Search and rescue robot - Group 7

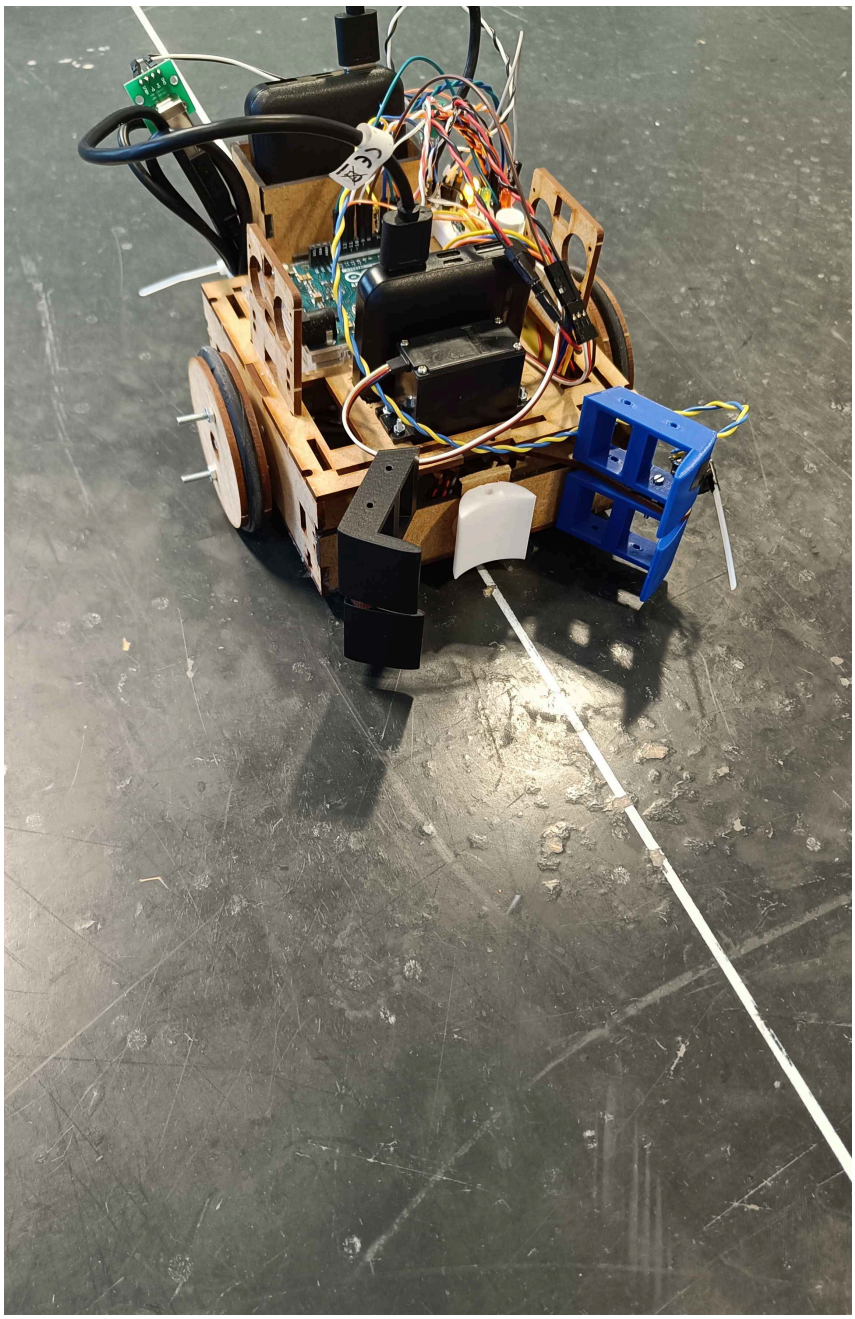
Authors:

Henrik Axelsson
he5207ax-s@student.lu.se

Simon Jacobsson Persson
perssonsimon23@gmail.com

Fredrik Sundt
sundtfredrik@gmail.com

Elias Storaas
eliastor@uio.no



Preface

	Henrik	**Simon**	**Fredrik**	**Elias**
Contributions	Main role in concept developement process and manufacturing.	Wrote, tested and debugged the arduino code for the robot, both regarding the components and the functionality	Wrote, tested and debugged the arduino code for the robot, both regarding the components and the functionality	Worked on design and construction of vehicle as well as assisting in bug fixing

Contents

1. Introduction

Search and rescue operations often take place in environments that are dangerous, unpredictable and difficult for humans to operate in effectively. In these situations robots can play a key role in locating and transporting people in need of help without placing personnel in dangerous positions. The goal of this project is to develop a small autonomous search and rescue robot with the capability of traversing a maze, detecting simulated injured people, in this case wooden blocks, and then removing them from the structure.

The robot platform used in this project was a two wheeled servomotor driven system controlled using an Arduino Leonardo. The task was to expand on these basic components with sensors, other mechanical components and program logic to move around the maze and locate the objects. Once started, the robot should complete its task without any human intervention.

2. Description of the robot system

2.1. Included components

The robot uses an IR sensor array to navigate the maze, and a small bump switch to identify obejcts. The robot runs on a Arduino Leonardo, and is driven by two FIT0458 motor. For gripping objects, the robot uses a Parallax standard 180 degree servo and gripper arm made of laser cut MDF and 3D-printed PLA. The robot is powered by two powerbanks supplying 5V to both the Arduino and the motor control board.

2.2. Physical overview

The robot is mainly based on the standard kit provided in the course. Some modifications have been made in order to mount all components to the chassis. The gripper doesn't lift the objects, but simply pushes them out of the maze. In order to do this securely, a gripper cage was designed and mounted to the gripper arms. In order to identify objects in front of the robot, the bump switch is mounted in front of the gripper cage, and has an antenna arm to be able to feel objects on a larger area in front of the robot. To carry the necessary power source for the robot, a battery carrier was designed to sit on the back of the robot. Most components are mounted on the outside of the robot in order to ease the programming and design phase. Only the motors and motor control board are mounted inside of the robot. The final design is shown in the following figure.

