Project 2 - Autonomous Vehicle Project

Team 05

By Alix Laurent, Austin Roach, Andrew Becker and Zhuanghuan Yin

<u>**Memorandum:**</u>

To:	Lead Engineer
From:	Engineering Team 05
Date:	March 13th 2025
Subject:	Reflections on Autonomous Robot Project

This memo will provide an analysis on our autonomous vehicle project. We spent many hours designing, configuring, and creating the vehicle. We attached multiple parts, designed a map algorithm, and combined both the physical and virtual work together. We tested adding an ultrasonic sensor, compass, and even 3d printed an extra component for the robot. Through this design process, we encountered many mistakes and learned many lessons. In this memo, I will describe the vehicle, detail the coding algorithm, and provide feedback on what we learned.

Design

Our team's design incorporated multiple stages to produce an autonomous robot.

For the vehicle itself, we used a BBC Micro Bit attached to a battery on a 3d printed frame. This frame was screwed onto the top of the vehicle. Underneath lay a metallic frame that held the rest of the parts. A motor was screwed into the side of the frame, with two protruding ends that held rubber wheels. The vehicle also had a mount in the front that held the ultraviolet sensor. On the battery pack was a robot:bit expansion board, which connected the controlled power of the battery to the wheels.

Algorithm Type

As for the programming, we implemented a breadth first search (a.k.a. wavefront) algorithm in python. To create this, we tested the algorithm in python with multiple sample maps made of 2d lists. These maps provided a diverse group of test cases to ensure accuracy of the algorithm. The algorithm itself starts in *wavefront_search(map)* by creating a wave increasing by one value after viewing each possible step from the current point. This is why the algorithm is called 'wavefront," as it creates a wave starting from the original value. After building the map, the algorithm marks the most efficient path in *navigate_to_goal(map)* starting from the end to the beginning of the map. This ensures that the algorithm does not get stuck. The algorithm then calls a final method *go_to_goal(map)* which follows the path from start to finish and provides the robot information on the turns and movement required.

Algorithm Choice

Our choice of using the wavefront algorithm is based on designing the most efficient algorithm from the start to finish. We could have just used a more simple brute force algorithm and designed a pathway to the finish. While this would have made the coding more simple, it would not have provided us with the most accurate and quick path to the solution.

Performance

The performance of the robot overall worked with the maze algorithm, but due to motor inconsistencies and charging issues, the robot failed to reach the end point. Even though we rigorously tested the vehicle–multiple surfaces, wheel accuracy, optimal speed/stopping time, ultrasonic sensor, compass, and an enhanced 3D printed surface, the robot still failed to accurately turn in orthogonal directions. The vehicle turned too far to the left, which caused it to end 3 squared off.

Lessons Learned

Next time when working on the vehicle, we should get new motors. Ideally, the vehicle should be able to stop with both motors working in unity. This will provide straightness and a greater degree of accuracy for our vehicle to ensure we reach the end goal of the maze.

Regards, Engineering Team 05

Executive Summary

Problem Statement:

This project focuses on designing a navigation system for a small, affordable Autonomous Vehicle (AV) system on a grid system with a known set of obstacles. The system must determine and correctly guide the AV along the shortest path from a starting point to an endpoint without touching an obstacle. This project will be useful for real world application in situations with known obstacles such as cleaning up city parks.

The AV must:

- Have a user input a 8x11 (10 in by 10 in) grid with known obstacles and be able to move straight and turn right and left 90 degrees.
- Consistently determine the shortest path from start to end without hitting obstacles.
- Must compute and complete actions in a reasonable amount of time.

Unique Features of the Device:

Our AV navigation system utilizes a smart pathfinding algorithm to deliver a reliable low-cost pathfinding robot capable of solving a complex grid system with obstacles.

- Wavefront Algorithm – The wavefront algorithm creates a “wave” of steps from the start to the end, to create the shortest path from start to finish.

