How to develop a system that leverages sensors to avoid obstacles.

Jonas Dolezal

Lumiere Education - Individual Research Program

August 23, 2024

Contents

1 Introduction

In today's rapidly evolving and increasingly integrating world of AI, an important counterpart must not be forgotten: vehicles. Most systems of AI we use today are ones used in browsers and apps, but to show real impact in the world, vehicles and robots have to be implemented and used. One distinction between challenges these two areas have to face is the presence of real world obstacles and bumps on a road. Vehicles have to adapt to the world we live in, while online chatbots create a comfortable world for themselves. These obstacles for real world vehicles can oftentimes pose an important threat that cannot be overlooked. Have you ever seen a robot that got stuck on a curb and couldn't move on? Quite a frequent occurrence. For these reasons vehicles must be able to see around them and act accordingly, avoiding any potential threats and preventing any sort of malfunction. Perception systems must reliably detect and avoid obstacles as they move from point A to point B, in order to prevent any damage from collisions. This makes obstacle avoidance crucial and fundamental to include in any system. This paper attempts to showcase how such a fundamental concept can be done through the use of sensors, their fusing and vector field histograms. Another key aim of this paper is to show how this goal can be achieved with as small of a budget as possible. Since we can expect robots to become a hit in the tech world (as they already are), we can expect many firms to try and achieve the lowest cost possible, in order to maximise their supply potential. This paper will attempt to show a low-cost, safe and reliable solution as to how such a crucial task can be achieved. The low-cost nature of this investigation is important for

many reasons: The robot production can be more scaled and larger, the low cost allows a swifter integration into less economically developed countries and possibly accelerate innovation as funds can be investigated elsewhere. Effective systems can be crucial in delivering life saving help in environments it hasn't seen before, like medical supplies or basic human resources. This paper aims to showcase the reliability and low cost of a system capable of obstacle avoidance through the use of vector field histograms. Also this paper aims to inspire another generation of developers excited to take these ideas further and prosper in the ever-expanding world of tech and AI.

2 Literature Review

Existing literature has a strong focus on alternative ways to solve the mentioned problem. It focuses greatly on the use of several types of sensors ,including LIDAR, cameras, ultrasonic sensors, and infrared sensors, each with its own distinct advantages and limitations.

By using several sensors, many studies focus on a different kind of fusion than the fusion introduced here. Existing literature discusses how different information from different sensors can be used to achieve obstacle avoidance and general perception of surroundings. This method significantly improves the performance of autonomous systems by leveraging the strengths of each sensor type.

Despite these clear advantages, most literature ends up describing high-cost solutions, often using expensive sensors, like LIDAR, and complex algorithms. This paper distinguishes itself by focusing on accessibility and low-cost so-

lutions, a relatively unexplored area in current literature. By prioritizing affordability, this paper aims to make obstacle avoidance and perception systems more accessible to a broader audience, including hobbyists, developers or companies with limited budgets, seeking to lower costs as much as possible.