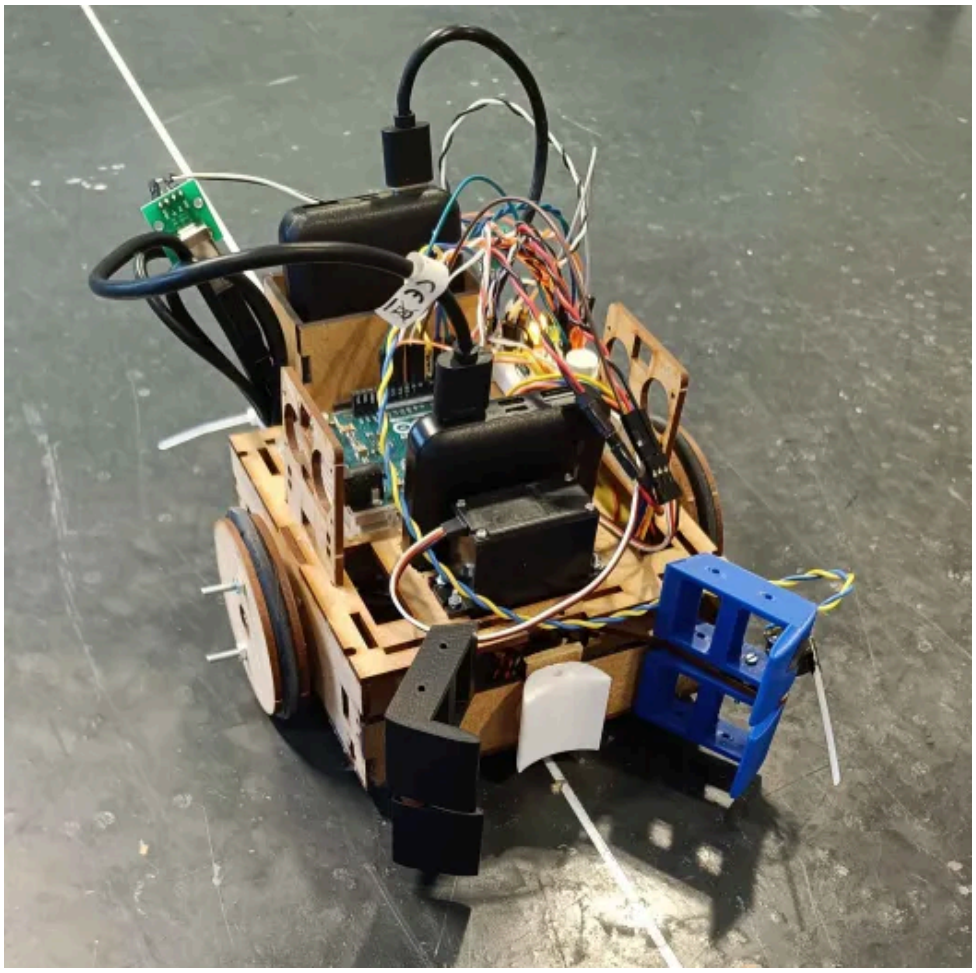


Figure 1: Final design of robot, front view.

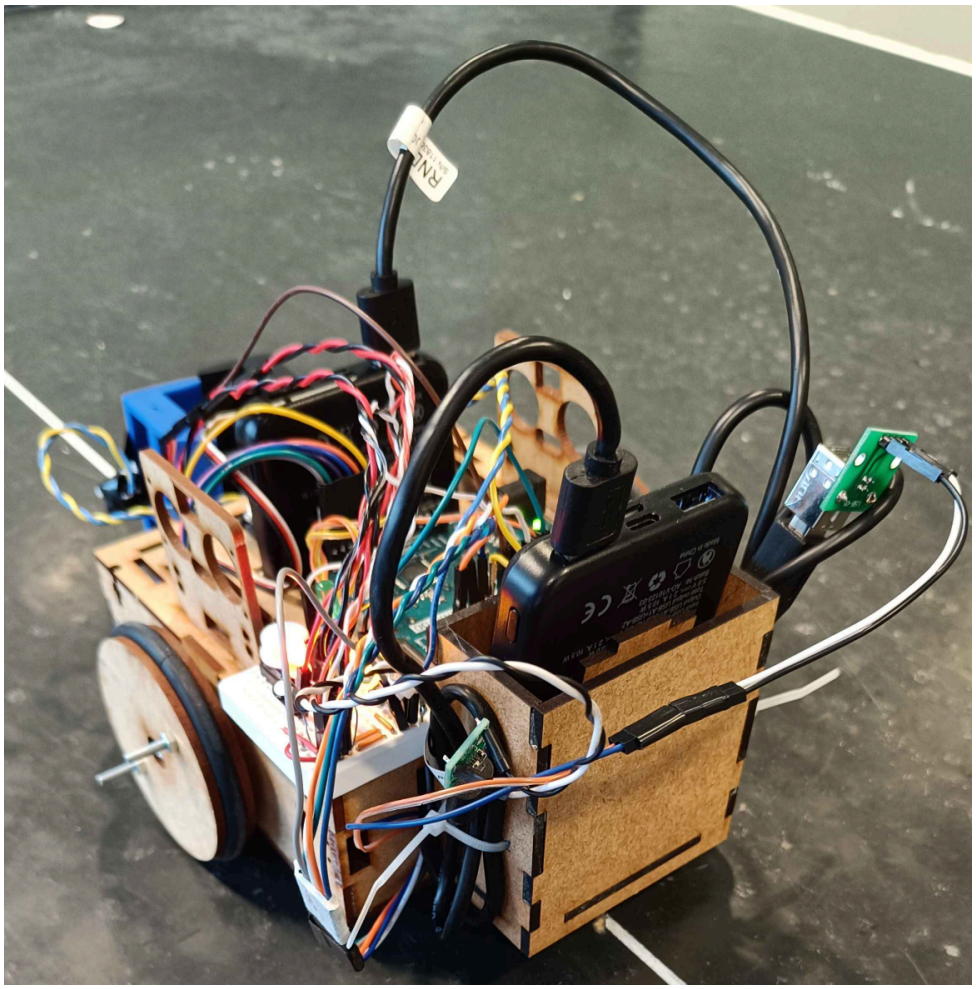


Figure 2: Final design of robot, rear view.

2.3. Arduino code

The code part of the report will explain the following: setup and functions for the components, the PD controller, the traversing methods, and the switch-case system.

2.3.1. Motor

The motor driving the robot uses the *CytronMotorDriver* package in Arduino. In the program a multitude of functions were written to use the motor.

There is one functions that is used to set the speed of both the left and right motor, this functions also constrains the speed to the motors limit and "flips the value of the negative speed". For some reason, the fastest negative speed is −1 and the slowest negative speed is −254, while the fastest positive speed is 255 and the slowest positive speed is 1. We think this could occur because of how the memory is allocated, where we have declared these variables as *int*, either way we had to handle this issue by flipping the speed value for the one that was negative.

The other functions are to handle 90 degrees turns. These work by setting the speed of the motors using the previous function and turning until the IR sensor finds the black line.

2.3.2. IR Sensor

The IR line array sensor has eight sensors, we only use six of them, and is driven using the *QTRSensor* package in Arduino. To make sense of the readings of the sensor we defined two thresholds, between 0 and 1000, one for which values match a black line and one for which values match the ground. This was chosen as above 750 and below 300 respectively.

To actually be able to use the IR sensor we had to calibrate it. This was done by "swaying or rotating" the robot back and forth while continuously reading the sensor. After the calibration is done the robot rotates until the sensor reads that the black line is under the middle of the sensor, that means the robot is straight on the line.

To use the IR sensor during operation we created some help functions. These were functions that returned a boolean depending on if the sensor was over the black line or not for different sensors, the two left-most, the two middle and the two right-most . There was also a functions that returned the number of sensors that was over a black line. This was used to define if the robot was in a intersection.

