalWrite(joy_b,HIGH);
}

void loop() {
  Serial.println("motorcontrol values: ");

  // Read the joystick positions
  x_pos = analogRead(joy1_x);
  y_pos = analogRead(joy1_y);

  x_pos = map(x_pos, 0, 1023, 0, 510); //Mapping the x values to (0 - 510)
  y_pos = map(y_pos, 0, 1023, 0, 510); //Mapping the y values to (0 - 510)

  if (x_pos > 240 & x_pos < 270){
    motorcontrol[0] = 255;
  }
  else{
    motorcontrol[0] = x_pos;
  }
  if (y_pos > 240 & y_pos < 270) // backward
  {
    motorcontrol[1] = 255;
  }
  else{
    motorcontrol[1] = y_pos;
  }

  //Display the x_pos and y_pos values in the serial monitor.
  Serial.print("x_pos: ");
  Serial.print(motorcontrol[0]);
  Serial.print("y_pos: ");
  Serial.print(motorcontrol[1]);
```

```
  // Read button state for ladder
  int button = digitalRead(joy_b);
  motorcontrol[2] = button;

  // Displaying the button state in serial monitor
  Serial.print("button state:");
  Serial.print(button);

  // Sending the Motor Control data to radio
  bool report = radio.write(&motorcontrol, sizeof(motorcontrol));  //
transmit report
  if (report)
  {
    Serial.println("transmission successful");
  }
  delay(10);  // delay for 10ms

}
```

2. **Manual Receiver and MAIN code:**

https://drive.google.com/file/d/1hAWjwh-rY-iYbdeWVa1sDZf5nOCo83Jk/view?usp=sharing

```
////////////////////////////////////////////VARIABLES/////////////////////
//////////////////////////

int checkTime= 690;
int minForWhite=150;
int maxTimeForWhite=200;
float sensorDistance=4;   //in
float avgVel=2;   //in/sec


float basePower=50;

const int buttonPin = 6; // Pin connected to the button
int buttonState = LOW;    // Variable to store the state of the button
```

```
int lastButtonState = LOW; // Variable to store the previous state of the
button
int goState = LOW;       // Variable to store the state of the robot


float angle=0; //deg
int lineCount=0;
int sense1=0;
int sense2=0;
bool whiteEnded=false;
float t1;
float t2;
unsigned long t_true;
unsigned long t;
unsigned long t_new;
unsigned long t_white;
unsigned long t_white_start;
unsigned long t_servo;

#include <Servo.h>
Servo servo_ladder;
int down_speed = 78;
int up_speed = 98;

// include libraries
#include <SPI.h>
#include "printf.h"
#include "RF24.h"
#include "IRremote.h"

#define CE_PIN 25
#define CSN_PIN 24

#define SERVO_PIN 34

RF24 radio(CE_PIN, CSN_PIN);
uint8_t address[][6] = { "1Node", "2Node" };
uint16_t motorcontrol[3];
bool radioNumber = 0;
```

```
//PWM Pins to control motor Speeds
int motor1 = 4; // Left Wheel
int motor2 = 3; // Right Wheel

// motor Speed Control
float m1_a=1; // left modifier
float m2_a=1; // right modifier

const int motorPin1_1 = 26; // Left Wheel
const int motorPin1_2 = 27; // Interrupt Pins

const int motorPin2_1 = 28; //  Right Wheel
const int motorPin2_2 = 29; //  Interrupt Pins

////////////////////////////////////////////////MAIN/////////////////////
//////////////////////////


void setup() {
  pinMode(10, OUTPUT); // RED LED pin
  pinMode(11, OUTPUT); // BLUE LED pin
  pinMode(12, OUTPUT); // YELLOW LED pin
  pinMode(13, OUTPUT); // GREEN LED pin
  pinMode(35, INPUT);
  pinMode(motor1,OUTPUT);
  pinMode(motor2,OUTPUT);
  pinMode(buttonPin, INPUT);

  servo_ladder.attach(SERVO_PIN, 500, 2500);
  // servo_ladder.write(80);

  servo_ladder.write(88); // set ladder speed to '0'