3 System Design

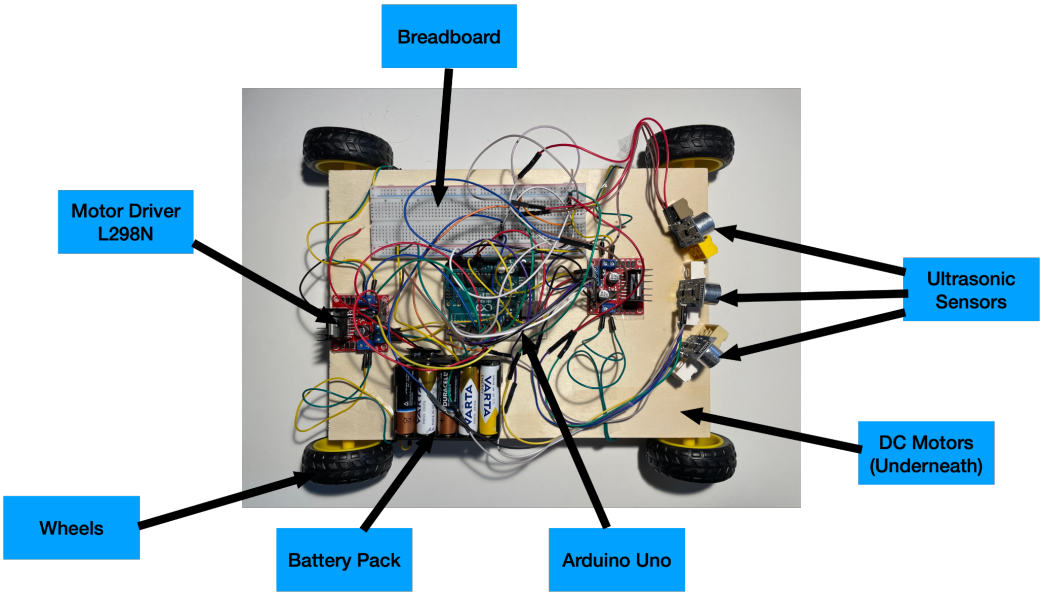

Figure 1: Vehicle Parts Diagram

3.1 Hardware

The hardware of this system will be centred around the Arduino Uno R3. This will be the brain of the vehicle and is in charge of coordinating the system's inputs and outputs. This has been chosen because of its relatively low cost, widespread use, and ease to implement, as it is a standard part used in many projects and even industrial manufacturing.

The component responsible for detection and perception of the surroundings will be the ultrasonic sensor HC-SR04. There will be more of these sensors, allowing the system to see in more directions, enhancing the ability to detect and avoid obstacles. These sensors have been selected because of their very low cost and simplicity. These sensors emit ultrasonic waves and measure the time it takes for the waves to bounce back from an obstacle, which is then used to calculate the distance to the obstacle. Therefore, they give one simple, frequently updated reading of how far an object is in one specific direction.

To move, the system uses simple DC motors, which can also be seen in many other small-scale vehicles. These motors have been chosen for their low cost and ease of connection with the Arduino. Coupled with these DC motors are motor drivers L298N, which allow the Arduino to control which motor is on at what time, how fast it's going, allowing the vehicle to accelerate, stop, and make turns, all crucial for simple steering and control of a vehicle.

The vehicle's chassis is a basic, lightweight, wooden frame designed to house the Arduino, sensors, motor driver, and power supply. The power supply, typically a set of AA batteries or a rechargeable battery pack, provides the necessary power for the entire system.

Along with all of the parts mentioned, a breadboard is being used to distribute the different connections to more places, as the Arduino itself doesn't have enough connectors to accommodate each connection of each part attached to it. This means that the breadboard is distributing 5V power and the ground connections as well.

All of the components mentioned above are low cost, easily accessible,

and widespread parts used very often in creation of small-scale vehicles. This aligns with the aim of this paper, as it allows for scalability and accessibility of such a system.

3.2 Software

The code for this vehicle can be found in Appendix 1. The code starts by setting predetermined parameters, such as where each connection from different parts leads into the Arduino and some constants such as the motor speed and detection threshold for the ultrasonic sensors. The language used is the Arduino modified version of C, making it a compilable language. The structure of the code consists of one main loop running again and again several times a second and within that loop there are many functions being run. Regardless of the vehicle situation, the ultrasonic reading function is called. This function will read the data from the ultrasonic sensors and store them in variables. Then, depending on the values that the sensors read, the vehicle makes a decision and calls one of four functions: Move forward; instructing the vehicle to move forward, Turn Right; temporarily setting the DC motors on the right side to move in reverse, Turn Left; temporarily setting the DC motors on the left to move in reverse and Stop Moving; a function that will stop all motors immediately, wait for a little moment and then move back a little bit.

3.3 Combination

The two systems described above work separately and on different levels. They of course have to be effectively combined to create a system that can achieve obstacle avoidance with what hardware it has and what software it can implement. A few obstacles had to be overcome in order to effectively combine these two systems. As mentioned previously, the Arduino itself has a limited number of ports for parts to connect to. This means that there had to be an innovative solution for using these ports effectively. One important thing to remember with ultrasonic sensors is the fact that they have four pins that each need to be connected. They include: one providing power, another a ground connection, a trigger pin and an echo pin. This means that if there was no combining of outputs from the Arduino, ultrasonic sensors alone would need twelve pins, leaving minimal room for motor drivers and battery pack connection. The obvious easy fix is to use one 5V output for all of the ultrasonic sensors, providing them all with power and one ground connection, completing a circuit, allowing each of the sensors to work. This easy fix brings down the number of needed ports for ultrasonic sensors down to eight. This is, however, still too many. The main improvement is the combination of the trigger pin. One important thing to remember is that the ultrasonic sensor works in the way that it receives a "trigger" from the trigger pin, signalling that the sensor should send out the ultrasonic wave. Then, it will wait for the signal to come back and then will "echo" the time taken for the signal to bounce off an object and to come back through the echo pin away. This means that it is crucial for each echo pin to be connected to a

different pin on the Arduino. In the case of the trigger pin, the sensors can share one pin, wired through breadboard, for all the sensors, as each trigger isn't exclusive to one ultrasonic sensor. This is a major improvement for the number of pins that the ultrasonic sensors take up on the Arduino. One consideration that is important to remember when writing code for this is that the code will be looking for an echo for a set amount of time by default, but this might not always work because an ultrasonic sensor might take a different amount of time to respond, because the signal it sends out might be travelling for longer, as it has to travel. There is an easy fix for that and it is that the code can wait for a long enough period of time to allow even the longest, maximum signals to come back.