2.3.3. Servo motor

The servo motor is used for the gripper on the robot and is controlled using the Arduino package *Servo*. In the code we have one functions that opens the gripper and one that closes the gripper, these are made using for-loops that sweep the motor between the open and close position. To actually use the functionality of the gripper in operation one function to pick up an object and one function to drop an object were also defined. These functions both move the robot forward and backward as well as opens and closes the gripper.

2.3.4. LEDs, Button, Bumper switch

Other components that had to be defined in the code were the LEDs, button and bumper switch. The LEDs were simply defined as an output and toggled between *HIGH* and *LOW* directly in the code. There is one LED that light up when the robot is in a intersection and one LED that light up when the robot is returning with an object.

The button was defined as *INPUT_PULLUP* and used to start the calibration of the robot and then to start the operation. This was done in a function by reading the state of the button. When it had been both pressed down and then released we counted it as one press.

The bumper switch was also defines as *INPUT_PULLUP* and was used to determine if the robot had found an object. When the switches reading was *LOW*, i.e the bumper was pressed in, the robot was in contact with an object.

2.3.5. PD controller

To follow the black line with the robot we used a PD controller to determine what speed to set the left and right motor to. This was done by reading the position of the black line under the IR sensor using the built in function *readLineBlack()*, this gave us a value between 0 and 5000 for six sensors. This value was compared to the reading it would give if it was perfectly in the middle which is 2500. This gave us an error which we could use to determine an output. The output was then subtracted and added to left our base speed for the left and right motor respectively.

PD controller code

```
e = 2500 - r
d = e - et-1
u = Kp*e + Kd*d
et-1 = e
leftSpeed = BASE_SPEED - u
rightSpeed = BASE_SPEED + u
```

For the straight parts of the maze that did not have a black line we used a lower fixed base speed.

2.3.6. Traversing methods and handling turns

For traversing the maze we had two different functions or methods that we used, these are the left hand rule and the right hand rule. In our implementation some special turns had to be handled separately, for example getting in and out of the island of the maze which these hand rules are unable to handle.

In our code an intersection is defined as three of the sensors on the line array sense the black line, and that either the two left-most or right-most sensors also sense the black line.

The two hand rules work very similarly and only the last step in different, the steps they do to handle an intersection are the following.

- When a intersection is found, drive straight until both the two left-most and right-most sensors are off the black line. At this drive through we also check if there is a line to both the left and right. That means that we can turn left or right respectively.
- At this position, read the sensors to check if there is a black line in forward of the robot. That means that the robot can continue forward.
- Left-hand rule: Check if it is possible to turn left, if so, turn left. If that is not possible, check if it is possible to keep driving forward. Is that not the case, check if it is possible to turn right.
- Right-hand rule: Turn right if it is possible. If not, keep driving forward if possible. Is that not the case, turn left if that is possible.

2.3.7. Switch cases and turn counting

Our full code solution for the project use two noteworthy parts: switch cases and turn counting. The switch cases are used to easily be able to switch between different modes for the robot. Counting the turns is used to keep track of where the robot is in the maze. Inside each case we also changed the robots behavior if it was searching for an object or returning with one.

The counting, which we defined in an variable *NODE* was needed to handle special cases in the maze and to know where the robot was located. This was needed to handle special turns which departed from the hand-rule in and out of the island. It was also necessary to handle the parts of the maze that lacked the black line. Between some turns we had to turn or drive straight on parts where the line was removed while in the normal case, the lack of a black line meant we were in a dead end and had to turn around. This solution was a must since we did not have any other way of knowing what the lack of the black line ment, like we would if we had used sonar sensors. The *NODE* was also used to know how many turns we had to do to return to the starting zone once we had picked up an object. The counting also made it possible to hack the logic of the maze, skipping specific turns and overriding the value of *NODE*.

The switch case in the count consisted of three different cases: Find-first, Island, and Explore. Find-first uses the right-hand rule to find the first object which was placed in a fixed and known position in the maze. Once the object was found the robot turns around and a boolean is set to true, which changes the traversing to left-hand, this would ensure that the robot would travel back to the start. When the turn counter said the robot was back at the start and it left the black line, it dropped of the object and switched its mode to Island.

Island uses left-hand rule, except for the first three way intersection where we force it to turn right to ensure it goes into the island. If the robot find an object in the island it turns around, sets the boolean flag to true, and uses right-hand rule to go back to the start. When traveling back it turns left in the same three way intersection as well as skipping unnecessary turns. If the robot traverses the whole of the island without finding an object, it sets an boolean flag that says that the island is empty to true, it then changes mode to Explore and sets the counter to the value it would be by using explore from the starting position.

Explore also uses the left-hand rule, but is the case with the most special cases that needs to be handled. Firstly, if an object is detected the boolean flag is set to true but depending on where the robot is it either turns around and uses right-hand rule to get back or it uses a hard coded path that it knows is empty back to the start. Traversing back using right hand-rule also skips some intersections that are unnecessary, where it also overrides the value of the turn counter. If the object is found after the three way junction without a black line the robot does not have to go all they way back, instead it can take a shortcut to the start through space we have already gone through with the first two cases. So instead of using the right-hand rule all turns are specified. The explore case also has to handle the two parts of the path that has its black line removed. For the straight parts it simply uses hard coded speeds for the motors instead of using the PD controller. For the three way junction it uses a hard coded turn. If the robot goes all the way through the explore path it will have gone through the part of the maze where the first object is placed, this will check for objects that was placed behind the first object since it would get missed during case Find-first. It the robot traverses through all this space without finding an object, it sets a boolean flag that says that mode Explore is completely empty to true, it then changes its mode to Island and updates the turn counter.

Once three objects had been returned to the start it knows that it is done and starts dancing.

Example of how the loop of a case could look.
- Read the value of the IR sensors.
- Check if the robot is in a intersection, handle it if it is the case.
- Check if the robot is off the line, handle that situation.
- Check if the bumper switch is pressed in, pick up the object and set state to returning.
- Follow the black line using the PD controller.

3. Demo results

We had to perform a demo to show our implementation. This demo was conducted in the maze which can be seen in the figure below. At first we started by placing the robot in the start square where we then ran the calibration for the qtr package. Once the calibration was finished we let it start exploring the maze, as mentioned in the code example we start exploring the find first case and thus in figure 4 we explore the yellow area and found our first person. The robot then notices the person by triggering the bumper switch and then open grippers and picks the first person. It then switches to returning mode which is indicated by the green LED. It then traverses back to the start and drops of the first person there and switches cases to the Island case, this is depicted with the orange colored line. There it explores and finds person 2 which the robot acts the same as for person 1 and returns it to the start. Then when we swapped case to explore, the purple colored line, we ran into a problem on our first try. The robot got misaligned with the black line in the top left corner of the maze after the part that was missing the black line. This resulted in the robot not turning as planned which affects the counted nodes, this ment it would not drive correctly in the later part of the maze and we knew we had to intervene and restart the test.