  // setup serial monitor
  Serial.begin(115200);
  radio.begin();
  if (!radio.begin()) {
    Serial.println(F("radio hardware is not responding!!"));
    while (1) {}  // hold in infinite loop
  }
```

```
  // set motor control pins to outputs

  pinMode(motor1,OUTPUT);  //PWM - Speed control pins
  pinMode(motor2,OUTPUT);

  pinMode(motorPin1_1, OUTPUT); // Motor 1 Interrupt Pins
  pinMode(motorPin1_2, OUTPUT);

  pinMode(motorPin2_1, OUTPUT); // Motor 2 Interrupt Pins
  pinMode(motorPin2_2, OUTPUT);

  //set up receiver
  radio.setPALevel(RF24_PA_LOW);
  radio.openReadingPipe(1, address[radioNumber]);
  radio.setPayloadSize(sizeof(motorcontrol));  // float datatype occupies
4 bytes
  radio.startListening();

}

void loop() { //MAIN

  t = millis() - t_new;
  t_white=millis()-t_white_start;
  t_true=millis();
  Serial.print(sense1);
  Serial.print(", ");
  Serial.println(t_white);

  if (executeLoop()){ //run code loop

    hbridgeDrive();

    detectLines();

    calcAngle();

    checkIR();

  } else {
```

```
    stopMotors();
    t_white_start=t_true;

  }
  delay(2);
}

////////////////////////////////////////////FUNCTIONS/////////////////////
//////////////////////////

bool executeLoop(){
  buttonState = digitalRead(buttonPin);

  // Check if the button state has changed
  if (buttonState != lastButtonState) {
    // If the button is pressed (changed from HIGH to LOW)
    if (buttonState == LOW) {
      // Toggle the LED state
      goState = !goState;
    }
    // Save the current button state
    lastButtonState = buttonState;
  }

  return (goState != LOW);
}

void hbridgeDrive(){

  uint8_t pipe;
  if (radio.available(&pipe)) // is there a payload? get the pipe number
that recieved it
  {
    uint16_t bytes = radio.getPayloadSize();  // get the size of the
payload
    radio.read(&motorcontrol, bytes);             // fetch payload from
FIFO

delay(10);
```

```
    // drive motors    - Set motor speeds
      analogWrite(motor1,248); //front left
      analogWrite(motor2,255);

    //direction control

    // Forward Direction when (Y_pos > 255)
    if (motorcontrol[1]>255){

      digitalWrite(motorPin1_1, HIGH);
      digitalWrite(motorPin1_2, LOW);

      digitalWrite(motorPin2_1, HIGH);
      digitalWrite(motorPin2_2, LOW);
    }

    //Backward Direction when (Y_pos < 255)
    else if(motorcontrol[1]<255){

      digitalWrite(motorPin1_1, LOW);
      digitalWrite(motorPin1_2, HIGH);

      digitalWrite(motorPin2_1, LOW);
      digitalWrite(motorPin2_2, HIGH);
    }

    // Rightside wheel Backwards, Left side wheel Forward - to make the
robot face right
    else if(motorcontrol[0]>255){    //(X_pos > 255) Right Turn

      digitalWrite(motorPin1_1, LOW);
      digitalWrite(motorPin1_2, HIGH);

      digitalWrite(motorPin2_1, HIGH);
      digitalWrite(motorPin2_2, LOW);
    }

    // Rightside wheel Forward, Left side wheel Backwards - to make the
robot face left
    else if(motorcontrol[0]<255){     //(X_pos < 255) Left Turn
```