Another thing that can be considered as combining two systems is combining the different power outputs. The battery pack itself provides 12V, but not everything in the system can handle so many volts. The Arduino itself can send out 5V outputs, which are used for the ultrasonic sensors. The conversion happens in the L298N Motor driver. The battery pack connects to the power rails in the Arduino and both of the motor drivers are connected to the rails. The Arduino is not powered using the battery pack directly, but rather takes a 5V output from the front motor driver. The L298N motor drivers prove themselves to be particularly useful and cost effective, as they can take in a 12V input, power the 2 DC motors they might be connected to and then output a 5V voltage back to an Arduino. Another thing that is important is that the Arduino and the motor driver have to have the same ground connection, meaning more connections have to be made to ensure safety.

4 Methods

This section describes how obstacle avoidance is actually achieved. Having discussed the vehicle's construction, we can now make an attempt at the main aim of this paper. This section will be focused on vector field histograms and how those can be used to our advantage to make a system capable of obstacle avoidance. Essentially, what we get from the ultrasonic sensor readings are eyes that tell us how far we can go in each direction we measure without crashing. Of course, the sensors don't see in every direction, as there is a limited number of them, but for the purpose of driving forward and avoiding any potential dangers, they are sufficient. The sensors themselves are placed in such an orientation, that one looks forward and the other two look 45 θ to each side.

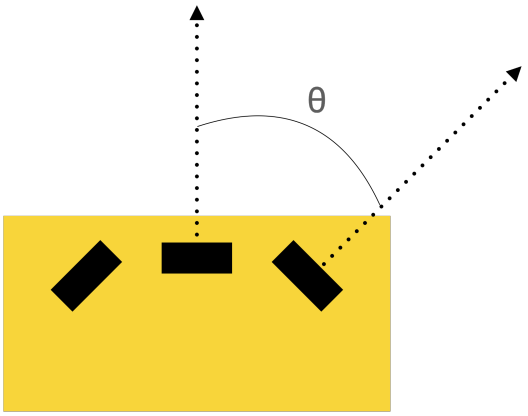

Figure 2: Sensor Separation

To have this optimal angle, there are a few considerations that have to be considered. One being that if this angle is too small, the sensors won't be able to detect many different objects. This is because each of these sensors has a detection cone with angle ϕ. Refer to the diagram below.

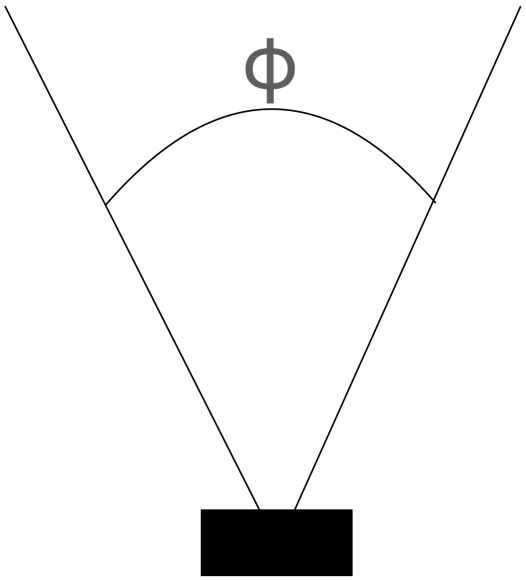

Figure 3: Sensor Vision Angle

If these sensors are placed too close to each other (θ is small), the cones of the sensors will overlap, meaning that there will be a loss of information. Also, if there happens to be an object at that angle, the vehicle will want to stop rather than turn away, essentially deeming the sideways sensors to be useful for only very specific objects placed at specific angles to the sides of the vehicle. This same problem also occurs when the angle between the edges of the cones is too small. When there is an object in the cone of the sideways sensor, the code will instruct the vehicle to turn to one side. This however

takes some time and the vehicle itself will have momentum and won't be able to stop immediately. This might not seem like a big problem, but because the vehicle has momentum from driving forward, it will drift further before it can turn and the object will not appear in the sensor cone of the vented sensor, forcing the vehicle to stop and retreat, rather than turn away and keep driving. Essentially, the objects sideways would be detected too late.