The robot then got put back at the start and performed the cases again and got person 1, however when it was approaching person 2 a new problem occurred that the bumper switch accidently pushed person 2 over so it laid flat on the ground. During our testing before the demo we ran into this problem so we already knew it could handle pushing a person out even if the person was laying down. So we didn't intervene and let it continue which can be seen in figure 5. After it returned person 2 it continued and this time the PD regulator had been better and the robot fixed the section where the black line was missing. It moved over to person 3 and brought it back using a shortcut, which is executed if the robot find a person after a certain node in the explore case. The robot

brought person 3 successfully back to the start and did a victory dance. It solved the full maze with these positions in approx 3 min and 35 seconds.

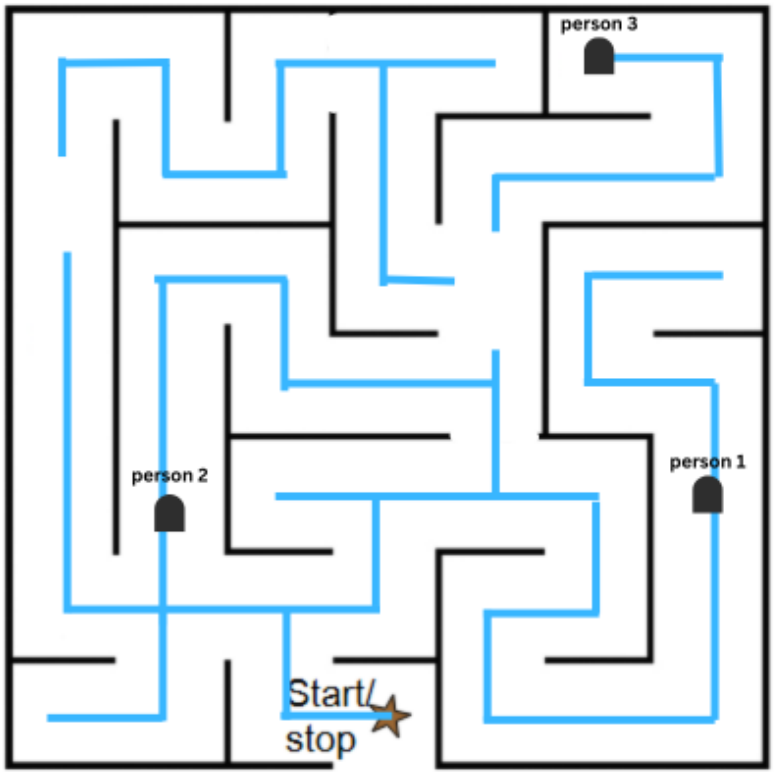


Figure 3: Maze during demonstration where the black lines are the walls of the maze and blue is the "black lines".

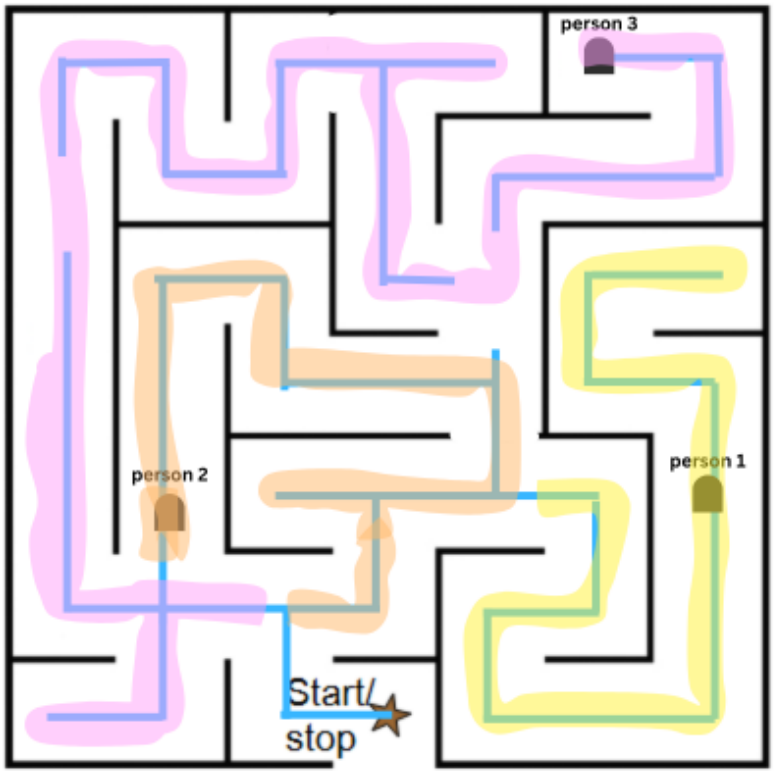


Figure 4: Maze during demonstration where the black lines are the walls of the maze and blue is the "black lines", also the colors represent the different cases mentioned in the code description.

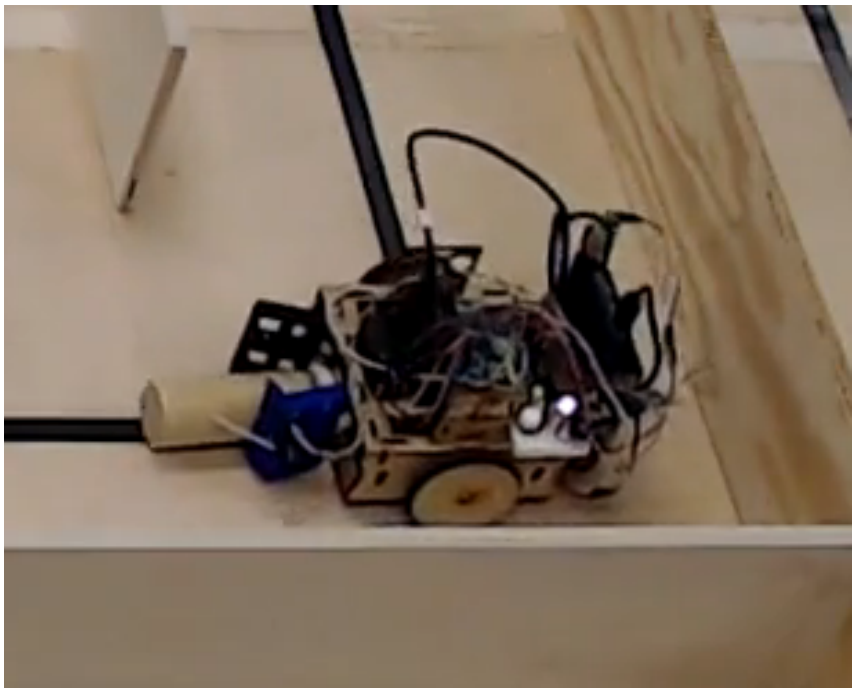


Figure 5: Maze during demonstration where the robot knocks over person 2.