```
      digitalWrite(motorPin1_1, HIGH);
      digitalWrite(motorPin1_2, LOW);

      digitalWrite(motorPin2_1, LOW);
      digitalWrite(motorPin2_2, HIGH);
    }

    //Stay Stable - No movement
    else{
      digitalWrite(motorPin1_1, LOW);
      digitalWrite(motorPin1_2, LOW);

      digitalWrite(motorPin2_1, LOW);
      digitalWrite(motorPin2_2, LOW);
    }
        // ladder
    if(motorcontrol[2]==0){
        t_servo=t_true;
    } else {
        if (t_true-t_servo<500){
        servo_ladder.write(down_speed);
        }
        else if (t_true-t_servo<1500){
        servo_ladder.write(88);
        }
        else if (t_true-t_servo<2000){
        servo_ladder.write(up_speed);
        }
        else if (t_true-t_servo<3000){
        servo_ladder.write(88);
        }
    }

  }

}

long RCTime(int sensorIn){
   long duration = 0;
```

```
   pinMode(sensorIn, OUTPUT);      // Make pin OUTPUT
   digitalWrite(sensorIn, HIGH);   // Pin HIGH (discharge capacitor)
   delay(1);                       // Wait 1ms
   pinMode(sensorIn, INPUT);       // Make pin INPUT
   digitalWrite(sensorIn, LOW);    // Turn off internal pullups
   while(digitalRead(sensorIn)){   // Wait for pin to go LOW
      duration++;
   }
   return duration;
}

void checkIR(){
  if(digitalRead(35)==LOW){
    digitalWrite(13, HIGH);
  } else {
    digitalWrite(13, LOW);
  }
}

void detectLines(){
  //line detection
  digitalWrite(10, LOW); // Red LED set to LOW
  digitalWrite(11, LOW); // Blue LED set to Low
  digitalWrite(12, LOW); //Yellow LED set to Low
  sense1=RCTime(8);
  sense2=RCTime(9);
  if (sense1>minForWhite && t<checkTime){
    t1=t_true;
  }
  if (sense2>minForWhite && t<checkTime){
    t2=t_true;
  }
  if (sense1<minForWhite){
    if (t>checkTime){
      whiteEnded=false;
      t_new=t+t_new;
      lineCount=1;
    } else {
      if (whiteEnded){
        lineCount=2;
```

```
        t_new=t+t_new;
      } else {
        t_new=t+t_new;
      }
    }
  } else {
    t_white_start=t_true;
    whiteEnded=true;
    if (t>checkTime){
      if (lineCount==3){
        digitalWrite(12, HIGH);
      }
      if (lineCount==2){
        digitalWrite(10, HIGH);
      }
      if (lineCount==1) {
        digitalWrite(11,HIGH);
      }
      lineCount=0;
    }
  }
  if (t_white>maxTimeForWhite){
      digitalWrite(12, HIGH);
      lineCount=3;
  }

}

void calcAngle(){
  if (sense1<minForWhite && sense2<minForWhite){
    angle =( atan2 (((abs(t1-t2))/1000)*avgVel, sensorDistance) *
180/3.14159265 );
  }
}

void stopMotors(){
  analogWrite(motor1,0); //front right
  analogWrite(motor2,0); //back right
}
```

3. Autonomous Vehicle Simulation:

https://drive.google.com/file/d/1MhXHo1AKM5HmjfGUUcFGQmTISOaufnCt/view?usp=sharing

```
//In serial grid:
//0 - unchecked
//1 - clear
//2 - box (unchecked)
//3 - box (no light)
//4 - box (light off)
//5 - box (being checked)
//6 - path check
//8 - box placeholder
//9 - car

char carOrientation='N'; //intial orientation of the car is NSEW cardinal directions, with N being away from the starting side
int xStart=7; //Hard coded start position of car; bottom left corner (0,0)
int yStart=0;