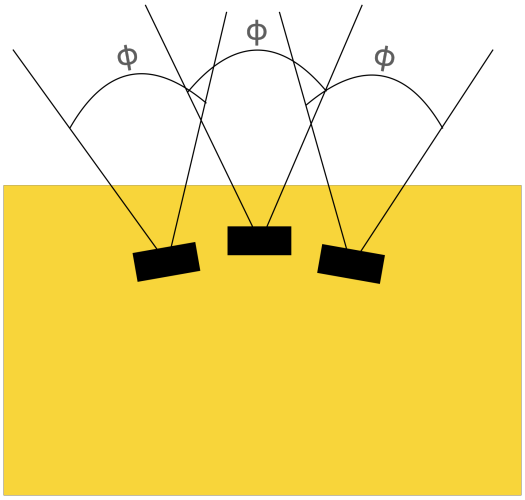

Figure 4: Vehicle Parts Diagram

On the flip side, the sensors can be too far from each other (θ is large). This also poses a problem, because the vehicle could be approaching an object or a wall at such an angle that it isn't within detection thresholds of any of the sensors and "sneaks through". This is very important for things like poles or objects, as they would truly be unnoticed, until the vehicle would crash.

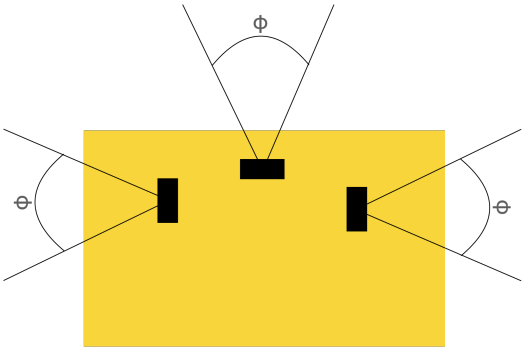

Figure 5: Small Theta Diagram

Therefore there has to be a centre point found and this proved to be around the 45 degree mark.

$\theta = 45$

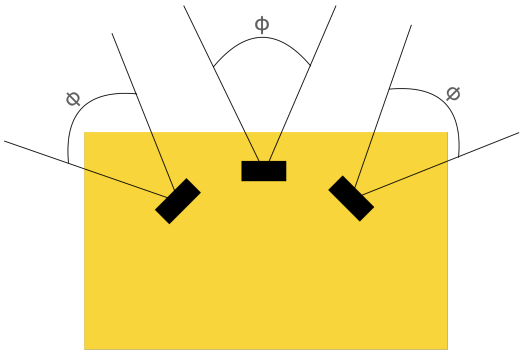

Figure 6: Large Theta Diagram

Now that the vehicle has relevant and useful data from the sensors, we can get to where this vehicle makes decisions. A lot of this decision making had to be fine tuned in real life, but it follows a general pattern. The sensors keep reading until one of them detects a distance lower than a predetermined value of the detection threshold. When this condition is met, the vehicle makes

either a turn, or a stop, as described above. This ensures that the vehicle won't crash, as because of the optimal placement of the sensors, we can avoid any obstacle that might cause a problem.This then allows the vehicle to turn and manoeuvre itself so that all the sensors see objects further than ones that would cause danger. Now the vehicle should be able to navigate through different courses without crashing and colliding with any barriers.

There still however remains one very important question. As is with most predetermined values set by a creator, there has to be a reason as to why it is set at that. The detection threshold has been mentioned above, but there was no real value for it and the question now becomes apparent: What value should it be set at? Let's repeat our flow of thought similar to above with extremes.

If the detection threshold would be too small, the sensors wouldn't trigger any sort of turn or stop until the wall or object got very close. This is an issue, as usually when the vehicle is already so close to the object, it will have some momentum and won't be able to stop in time, leading to a collision: something we are not looking for.

The other extreme is that the detection threshold would be set way too high. This would make the vehicle very "careful", meaning it would start turning away and stopping when it sees an object, which wouldn't pose much of a threat, far away. As it will come to show, this weariness will mean that the vehicle won't even be able to navigate a track at all.

Therefore we have determined our extremes and it leaves us with attempting to find the optimal threshold distance. We will be conducting an experiment to try and find the optimal threshold for a track set up.