4. Discussion

4.1. Design decisions

Most design decisions regarded the gripper. It was early decided that for moral reasons, each object was to be carried out individually. The group decided to follow the design principle of "Don't reinvent the wheel" to overall simplify the design of the robot.

As soon as the robot was able to drive, a drag test was preformed in order to asses the standard gripper. Its simplicity was its strong advantage, but it was not reliable to drag the objects. The objects tended to tip, either sideways or front to back due to the uneven ground surface. It was considered to scrap the standard gripper in exchange for a system that could grab and then lift the objects a few millimeters. This would entail a deeper concept design phase for the gripper which would result in more time being focused on the gripper instead of the whole robot. Before scrapping the standard gripper, the group decided to look at potential fixes for the problems that occurred. The concept decided to explore was larger gripper arms to support the object on the top and bottom, so that it could not pivot around any contact point. As a concept test, larger grippers were 3D-printed in PLA.

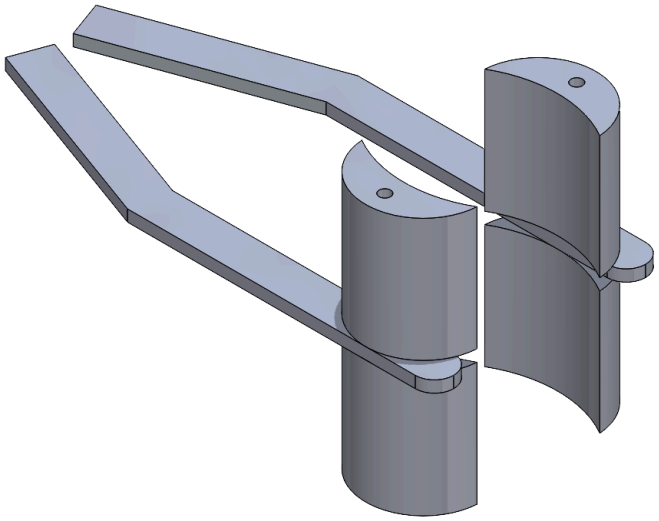


Figure 6: First iteration of gripper support.

While these helped greatly with side to side pivoting, they were hard to line up with the object reliably. The concept was reconsidered and the following figure shows the second iteration, utilizing a bracket design to improve the depth of the support.

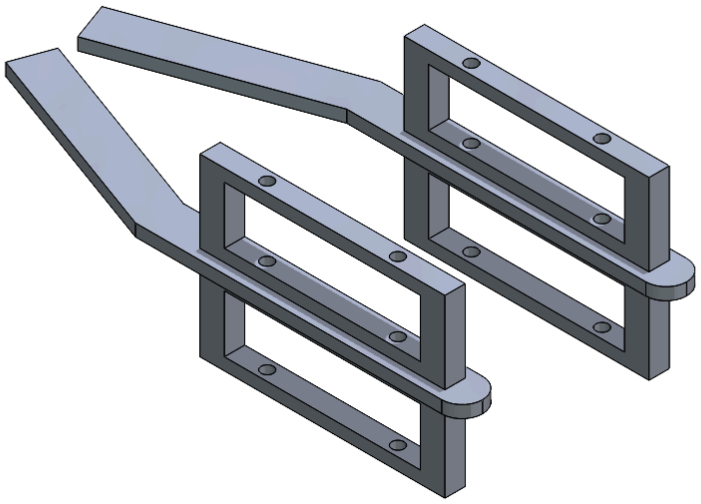


Figure 7: Second iteration of gripper support.

The second iteration showed good results with depth, and fairly good side pivot support. The brackets were intentionally printed thin in order to validate the design without wasting time on prints. They were however accidentally printed too deep and had to be shortened in testing.

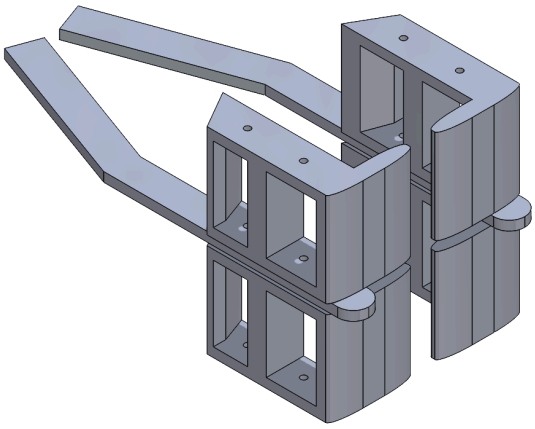


Figure 8: Third and final iteration of gripper support.

With the results from previous concepts, a final gripper support was designed and printed. This iteration had a thicker and higher support, with gates on the front in order to prevent the object from pivoting forwards or backwards. As a final touch, the first iteration of supports were added to the body of the robot to further prevent forward and backward movement. The final design shown in figure 1 proved to be a reliable gripper.

Sometimes the robot would not stop in time when an object was detected, although the grippers were stable enough to still grab the object and drag it to safety, it was deemed as a potential problem with the design. The bump switch antenna was adjusted which greatly increased the distance

allowed to brake. More intricate concepts were considered and tested, but ultimately the chosen antenna design was the most reliable due to its light weight and therefore high sensitivity to objects. The antenna could be adjusted more for an even more reliable detection and lower risk of driving into objects.

The battery carrier was designed to hold both of the power banks however this shifted the center of mass too much to the back and messed with the driving dynamics of the robot. Therefore one power bank was placed up front and one in the carrier. The carrier was designed to be easily removed from the robot, but still stable enough to carry the heavy power banks. The first iteration of the was made of laser cut MDF, glued together with two small 3D-printed brackets. The brackets were glued to the MDF to stay in place when the carrier was removed from the robot, and due to the design of the brackets, the weight was not dependant on the glued bracket. The laser cutter was chosen due to the simple geometry of the carrier.

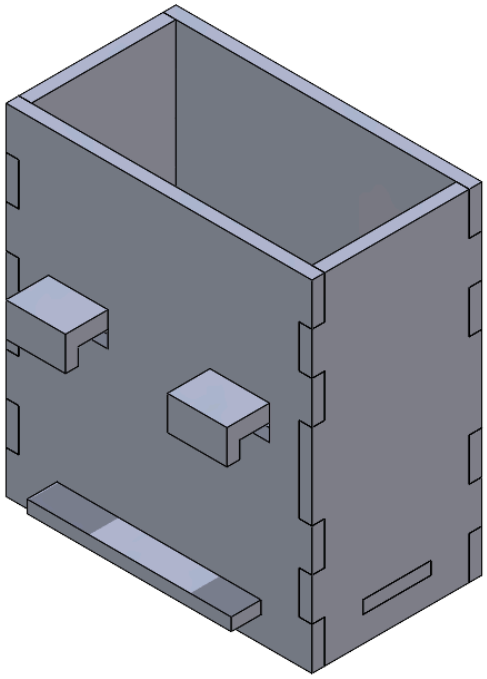


Figure 9: First iteration of battery carrier.