int xPos=xStart; //intializes car position
int yPos=yStart;
int layout[8][8]={ //initializes simulated grid
  {1, 1, 1, 1, 1, 1, 1, 9,},
  {0, 0, 0, 0, 0, 0, 0, 0,},
  {0, 0, 0, 0, 0, 0, 2, 0,},
  {0, 0, 2, 0, 0, 0, 0, 0,},
  {0, 0, 0, 0, 0, 0, 0, 0,},
  {0, 2, 0, 0, 0, 0, 0, 0,},
  {0, 0, 0, 2, 0, 2, 0, 0,},
  {0, 0, 0, 0, 0, 0, 0, 0,}
};
int history[][2]={{xStart,yStart}}; //initializes the "memory" of the path the car has taken
bool onStart=true; //intializes that car begins by traveling to (0,0)
bool checkBox=false; //initializes that car is not actively searching a box
int step=0; //initializes that no actions have occured yet
bool record=true; //initializes that car will record its path
int checksides[]={0,0,0,0}; //initializes that the satate of no sides of a box are known
int boxx; //initializes searched box x and y pos as global variables
int boxy;

void setup() {
  Serial.begin(9600); //simulation is printed to serial
}
```

```
void loop() {
  record=true;//in each loop, the car should default to recording its actions
  if(onStart){ //if in the starting phase, traveling to the (0,0) square, move left until the (0,0) square is reached, then end the starting phase
    if(layout[0][0]==9){
      onStart=false;
    } else {
      moveleft();
    }
  } else { //if not in the starting phase:
    if (!checkBox){ //if not checking a box
      if (lookforbox()){ //look for a box in the adjacent grid cells using the distance sensors, and if found, begin searching that box, and determine which sides of the box are accessible to be searched
        checkBox=true;
        boxaccess();
      } else { //if no box is found:
        if (leftclear()){ //if the car can move left (West), and has not already checked that tile, do so
          moveleft();
        } else if(!leftclear()&&upclear()){ //if the car can move up (North), and has not already checked that tile, do so
          moveup();
        } else if(!leftclear()&&!upclear()&&downclear()){ //if the car can move down (South), and has not already checked that tile, do so
          movedown();
        } else if (!leftclear()&&!upclear()&&!downclear()&&rightclear()){ //if the car can move right (East), and has not already checked that tile, do so
          moveright();
        } else if (!leftclear()&&!upclear()&&!downclear()&&!rightclear()){ //if the car has already cheked every adjacent tile:
          record=false; //do not record this movement
          step--; //reduce the record entry number by one (like going back in time/undoing)
          int deltax=xPos-history[step][0];//see if the car moved horizontally to get to the current grid tile (-1 -> entered from East, 1 -> entered from West)
          int deltay=yPos-history[step][1];//see if the car moved vertically to get to the current grid tile (-1 -> entered from South, 1 -> entered from North)
          if (deltax==1){//if entered from the west, move west
            moveleft();
          }
          if (deltax==-1){//if entered from the east, move east
            moveright();
          }
          if (deltay==1){//if entered from the south, move south
            movedown();
          }
          if (deltay==-1){//if entered from the north, move north
            moveup();
          }
        }
      }
    } else { //if checking a box:
      int dstFromBoxx=xPos-boxx; //determine distance from the box
      int dstFromBoxy=yPos-boxy;
      if (dstFromBoxx==1&&dstFromBoxy==0&&checksides[2]==0){//if the car is on the east side of the box, scan for IR while facing west (towards the box), and record if it is lit or not
        ifrW();                                                                                                                    //scan ifrs
      }
      if (dstFromBoxx==-1&&dstFromBoxy==0&&checksides[3]==0){//if the car is on the west side of the box, scan for IR while facing east (towards the box), and record if it is lit or not
        ifrE();
      }
      if (dstFromBoxx==0&&dstFromBoxy==1&&checksides[0]==0){//if the car is on the north side of the box, scan for IR while facing south (towards the box), and record if it is lit or not
        ifrS();                                                                                                                    //scan ifrs
      }
      if (dstFromBoxx==0&&dstFromBoxy==-1&&checksides[1]==0){//if the car is on the south side of the box, scan for IR while facing north (towards the box), and record if it is lit or not
        ifrN();
      }
      if (leftsix()){//if there is a path check grid to west, move west
          moveleft();
        } else if(!leftsix()&&upsix()){//if there is a path check grid to the north, move north
          moveup();
        } else if(!leftsix()&&!upsix()&&downsix()){{//if there is a path check grid to the south, move south
          movedown();
        } else if (!leftsix()&&!upsix()&&!downsix()&&rightsix()){{//if there is a path check grid to the west, move west
          moveright();
        } else if (!leftsix()&&!upsix()&&!downsix()&&!rightsix()){{//if there are no path check grids on any side, begin backtracking with the same process as seen at line 65
          record=false;
          step--;
          int deltax=xPos-history[step][0];
          int deltay=yPos-history[step][1];
          if (deltax==1){
            moveleft();
          }
          if (deltax==-1){
            moveright();
          }
          if (deltay==1){
            movedown();
          }
          if (deltay==-1){
            moveup();
          }
        }
      boxchecked();//check if the box has been fully checked, or if it should continue to be searched
    }
  }
```