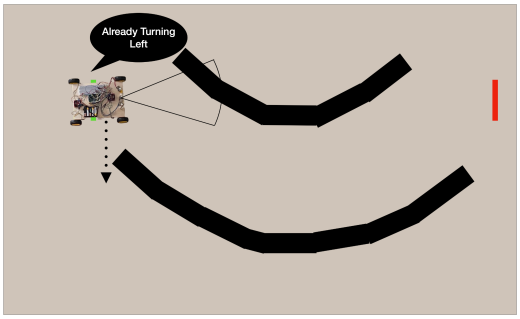

Figure 7: Large Detection Threshold

One last consideration to take into account is the fact that we might not be looking for an optimal threshold distance but rather an optimal detection threshold as a ratio to the width of the track. The ratio might be significant, because the track widths can vary in real world situations and if the detection threshold is set at some distance and the track width increases, the vehicle will seem to stick to one wall. The detection threshold, seen as a ratio to the width of the track, allows the vehicle to be entered towards the middle of the track, potentially making the journey safer because of less imminent dangers that the walls pose.

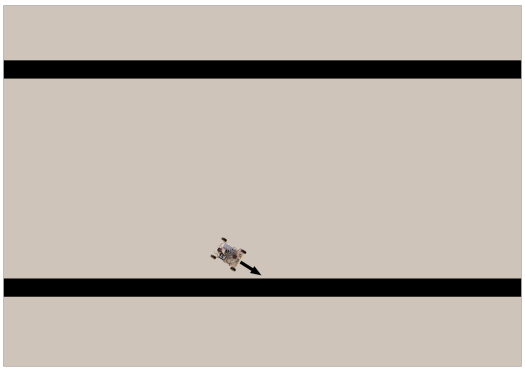

Figure 8: Small Detection Threshold Ratio

5 Experiment

Detection Threshold Optimisation Experiment

Control Variables: Track

Dependent Variables: Time taken, success rate, number of collisions

Independent Variable: Detection threshold

This experiment aimed to find the most optimal detection threshold ratio for a small cart, aiming to minimize the time taken to pass through a constant track, reduce the number of collisions with barriers, and maximize the success rate of passing through the course. The track has a set start point and end point, which were used to measure the time taken for a ride. The stopwatch was started when the vehicle first started moving and stopped when the vehicle touched the final area. For each detection threshold, 7 repetitions were performed, recording all important information during the run. This data was then averaged or summed and analyzed.

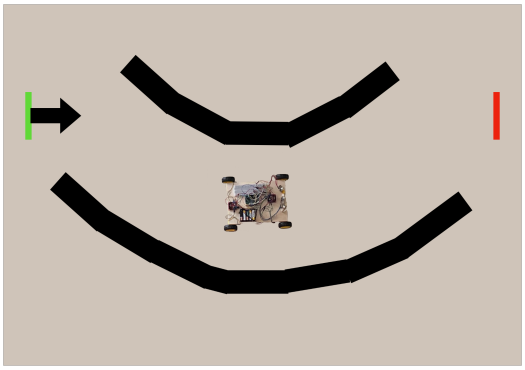

Figure 9: Test Track

Hypothesis:

From this experiment, we aim to observe clear trends. As the detection threshold increases (and thus the threshold ratio), the number of collisions should decrease, as the robot will become more "careful" and will try stopping earlier to avoid collisions. The time taken to pass the course should slightly decrease and then plateau, because as the threshold increases, the reduced number of collisions is expected to make the route through the track more efficient, resulting in a shorter time. However, if the detection threshold increases too much, we anticipate the robot will become excessively "careful" and may fail to complete the track. Consequently, the success rate should also decrease.

The raw data can be found in Appendix 2.

From this data, we can plot each parameter as a function of the detection threshold and obtain the following results:

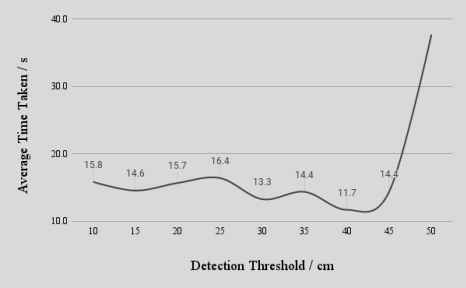

Figure 10: Time vs Detection Threshold Plot

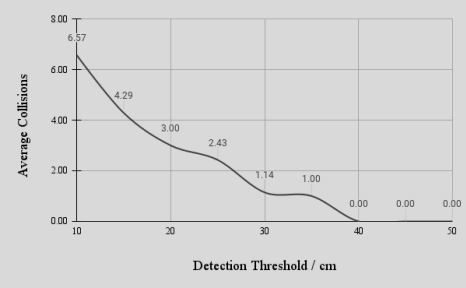

Figure 11: Collisions vs Detection Threshold Plot

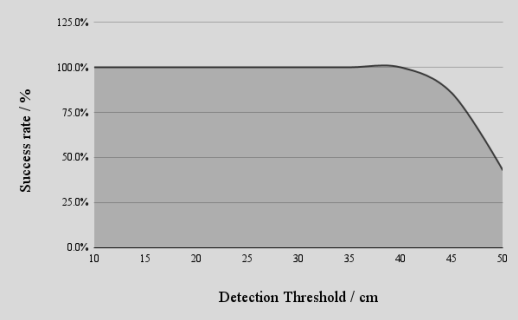

Figure 12: Success Rate vs Detection Threshold Plot

Analysis

From the plots above there are several observations we can make. First let's have a look at the success rate (Figure 12.). We can see that the robot successfully passed the track for most of the small detection thresholds, but then started occasionally failing once we got to the higher thresholds. This can be explained by the phenomenon that we described earlier. The robot becomes too "careful". When the robot tries to drive through the track it sees a wall from far away and immediately starts turning away from it. Through this though it might turn to see a wall closer and turn to the other side away from the other wall. Eventually the robot ends up in this loophole where it ends up just "wiggling" left and right and not advancing forward at all. That is why the success rate dropped towards the end.

To further understand the success rate, we have to have a look at the number of collisions the robot had (Figure 11). There is a clear downwards trend as we have hypothesised. This is because the vehicle was always given more time to react to a wall as the detection threshold increased, therefore meaning a very high number of collisions when the detection threshold was low.

Combined with the success rate we can see that at low detection thresholds the robot still was successful, but with lots of collisions, meaning that it stumbled its way through the track hitting many walls.

Finally looking at the time taken (Figure 11) we can see that despite the very spiky nature of the graph there is a minimum at around 40 cm. The time values do change quite a bit for different thresholds, meaning that the time taken is very sensitive to other factors, which could possibly include the angle of release, current state of the track and despite my efforts to keep them controlled, they might have moved by small amounts, possibly changing the time taken.

Overall we can draw a conclusion for this experiment and the aim was to find the optimal detection threshold ratio. Overall, we can state that the optimal ratio could be set at 40cm, or slightly less, to still gain the benefits of the faster time taken, but to not transfer into the lower success rate that can be observed at the next detection threshold. Therefore around 38 is optimal. This however isn't the thing we were looking for. The detection threshold ratio to the width of the track is a more useful value, as then the track and vehicle can be scaled. The track width was 70cm, meaning that the ratio of 38cm to 70cm gives us 0.543. Therefore we can assume that the detection threshold should always be set at roughly 54

6 Discussion

This paper isn't perfect. There are obviously many things that could have been done better and many factors that could be considered but weren't. Some of these are further explanations, notes, and comments on what has been shown so far.

- **Momentum of Cars:** At the detection threshold of 35 cm, the car often had at least one collision. This was the same collision in every run. It was caused because the car had a run-up into the first turn and, when it wanted to stop, it still had momentum and hit the wall. Within the track, it mostly did not crash anymore, indicating that this might be a good detection threshold, but only if the car was a bit slower.

- **Sensor Vision:** Another possible consideration is that the sensors are placed at some height above the ground and don't see the full height of the car itself. This could lead to them not detecting curbs or low-placed objects, or missing objects suspended in the air, which could pose a threat to the components. A potential solution could be to make the car slimmer or to increase the number of sensors.

- **Hardware/Software Improvement:** To improve the system and reduce the number of pins the sensors use on the Arduino, one Echo pin could be used. This would require the ultrasonic sensors to send information back in sequence rather than simultaneously. However, this still needs to maintain a quick response time and address issues that can arise from delayed signals. This improvement would allow for

more ultrasonic sensors to be added, enhancing the ability for obstacle avoidance.

7 Conclusion

From this paper we can gain a valuable insight into what it means to construct a simple vehicle avoidance system, at a low cost. There has been demonstrated a method of what to use and how to calibrate a system like this for individuals or groups to take away and recreate, adding their own capabilities and improving upon the system.

This paper aimed to showcase a system of obstacle avoidance while maintaining a low cost. Obstacle avoidance has been clearly achieved, demonstrated by the fact that the vehicle itself had a high success rate during the experiment to find the optimal detection threshold. Also a question of safety was raised earlier and this goal could also be seen as achieved, because the vehicle at the optimal detection threshold achieved no collisions with its surroundings, proving that the goal of not damaging itself, its contents and the surroundings was achieved. The secondary goal of this research paper was to build this system at a low cost. This has also been achieved. Of course, the term "low cost" is subjective, but in comparison to other research papers showcasing similar things, this one has achieved a much lower cost, using easy to access, cheap components for no more than a few tens of dollars. The exact prices are not mentioned here and this is because they can change from region to region and highly depend on how the parts are delivered, what quantities etc.