Due to the good results of the first iteration, no further iterations were deemed necessary. If another iteration was made, the thickness of the carrier could be smaller due to it only holding one battery.

It's also worth noting that the Ultrasonic sensors were tested to guide the robot where the black line was not available, as the brackets can be seen in figure 1 and 2, however these were very unreliable in code and would need some sort of digital filtering in order to be used successfully, as they were sending a lot of false readings. They were scrapped early in the design process but the brackets remained.

4.2. Results

The robot successfully achieved all of the objectives we designed it to do. Across repeated test runs, it managed to traverse the maze in its intended way whilst also being able to pick up and return the blocks to safety. Although we had to perform one restart at the demonstration, the line following,

turning, and object detection work as intended and with a good success rate. The PD controller stabilizes the robot very fast, which is great for stability and reliability, especially in the parts of the maze where there is no line to follow. The bumper button used to detect objects works for the most part, there are however times where it will trip over the blocks making it more difficult for the gripper to hold them.

4.3. Potential improvements

Even though the robot achieved its intended task, there are still many improvements that could be made. A container to hold the blocks would be a useful addition, as the current design requires the robot to repeatedly return to the start of the maze and transport one block at a time. Such a container would result in a significant time saving during the overall demonstration. In addition, a more reliable object detection system would be beneficial. The bumper button that is currently installed has a tendency to push the blocks over. This could be improved by lowering the impact point of the bumper button so that it contacts the block at a lower point. Alternatively, a completely new detection method, such as additional sensors or even a camera, could provide further improvements. However, these changes would increase system complexity and require additional development time.

5. Appendix

```
#include <QTRSensors.h>
#include <CytronMotorDriver.h>
#include <Servo.h>
CytronMD motorLeft(PWM_DIR, 5, 6);
CytronMD motorRight(PWM_DIR, 3, 11);

QTRSensors qtr;
const uint8_t SensorCount = 6;
uint16_t sensorValues[SensorCount];
const uint16_t BLACK_LINE = 750;
const uint16_t WHITE_LINE = 300;

Servo gripper;
const int GRIPPER_PIN = 8;
const int open = 105;
const int closed = 70;

const int LED_OTHER = 9;
const int BUMPER_PIN = 7;
const int BUTTON_PIN = 12;

// PID CONSTANTS
float Kp = 0.1;
float Kd = 1;

int16_t lastError = 0;
float integral = 0;

const int BASE_SPEED = 150;
const int MAX_SPEED = 255;
const int TURN_SPEED = 100;

// CASE HANDELING
enum RobotMode {
  MODE_FINDFIRST,
  MODE_ISLAND,
  MODE_EXPLORE,
  MODE_TEST
};
RobotMode mode = MODE_FINDFIRST;
bool RETURNING = false;
int NODE = 0;
int numSaved = 0;

bool middleEmpty = false;
bool exploreEmpty = false;
```

Listing 1:

```
  // ------------------------------------------------------------ FUNCTIONS
------------------------------------------------------------
// LINE DETECTION FUNCTIONS
int countBlackLines(){
  int count = 0;
  for(int8_t i = 0; i < SensorCount; i++){
    if(sensorValues[i] > BLACK_LINE) count++;
  }
  return count;

  bool leftSideBlack() {
  return sensorValues[0] > BLACK_LINE && sensorValues[1] > BLACK_LINE;
}

bool rightSideBlack() {
  return sensorValues[4] > BLACK_LINE && sensorValues[5] > BLACK_LINE;
}

bool middleOn() {
  return (sensorValues[2] > BLACK_LINE && sensorValues[3] > BLACK_LINE);
}

bool middleOff() {
  return (sensorValues[2] < WHITE_LINE && sensorValues[3] < WHITE_LINE);
}
// MOTOR FUNCTIONS
void setMotorSpeeds(int leftSpeed, int rightSpeed){
  if(leftSpeed >= 0){
    motorLeft.setSpeed(constrain(leftSpeed, 0, MAX_SPEED));
  }else{
    motorLeft.setSpeed(constrain(-255-leftSpeed, -MAX_SPEED, 0));
  }
  if(rightSpeed >= 0){
    motorRight.setSpeed(constrain(rightSpeed, 0, MAX_SPEED));
  }else{
    motorRight.setSpeed(constrain(-255-rightSpeed, -MAX_SPEED, 0));
  }
}

void turnAround(){
  setMotorSpeeds(-TURN_SPEED, TURN_SPEED);
  while(true){
    qtr.readCalibrated(sensorValues);
    if(middleOff()) break;
  }
  while(true){
    qtr.readCalibrated(sensorValues);
    if(middleOn()) break;
  }
  setMotorSpeeds(0, 0);
}
```

Listing 2:

```
  void turnLeftUntilLine() {
  setMotorSpeeds(-TURN_SPEED, TURN_SPEED);
  while(true){
    qtr.readCalibrated(sensorValues);
    if(middleOff()) break;
  }
  while(true){
    qtr.readCalibrated(sensorValues);
    if(middleOn()) break;
  }
  setMotorSpeeds(0, 0);
}

void turnRightUntilLine() {
  setMotorSpeeds(TURN_SPEED, -TURN_SPEED);
  while(true){
    qtr.readCalibrated(sensorValues);
    if(middleOff()) break;
  }
  while(true){
    qtr.readCalibrated(sensorValues);
    if(middleOn()) break;
  }
  setMotorSpeeds(0, 0);
}

// GRIPPER FUNCTION
void pickupObject(){
  setMotorSpeeds(0, 0);
  setMotorSpeeds(-BASE_SPEED, -BASE_SPEED);
  delay(150);
  setMotorSpeeds(0, 0);
  openGripper();
  setMotorSpeeds(BASE_SPEED, BASE_SPEED);
  delay(500);
  setMotorSpeeds(0, 0);
  closeGripper();
}
void dropObject(){
  setMotorSpeeds(0, 0);
  openGripper();
  setMotorSpeeds(-BASE_SPEED, -BASE_SPEED);
  delay(500);
  setMotorSpeeds(0, 0);
  closeGripper();
}
void openGripper(){
  for(int pos = closed; pos <= open; pos += 1){
    gripper.write(pos);
    delay(10);
  }
}
```

Listing 3:

```
void closeGripper(){
  for(int pos = open; pos >= closed; pos -= 1){
    gripper.write(pos);
    delay(10);
  }
}
// CALIBRATION FUNCTION
void calibrateAndRotate(){
  const int fullSweepIters  = 40;
  const int halfSweepIters  = fullSweepIters/2;

  int leftMotorSpeed = -TURN_SPEED;
  int rightMotorSpeed = TURN_SPEED;

  int nextSwitch = halfSweepIters;
  for (int i = 0; i < fullSweepIters*8.5; i++) {
    if(i == nextSwitch){
      setMotorSpeeds(0, 0);
      leftMotorSpeed = -leftMotorSpeed;
      rightMotorSpeed = -rightMotorSpeed;
      nextSwitch += fullSweepIters;
    }
    setMotorSpeeds(leftMotorSpeed, rightMotorSpeed);
    qtr.calibrate();
  }
  setMotorSpeeds(0, 0);
  setMotorSpeeds(-leftMotorSpeed, -rightMotorSpeed);
  while (true) {
    qtr.readCalibrated(sensorValues);
    if (middleOn()) {
      break;
    }
  }
  setMotorSpeeds(0, 0);
}
```