```
  for(int i=7;i>=0;i--){//print out the current grid state, with (0,0) in the bottom left
    for (int j=0;j<7;j++){
      Serial.print(layout[i][j]);
      Serial.print(" ");
    }
    Serial.println(layout[i][7]);
  }
  Serial.print("  ");
  Serial.print(checkBox);//Print whether the car is actively searching a box for IR signals (0 = not searching, 1 = searching)
  Serial.print("  ");
  Serial.print(checksides[0]);//Print the checked-state of each side of the box corresponding to NSEW (0 = unchecked, 1 = checked - no IR, 5 = checked - IR detected)
  Serial.print(checksides[1]);
  Serial.print(checksides[2]);
  Serial.print(checksides[3]);
  Serial.print("  ");
  Serial.println(carOrientation);//Print the facing of the car
  if (record){//if the car is recording this step, increment the step and log the position of the car
    step++;
    history[step][0]=xPos;
    history[step][1]=yPos;
  }
  delay(1000);//delay 1s for loop to allow for readability

}


void ifrN(){//Checks for IR to the north, if there is a signal, set the side's value to 5, otherwise set it to 1
  if (random(6)<5){ //SIMULATED CODE RANDOMIZES IF A SIDE IS EMITTING IR, REPLACE WITH IR SENSOR CONDITIONAL
    checksides[1]=1;
  } else {
    checksides[1]=5;
  }
}
void ifrS(){//Checks for IR to the south, if there is a signal, set the side's value to 5, otherwise set it to 1
  if (random(6)<5){ //SIMULATED CODE RANDOMIZES IF A SIDE IS EMITTING IR, REPLACE WITH IR SENSOR CONDITIONAL
    checksides[0]=1;
  } else {
    checksides[0]=5;
  }
}
void ifrE(){//Checks for IR to the east, if there is a signal, set the side's value to 5, otherwise set it to 1
  if (random(6)<5){ //SIMULATED CODE RANDOMIZES IF A SIDE IS EMITTING IR, REPLACE WITH IR SENSOR CONDITIONAL
    checksides[3]=1;
  } else {
    checksides[3]=5;
  }
}
void ifrW(){//Checks for IR to the west, if there is a signal, set the side's value to 5, otherwise set it to 1
  if (random(6)<5){ //SIMULATED CODE RANDOMIZES IF A SIDE IS EMITTING IR, REPLACE WITH IR SENSOR CONDITIONAL
    checksides[2]=1;
  } else {
    checksides[2]=5;
  }
}