Overall, this research paper has effectively achieved its goals and proved

that systems capable of obstacle avoidance while maintaining a low cost are possible.

8 References

- Zhang, J., Hou, J., Hu, J., Zhao, C., Xu, Z., and Cheng, C., 2021. UGV autonomous driving system design for unstructured environment. Proceedings of the 40th Chinese Control Conference, July 26-28, Shanghai, China.

- Guastella, D.C. and Muscato, G., 2021. Learning-based methods of perception and navigation for ground vehicles in unstructured environments: A review. Sensors, 21, p.73. Available at: https://dx.doi.org/10.3390/s21010073.

- Borenstein, J. and Koren, Y., 1991. The vector field histogram—Fast obstacle avoidance for mobile robots. IEEE Transactions on Robotics and Automation, 7(3), pp.278-288.

9 Appendices

```

const int trigPin = A0;    // Trigger pin
const int echoPin1 = A1;   // Echo pin for Sensor 1
const int echoPin2 = A2;   // Echo pin for Sensor 2
const int echoPin3 = A3;   // Echo pin for Sensor 3

long duration1, duration2, duration3;
int distance1, distance2, distance3;

// Define motor control pins for L298N
const int ENA1 = 3;  // PWM pin for Motor 1
const int IN1_1 = 2;
const int IN2_1 = 4;

const int ENA2 = 6;  // PWM pin for Motor 2
const int IN1_2 = 5;
```

```
17 const int IN2_2 = 7;
18
19 const int ENA3 = 10;  // PWM pin for Motor 3
20 const int IN1_3 = 8;
21 const int IN2_3 = 9;
22
23 const int ENA4 = 11;  // PWM pin for Motor 4
24 const int IN1_4 = 12;
25 const int IN2_4 = 13;
26
27
28 int detection_threshold = 38;  // cms
29
30
31 int wheel_speed = 70;
32
33 void setup() {
34   Serial.begin(9600);
35   pinMode(trigPin, OUTPUT);
36   pinMode(echoPin1, INPUT);
37   pinMode(echoPin2, INPUT);
38   pinMode(echoPin3, INPUT);
39
40   pinMode(ENA1, OUTPUT);
41   pinMode(IN1_1, OUTPUT);
42   pinMode(IN2_1, OUTPUT);
43
44   pinMode(ENA2, OUTPUT);
45   pinMode(IN1_2, OUTPUT);
46   pinMode(IN2_2, OUTPUT);
47
48   pinMode(ENA3, OUTPUT);
49   pinMode(IN1_3, OUTPUT);
50   pinMode(IN2_3, OUTPUT);
51
52   pinMode(ENA4, OUTPUT);
53   pinMode(IN1_4, OUTPUT);
54   pinMode(IN2_4, OUTPUT);
55
56   Serial.println("Setup␣Complete");
57 }
58
59 int measureDistance(int trigPin, int echoPin) {
```

```
  long duration;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH, 70000);  //70ms

  if (duration == 0) {
    Serial.println("No echo received");
    return -1;
  } else {
    int distance = duration * 0.034 / 2;
    Serial.print("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");
    return distance;
  }
}

void moveForward() {
  Serial.println("Moving Forward");
  analogWrite(ENA1, wheel_speed);
  digitalWrite(IN1_1, LOW);
  digitalWrite(IN2_1, HIGH);
  analogWrite(ENA2, wheel_speed);
  digitalWrite(IN1_2, LOW);
  digitalWrite(IN2_2, HIGH);
  analogWrite(ENA3, wheel_speed);
  digitalWrite(IN1_3, LOW);
  digitalWrite(IN2_3, HIGH);
  analogWrite(ENA4, wheel_speed);
  digitalWrite(IN1_4, LOW);
  digitalWrite(IN2_4, HIGH);
}

void turnLeft() {
  Serial.println("Turning Left");
  analogWrite(ENA1, wheel_speed);
  digitalWrite(IN1_1, LOW);
  digitalWrite(IN2_1, HIGH);
```

```
103   analogWrite(ENA2, wheel_speed);
104   digitalWrite(IN1_2, HIGH);
105   digitalWrite(IN2_2, LOW);
106   analogWrite(ENA3, wheel_speed);
107   digitalWrite(IN1_3, LOW);
108   digitalWrite(IN2_3, HIGH);
109   analogWrite(ENA4, wheel_speed);
110   digitalWrite(IN1_4, HIGH);
111   digitalWrite(IN2_4, LOW);
112   delay(100);
113 }
114
115 void turnRight() {
116   Serial.println("Turning␣Right");
117   analogWrite(ENA1, wheel_speed);
118   digitalWrite(IN1_1, HIGH);
119   digitalWrite(IN2_1, LOW);
120   analogWrite(ENA2, wheel_speed);
121   digitalWrite(IN1_2, LOW);
122   digitalWrite(IN2_2, HIGH);
123   analogWrite(ENA3, wheel_speed);
124   digitalWrite(IN1_3, HIGH);
125   digitalWrite(IN2_3, LOW);
126   analogWrite(ENA4, wheel_speed);
127   digitalWrite(IN1_4, LOW);
128   digitalWrite(IN2_4, HIGH);
129   delay(100);
130 }
131
132 void stopMoving() {
133   Serial.println("Stopping");
134   analogWrite(ENA1, 0);
135   digitalWrite(IN1_1, LOW);
136   digitalWrite(IN2_1, LOW);
137   analogWrite(ENA2, 0);
138   digitalWrite(IN1_2, LOW);
139   digitalWrite(IN2_2, LOW);
140   analogWrite(ENA3, 0);
141   digitalWrite(IN1_3, LOW);
142   digitalWrite(IN2_3, LOW);
143   analogWrite(ENA4, 0);
144   digitalWrite(IN1_4, LOW);
145   digitalWrite(IN2_4, LOW);
```

```
146
147   delay(1000);
148   Serial.println("Reversing");
149   analogWrite(ENA1, wheel_speed);
150   digitalWrite(IN1_1, HIGH);
151   digitalWrite(IN2_1, LOW);
152   analogWrite(ENA2, wheel_speed);
153   digitalWrite(IN1_2, HIGH);
154   digitalWrite(IN2_2, LOW);
155   analogWrite(ENA3, wheel_speed);
156   digitalWrite(IN1_3, HIGH);
157   digitalWrite(IN2_3, LOW);
158   analogWrite(ENA4, wheel_speed);
159   digitalWrite(IN1_4, HIGH);
160   digitalWrite(IN2_4, LOW);
161 }
162
163 void loop() {
164   distance1 = measureDistance(trigPin, echoPin1);
165   delay(wheel_speed);
166
167   distance2 = measureDistance(trigPin, echoPin2);
168   delay(wheel_speed);
169
170   distance3 = measureDistance(trigPin, echoPin3);
171   delay(wheel_speed);
172
173   if (distance1 == -1 || distance2 == -1 || distance3 ==
        -1) {
174     stopMoving();
175   } else if (distance2 < (detection_threshold * 0.8)) {
176     Serial.println("Stopping");
177     stopMoving();
178   } else if (distance1 < distance3) {
179     turnLeft();
180   } else if (distance3 < distance1) {
181     turnRight();
182   } else {
183     moveForward();
184   }
185
186   delay(100);
```

```
}
```