Listing 4:

```
// BUTTON FUNCTION
void buttonWait(){
  int lastButtonState = HIGH;
  int curButtonState = HIGH;
  while(true){
    curButtonState = digitalRead(BUTTON_PIN);
    if(curButtonState != lastButtonState){
      lastButtonState = curButtonState;
      if(curButtonState == HIGH) break;
    }
  }
}
bool bumperPressed(){
  return digitalRead(BUMPER_PIN) == LOW;
}
// TRAVERSING FUNCTIONS
void leftHandRule(bool leftBlack, bool rightBlack){
  bool rightLineMissed = false;
  bool leftLineMissed = false;
  setMotorSpeeds(BASE_SPEED, BASE_SPEED);
  do{
    qtr.readCalibrated(sensorValues);
    if(leftSideBlack() && !leftLineMissed) leftLineMissed = true;
    if(rightSideBlack() && !rightLineMissed) rightLineMissed = true;
  }while(sensorValues[0] > BLACK_LINE || sensorValues[5] > BLACK_LINE);
  delay(100);
  setMotorSpeeds(0, 0);
  if(leftBlack || leftLineMissed){
    turnLeftUntilLine();
  }
  else if(middleOn()){
    setMotorSpeeds(BASE_SPEED, BASE_SPEED);
  }
  else if(rightBlack || rightLineMissed){
    turnRightUntilLine();
  }
}
```

Listing 5:

```
void rightHandRule(bool leftBlack, bool rightBlack){
  bool rightLineMissed = false;
  bool leftLineMissed = false;
  setMotorSpeeds(BASE_SPEED, BASE_SPEED);
  do{
    qtr.readCalibrated(sensorValues);
    if(rightSideBlack() && !rightLineMissed) rightLineMissed = true;
    if(leftSideBlack() && !leftLineMissed) leftLineMissed = true;
  }while(sensorValues[0] > BLACK_LINE || sensorValues[5] > BLACK_LINE);
  delay(100);
  setMotorSpeeds(0, 0);

  if(rightBlack || rightLineMissed){
    turnRightUntilLine();
  }
  else if(middleOn()){
    setMotorSpeeds(BASE_SPEED, BASE_SPEED);
  }
  else if(leftBlack || leftLineMissed){
    turnLeftUntilLine();
  }
}
void followLine(int position){
  int16_t error = 2500 - position;
  int derivative = error - lastError;

  int output = Kp * error + Kd * derivative;
  lastError = error;
  int leftSpeed  = BASE_SPEED - output;
  int rightSpeed = BASE_SPEED + output;
  if(mode == MODE_EXPLORE && (NODE == 21 || NODE == 115)){
    leftSpeed = TURN_SPEED - output;
    rightSpeed = TURN_SPEED + output;
  }
  setMotorSpeeds(leftSpeed, rightSpeed);
}
```

Listing 6:

```
void victoryRoyal(){
  digitalWrite(LED_BUILTIN, LOW);
  digitalWrite(LED_OTHER, LOW);
  setMotorSpeeds(TURN_SPEED, -TURN_SPEED);
  while(true){
    digitalWrite(LED_BUILTIN, HIGH);
    openGripper();
    delay(50);

    digitalWrite(LED_OTHER, HIGH);
    closeGripper();
    delay(50);

    digitalWrite(LED_BUILTIN, LOW);
    openGripper();
    delay(50);

    digitalWrite(LED_OTHER, LOW);
    closeGripper();
    delay(50);
  }
}

// ------------------------------------------------------------ SETUP AND LOOP
------------------------------------------------------------
void setup() {
  Serial.begin(9600);

  qtr.setTypeAnalog();
  qtr.setSensorPins((const uint8_t[]){A0, A1, A2, A3, A4, A5}, SensorCount);

  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(LED_OTHER, OUTPUT);
  pinMode(BUMPER_PIN, INPUT_PULLUP);
  pinMode(BUTTON_PIN, INPUT_PULLUP);

  gripper.attach(GRIPPER_PIN);
  gripper.write(closed);
  openGripper();
  closeGripper();
```

Listing 7:

```
  digitalWrite(LED_BUILTIN, HIGH);
  buttonWait();
  digitalWrite(LED_BUILTIN, LOW);
  delay(150);
  calibrateAndRotate();
  digitalWrite(LED_BUILTIN, HIGH);
  buttonWait();
  digitalWrite(LED_BUILTIN, LOW);
  delay(150);

  mode = MODE_FINDFIRST;
}

void loop() {
  switch(mode){
    case MODE_FINDFIRST:
      handleModeFindFirst();
    break;
    case MODE_ISLAND:
      handleModeIsland();
    break;
    case MODE_EXPLORE:
      handleModeExplore();
    break;
    break;
    case MODE_TEST:
      uint16_t position = qtr.readLineBlack(sensorValues);
      bool leftBlack    = leftSideBlack();
      bool rightBlack   = rightSideBlack();
      int numBlackLines = countBlackLines();

      if((numBlackLines >= 3) && (leftBlack || rightBlack)){
        leftHandRule(leftBlack, rightBlack);
        return;
      }
      if(numBlackLines == 0){
        setMotorSpeeds(BASE_SPEED+10, BASE_SPEED);
        return;
      }
      followLine(position);
    break;
  }
}
```

Listing 8:

```
// ------------------------------------------------------------ MODE FUNCTIONS
------------------------------------------------------------
void handleModeFindFirst(){
  uint16_t position = qtr.readLineBlack(sensorValues);
  bool leftBlack    = leftSideBlack();
  bool rightBlack   = rightSideBlack();
  int numBlackLines = countBlackLines();

  if((numBlackLines >= 3) && (leftBlack || rightBlack)){
    digitalWrite(LED_BUILTIN, HIGH);
    if(RETURNING){
      NODE--;
      leftHandRule(leftBlack, rightBlack);
    }
    else{
      if(NODE == 301){
        NODE = 2;
        leftHandRule(leftBlack, rightBlack);
      }else{
        NODE++;
        rightHandRule(leftBlack, rightBlack);
      }
    }
    return;
  }
  digitalWrite(LED_BUILTIN, LOW);
  if(numBlackLines == 0){
    if(RETURNING && (NODE <= 0)){
      Serial.println("Home");
      dropObject();
      turnAround();
      RETURNING = false;
      digitalWrite(LED_OTHER, LOW);
      NODE = 0;
      numSaved++;
```