void boxchecked(){//checks if a box has been fully checked, either by determining that no sides are emitting, or by finding an emitting side
  int sum=0;
  for (int i=0;i<4;i++){
    sum=sum+checksides[i];//finds the total of the checksides array
  }
  if (sum>4){ //if an IR emitter has been found, stop searching the box, and lower the ladder
    checkBox=false;
    layout[boxy][boxx]=4;//mark the box as searched with IR turned off
    //INSERT CODE TO LOWER LADDER
  }
  if (sum==4){ //if all sides are not emitting, stop searching the box, and set all path tiles as unchecked
    checkBox=false;
    for (int i=0;i<8;i++){
      for (int j=0;j<8;j++){
        if (layout[i][j]==6){
          layout[i][j]=0;
        }
      }
    }
    layout[boxy][boxx]=3;//mark box on grid as searched with no IR
  }
}
```

```
void boxaccess(){//intializes inaccessible faces as checked with no signal
  for (int i=0;i<4;i++){
    checksides[i]=0;//NSEW faces: 0-checked 1-no signal 5-signal
  }
  if (boxx==0){//if the box is on the left edge of the grid, the west side is inaccessible
    checksides[3]=1;
  } else if(layout[boxy][boxx-1]>1){//if there is a box or vehicle occupying the space to the left of the box, the west side is inaccessible
    checksides[3]=1;
  }
  if (boxx==7){//if the box is on the right edge of the grid, the east side is inaccessible
    checksides[2]=1;
  } else if(layout[boxy][boxx+1]>1){//if there is a box or vehicle occupying the space to the right of the box, the east side is inaccessible
    checksides[2]=1;
  }
  if (boxy==0){//if the box is on the bottom edge of the grid, the south side is inaccessible
    checksides[1]=1;
  } else if(layout[boxy-1][boxx]>1){//if there is a box or vehicle occupying the space below the box, the south side is inaccessible
    checksides[1]=1;
  }
  if (boxy==7){//if the box is on the top edge of the grid, the north side is inaccessible
    checksides[0]=1;
  } else if(layout[boxy+1][boxx]>1){//if there is a box or vehicle occupying the space above box, the north side is inaccessible
    checksides[0]=1;
  }
  for (int i=0;i<3;i++){//loop through the grid tiles within one space of the box (corners inclusive)
    if (boxx+i-1>=0&&boxx+i-1<=7){
      for (int j=0;j<3;j++){
        if (boxy+j-1>=0&&boxy+j-1<=7){
          if (layout[boxy+j-1][boxx+i-1]<2){//if the tile is clear or unexplored, set it to a path tile
            layout[boxy+j-1][boxx+i-1]=6;
          } else if (layout[boxy+j-1][boxx+i-1]==2||layout[boxy+j-1][boxx+i-1]==3||layout[boxy+j-1][boxx+i-1]==4){//if the tile is another box of any type, set path tiles around that box
            goaround(boxx+i-1, boxy+j-1);
          }
        }
      }
    }
  }
  layout[boxy][boxx]=5;//identify the current box as being checked on the serial grid
}
```

```
void goaround(int x, int y){//creates a path around the called grid tile
  for (int i=0;i<3;i++){//loop through the grid tiles within one space of the inputted tile (corners inclusive)
    if (x+i-1>=0&&x+i-1<=7){
      for (int j=0;j<3;j++){
        if (y+j-1>=0&&y+j-1<=7){
          if (layout[y+j-1][x+i-1]<2){//if the tile is clear or unexplored, set it to a path tile
            layout[y+j-1][x+i-1]=6;
          }
        }
      }
    }
  }
}
```