Listing 1: Arduino Code for Motor Control and Distance Measurement

Listing 2: Raw Experiment Data

	1			2			3			4		
Detection Threshold / cm	Collisions	time taken	Pass?	Collisions	time taken	Pass?	Collisions	time taken	Pass?	Collisions	time taken	Pass?
10	5	12.03	☑	7	19.35	☑	6	14.03	☑	7	16.61	☑
15	4	15.25	☑	4	14.25	☑	5	16.26	☑	3	9.97	☑
20	3	14.13	☑	3	16.19	☑	2	14.5	☑	2	15.65	☑
25	2	13.19	☑	3	20.19	☑	2	14.71	☑	2	16.12	☑
30	1	11.56	☑	2	13.37	☑	1	13.59	☑	0	9.75	☑
35	1	14.03	☑	1	12.09	☑	1	16.07	☑	2	17.50	☑
40	0	13.22	☑	0	11.00	☑	0	14.53	☑	0	11.37	☑
45	0	14.41	☑	0	13.6	☑	0	16.85	☑	0	20.15	☑
50	0	41.69	☑	0	39.07	☑	N/A	N/A	☐	0	32.31	☑

5			6			7		
Collisions	time taken	Pass?	Collisions	time taken	Pass?	Collisions	time taken	Pass?
6	12.22	☑	7	19.53	☑	8	17.16	☑
4	13.44	☑	5	18.10	☑	5	14.63	☑
5	18.97	☑	3	14.46	☑	3	15.90	☑
5	21.18	☑	1	15.05	☑	2	14.47	☑
1	12.88	☑	2	18.79	☑	1	13.00	☑
1	14.67	☑	0	11.91	☑	1	14.35	☑
0	9.82	☑	0	11.09	☑	0	10.94	☑
0	10.81	☑	0	24.88	☑	N/A	N/A	☐
N/A	N/A	☐	N/A	N/A	☐	N/A	N/A	☐