Listing 9:

```
      if(numSaved >= 3){
        victoryRoyal();
        return;
      }
      if(middleEmpty){
        if(exploreEmpty){
          mode = MODE_FINDFIRST;
        }
        else{
          mode = MODE_EXPLORE;
        }
      }else{
        mode = MODE_ISLAND;
      }
    }
    else{
      Serial.println("Wall to close");
      turnAround();
    }
    return;
  }
  if(bumperPressed() && !RETURNING){
    Serial.println("Found object!");
    pickupObject();
    turnAround();
    RETURNING = true;
    digitalWrite(LED_OTHER, HIGH);
    return;
  }
  followLine(position);
}
```

Listing 10:

```
void handleModeIsland(){
  uint16_t position = qtr.readLineBlack(sensorValues);
  bool leftBlack    = leftSideBlack();
  bool rightBlack   = rightSideBlack();
  int numBlackLines = countBlackLines();

  if((numBlackLines >= 3) && (leftBlack || rightBlack)){
    digitalWrite(LED_BUILTIN, HIGH);
    if(RETURNING){
      if(NODE == 5){
        NODE = 3;
        leftHandRule(leftBlack, rightBlack);
      }
      else if(NODE <= 3){
        leftHandRule(leftBlack, rightBlack);
        NODE--;
      }
      else {
        rightHandRule(leftBlack, rightBlack);
        NODE--;
      }
    }
    else{
      if(NODE <= 2){
        rightHandRule(leftBlack, rightBlack);
        NODE++;
      }
      else if(NODE == 10){
        middleEmpty = true;
        if(exploreEmpty){
          leftHandRule(leftBlack, rightBlack);
          NODE = 301;
          mode = MODE_FINDFIRST;
        }
        else{
          setMotorSpeeds(BASE_SPEED, BASE_SPEED);
          delay(250);
          NODE = 3;
          mode = MODE_EXPLORE;
        }
      }
```

Listing 11:

```
      else{
        leftHandRule(leftBlack, rightBlack);
        NODE++;
      }
    }
    return;
  }
  digitalWrite(LED_BUILTIN, LOW);
  if(numBlackLines == 0){
    if(RETURNING && (NODE <= 0)){
      Serial.println("Home");
      dropObject();
      turnAround();
      RETURNING = false;
      digitalWrite(LED_OTHER, LOW);
      NODE = 0;
      numSaved++;
      if(numSaved >= 3){
        victoryRoyal();
      }
      else{
        mode = MODE_EXPLORE;
      }
    }
    else{
      turnAround();
    }
    return;
  }
  if(bumperPressed() && !RETURNING){
    Serial.println("Found object!");
    pickupObject();
    turnAround();
    RETURNING = true;
    digitalWrite(LED_OTHER, HIGH);
    return;
  }
  followLine(position);
}
```

Listing 12:

```
void handleModeExplore(){
  uint16_t position = qtr.readLineBlack(sensorValues);
  bool leftBlack    = leftSideBlack();
  bool rightBlack   = rightSideBlack();
  int numBlackLines = countBlackLines();

  if((numBlackLines >= 3) && (leftBlack || rightBlack)){
    digitalWrite(LED_BUILTIN, HIGH);
    if(RETURNING){
      if((NODE >= 220) && (NODE <= 231)){
        NODE--;
        leftHandRule(leftBlack, rightBlack);
      }
      else if(NODE == 219){
        NODE = 0;
        leftHandRule(leftBlack, rightBlack);
      }
      else if((NODE >= 115) && (NODE <= 120)){
        NODE--;
        leftHandRule(leftBlack, rightBlack);
      }
      else if(NODE == 114){
        NODE--;
        rightHandRule(leftBlack, rightBlack);
      }
      else if(NODE >= 111 && NODE <= 113){
        NODE--;
        leftHandRule(leftBlack, rightBlack);
      }
      else if(NODE == 110){
        NODE = 0;
        leftHandRule(leftBlack, rightBlack);
      }
      else if(NODE == 6){
        setMotorSpeeds(BASE_SPEED, BASE_SPEED);
        delay(250);
        NODE = 2;
      }
      else{
        NODE--;
        rightHandRule(leftBlack, rightBlack);
      }
    }
```

Listing 13:

```
  else{
    if(NODE == 39){
      rightHandRule(leftBlack, rightBlack);
      NODE = 6;
      mode = MODE_ISLAND;
    }
    else if(NODE == 30){
      NODE++;
      rightHandRule(leftBlack, rightBlack);
    }
    else if(NODE == 18){
      exploreEmpty = true;
      NODE++;
      leftHandRule(leftBlack, rightBlack);
    }
    else if(NODE == 3){
      NODE++;
      rightHandRule(leftBlack, rightBlack);
    }
    else{
      NODE++;
      leftHandRule(leftBlack, rightBlack);
    }
  }
  return;
}
digitalWrite(LED_BUILTIN, LOW);
if(numBlackLines == 0){
  if(NODE == 7){
    setMotorSpeeds(BASE_SPEED+5, BASE_SPEED);
  }
  else if(NODE == 15){
    setMotorSpeeds(0,0);
    setMotorSpeeds(BASE_SPEED, BASE_SPEED);
    delay(925);
    setMotorSpeeds(0,0);
    setMotorSpeeds(-TURN_SPEED, TURN_SPEED);
    delay(750);
    setMotorSpeeds(BASE_SPEED, BASE_SPEED);
    do{
      qtr.readLineBlack(sensorValues);
    }while(countBlackLines() == 0);
  }
```

Listing 14:

```
    else if(NODE == 115 || NODE == 21){
      setMotorSpeeds(TURN_SPEED+5, TURN_SPEED);
    }
    else if(RETURNING && (NODE == 0)){
      Serial.println("Home");
      dropObject();
      turnAround();
      RETURNING = false;
      digitalWrite(LED_OTHER, LOW);
      NODE = 0;
      numSaved++;
      if(numSaved >= 3){
        victoryRoyal();
        return;
      }
      if(middleEmpty){
        if(exploreEmpty){
          mode = MODE_FINDFIRST;
        }
        else{
          mode = MODE_EXPLORE;
        }
      }
      else{
        mode = MODE_ISLAND;
      }
    }
    else{
      turnAround();
    }
    return;
  }
  if(bumperPressed() && !RETURNING){
    Serial.println("Found object!");
    pickupObject();
    RETURNING = true;
    digitalWrite(LED_OTHER, HIGH);
    if(NODE != 21){
      turnAround();
    }
    else if(NODE == 21){
      NODE = 114;
      setMotorSpeeds(TURN_SPEED, TURN_SPEED);
      delay(250);
    }

    if((NODE >= 15) && (NODE <= 20)){
      NODE = NODE + 100;
    }else if((NODE >= 22) && (NODE <= 31)){
      NODE = NODE + 200;
    }
    return;
  }
  followLine(position);
}
```

Listing 15:

6. References

Bibliography