```
bool lookforbox(){//Uses the "radar" distance sensor to check the tiles one to the left, to the right, and in front of the car for boxes, and return true if a box is found, and false otherwise
  if (carOrientation=='N'){//if oriented north, check the NEW directions, and if a box is found, return true
    if (boxN()){
      return true;
    }
    if (boxE()){
      return true;
    }
    if (boxW()){
      return true;
    }
  }
  if (carOrientation=='S'){//if oriented south, check the SEW directions, and if a box is found, return true
    if (boxS()){
      return true;
    }
    if (boxE()){
      return true;
    }
    if (boxW()){
      return true;
    }
  }
  if (carOrientation=='E'){//if oriented east, check the NSW directions, and if a box is found, return true
    if (boxN()){
      return true;
    }
    if (boxS()){
      return true;
    }
    if (boxE()){
      return true;
    }
  }
  if (carOrientation=='W'){//if oriented west, check the NSE directions, and if a box is found, return true
    if (boxN()){
      return true;
    }
    if (boxS()){
      return true;
    }
    if (boxW()){
      return true;
    }
  }
  return false;//return false if no boxes are found
}
```

```
bool boxN(){//if a box is detected to the north, return true and assign its position, otherwise return false
  if (yPos<7){
    if (layout[yPos+1][xPos]==2){//REPLACE WITH DISTANCE SENSOR CONDITIONAL
      boxx=xPos;//set box's position
      boxy=yPos+1;
      return true;
    }
  }
  return false;
}
bool boxS(){//if a box is detected to the south, return true and assign its position, otherwise return false
  if (yPos>0){
    if (layout[yPos-1][xPos]==2){//REPLACE WITH DISTANCE SENSOR CONDITIONAL
      boxx=xPos;//set box's position
      boxy=yPos-1;
      return true;
    }
  }
  return false;
}
bool boxE(){//if a box is detected to the east, return true and assign its position, otherwise return false
  if (xPos<7){
    if (layout[yPos][xPos+1]==2){//REPLACE WITH DISTANCE SENSOR CONDITIONAL
      boxx=xPos+1;//set box's position
      boxy=yPos;
      return true;
    }
  }
  return false;
}
bool boxW(){//if a box is detected to the west, return true and assign its position, otherwise return false
  if (xPos>0){
    if (layout[yPos][xPos-1]==2){//REPLACE WITH DISTANCE SENSOR CONDITIONAL
      boxx=xPos-1;//set box's position
      boxy=yPos;
      return true;
    }
  }
  return false;
}

void turnleft(){//turn the car 90 degrees counter clockwise and assign it a new facing accordingly
  //INSERT TURNING CODE
  if(carOrientation=='N'){
    carOrientation='W';
  } else if(carOrientation=='W'){
    carOrientation='S';
  } else if(carOrientation=='S'){
    carOrientation='E';
  } else {
    carOrientation='N';
  }
}
void turnright(){//turn the car 90 degrees clockwise and assign it a new facing accordingly
  //INSERT TURNING CODE
  if(carOrientation=='N'){
    carOrientation='E';
  } else if(carOrientation=='W'){
    carOrientation='N';
  } else if(carOrientation=='S'){
    carOrientation='W';
  } else {
    carOrientation='S';
  }
}
```

```
bool leftclear(){//if there is no box to the west, and the square has not yet been checked, return true, otherwise, return false
  if (xPos!=0){
    if (layout[yPos][xPos-1]==0){ //CHANGE TO IF NO DISTANCE SENSOR IN WEST DIRECTION AND WESTERN GRID TILE UNCHECKED
      return true;
    }
  }
  return false;
}
bool rightclear(){//if there is no box to the east, and the square has not yet been checked, return true, otherwise, return false
  if (xPos!=7){
    if (layout[yPos][xPos+1]==0){ //CHANGE TO IF NO DISTANCE SENSOR IN EAST DIRECTION AND EASTERN GRID TILE UNCHECKED
      return true;
    }
  }
  return false;
}
bool upclear(){//if there is no box to the north, and the square has not yet been checked, return true, otherwise, return false
  if (yPos!=7){
    if (layout[yPos+1][xPos]==0){ //CHANGE TO IF NO DISTANCE SENSOR IN NORTH DIRECTION AND NORTHERN GRID TILE UNCHECKED
      return true;
    }
  }
  return false;
}
bool downclear(){//if there is no box to the south, and the square has not yet been checked, return true, otherwise, return false
  if (yPos!=0){
    if (layout[yPos-1][xPos]==0){ //CHANGE TO IF NO DISTANCE SENSOR IN EAST DIRECTION AND EASTERN GRID TILE UNCHECKED
      return true;
    }
  }
  return false;
}
```

```
void moveleft(){//turn to face west, then drive forwards one square
  while (carOrientation!='W'){//turn until facing west
    if (carOrientation=='N'){
      turnleft();
    } else {
      turnright();
    }
  }
  layout[yPos][xPos]=1;//update serial grid to show current square as checked
  xPos=xPos-1;//update position to be one to the west
  layout[yPos][xPos]=9;//update serial grid to show current position of car
}
void moveright(){//turn to face east, then drive forwards one square
  while (carOrientation!='E'){//turn until facing east
    if (carOrientation=='S'){
      turnleft();
    } else {
      turnright();
    }
  }
  layout[yPos][xPos]=1;//update serial grid to show current square as checked
  xPos=xPos+1;//update position to be one to the east
  layout[yPos][xPos]=9;//update serial grid to show current position of car
}
void moveup(){//turn to face north, then drive forwards one square
  while (carOrientation!='N'){//turn until facing north
    if (carOrientation=='E'){
      turnleft();
    } else {
      turnright();
    }
  }
  layout[yPos][xPos]=1;//update serial grid to show current square as checked
  yPos=yPos+1;//update position to be one to the north
  layout[yPos][xPos]=9;//update serial grid to show current position of car
}
void movedown(){//turn to face south, then drive forwards one square
  while (carOrientation!='S'){//turn until facing south
    if (carOrientation=='W'){
      turnleft();
    } else {
      turnright();
    }
  }
  layout[yPos][xPos]=1;//update serial grid to show current square as checked
  yPos=yPos-1;//update position to be one to the south
  layout[yPos][xPos]=9;//update serial grid to show current position of car
}
```

```
bool leftsix(){//if there is a path tile to the west and the tile does not contain a box, return true, otherwise return false
  if (xPos!=0){
    if (layout[yPos][xPos-1]==6){ //ADD DISTANCE SENSOR EXPRESSION TO CHECK FOR BOX IN WEST DIRECTION
      return true;
    }
  }
  return false;
}
bool rightsix(){//if there is a path tile to the east and the tile does not contain a box, return true, otherwise return false
  if (xPos!=7){
    if (layout[yPos][xPos+1]==6){ //ADD DISTANCE SENSOR EXPRESSION TO CHECK FOR BOX IN EAST DIRECTION
      return true;
    }
  }
  return false;
}
bool upsix(){//if there is a path tile to the north and the tile does not contain a box, return true, otherwise return false
  if (yPos!=7){
    if (layout[yPos+1][xPos]==6){ //ADD DISTANCE SENSOR EXPRESSION TO CHECK FOR BOX IN NORTH DIRECTION
      return true;
    }
  }
  return false;
}
bool downsix(){//if there is a path tile to the south and the tile does not contain a box, return true, otherwise return false
  if (yPos!=0){
    if (layout[yPos-1][xPos]==6){ //ADD DISTANCE SENSOR EXPRESSION TO CHECK FOR BOX IN SOUTH DIRECTION
      return true;
    }
  }
  return false;
}
```

4. **Combining of MAIN and Automated Code (Not Functional):**

https://drive.google.com/file/d/1JpE3uyZqyUnE8LMT4LUhTSEogyC3MJYB/view?usp=sharing
(Images of code not included due to file length)

REFERENCES :

1. QTI Sensor Documentation - https://www.parallax.com/package/qti-line-sensor-documentation/
2. Wireless Transceiver - https://nrf24.github.io/RF24/
3. Transceiver Demo - https://forum.arduino.cc/t/simple-nrf24l01-2-4ghz-transceiver-demo/405123
4. Continuous Servo Motor - https://www.adafruit.com/product/154