Performance Analysis:

Our AV navigation system successfully found the shortest path from the start grid to the end grid using the wavefront algorithm, achieving 100% accuracy in determining the correct path across the 8x11 grid. However, during our demonstration run, power issues and traction differences between the ground and tape caused the robot to drift severely to the left, resulting in the Av running into obstacles and it ended up being 3 grid cells off-target. The algorithm worked perfectly, but hardware challenges caused the AV to not successfully be in the correct square at the end of the demonstration run. We believe with better wheels and batteries, we could hit the target accurately next time.

Design Considerations:

The primary components of the Autonomous Vehicle Navigation System are an accurate and consistent movement system, with the addition of a robust navigation and pathfinding algorithm to find the shortest path. Both elements are necessary in order for the Autonomous Vehicle (AV) in determining the shortest path and how to navigate the maze reliably and efficiently. In this section, we will discuss the ways in which each component of the final design of our AV robot, including the drive system, calibration, movement code, compass feature, and pathfinding algorithm, contributes to the successful movement and implementation of the wavefront pathfinding algorithm of the AV.

Our first and biggest challenge in designing the AV was standardizing the movement of the robot across all trials. We were given two DC motors with significantly different resistances, which meant that it was difficult to have the robot drive in a straight line. In fact, the difference in the resistances was so great that it was making it impossible for the robot to drive perfectly straight, it was always wearing heavily on each side. As a result, we effectively had to replace one of the defective motors to try to fix this issue which was mostly successful, but the inaccuracy problem still remained although greatly decreased. We thought this was inevitably a problem with the motors and requested permission to change our motors once again but there were no more available unfortunately. Because our access to useful hardware tools, such as encoders, was limited by the provided motors, we were forced to try and account for the movement variance in the actual software. This would end up being not a perfect solution, we found that the robot would veer severely to one side when going “straight”, especially when crossing multiple grids at a time. We tried our best to account for this veering by having all motor movement governed by a variable ratio that provided more power to the right wheel and a little less to the left wheel. This ratio was easy to adjust during calibration, so we could increase it to pull the robot to the left, and decrease it to pull the robot to the right. However, when the motors turned off at the end of a movement, the right wheel would spin for longer than the left one, causing more drift. We attempted to account for this by gradually decelerating the AV instead of turning off the motors immediately. This reduced the jerk at the end of the movement and overall made our movements more consistent. However, these measures were not entirely sufficient, and we were plagued by reliability issues between trials. We intended this to be a temporary solution to the inaccuracy problem but this ended up becoming our permanent solution since we could not switch out the motors again, and because of the failure of our compass feature which I will touch on in another paragraph.This was by far the largest limiter on our performance during test runs, and further paragraphs will discuss further attempts to remedy the issue.

Another crucial challenge that we have encountered is about improving the movement code itself in the software of the microbit, which was in fact the most integral piece to the overall design of the system.We employed a series of commands and code that dictated the AV's speed, direction, and turning behavior. This code was designed to be flexible, allowing us to adjust for changes in the environment or the robot's performance. This aspect was critical in the debugging, wheel power ratio phase and the overall testing phase. By using code that can be easily activated and deactivated with a # symbol in front, we were able to test our code however we wanted which enabled us to tweak the wheel power ratio and tried to fix the problem where one motor was providing more power than the other, resulting in the robot veering to one side. We included provisions for both linear motion and rotational movement, trying to take into account factors such as the robot's turning radius and the desired speed at each phase of the grid movement. We wanted the robot to turn slowly so that it wouldn't overshoot because we found out through testing that the higher turning speed, the higher the inaccuracy of the 90 degree turns. As we already had a lot of errors with the inaccuracy of our motor, we wanted to limit this best we can by modifying the speed the robot operates at. However, the challenge of the inaccurate motors mismatch persisted even with all of this fine tuning we were still unable to get it working perfectly. To make matters worse, this technique, where we modified the power output of the wheels turning, was based on the actual power of the batteries which decreased in power output as we used them. As a result, these commands were often adapted on the fly, requiring continuous fine-tuning during in-class testing since the power of the batteries were being drained and the wheels were once again super inaccurate. Over time, the movement code became more and more refined, but it remained susceptible to clear errors when combined with the inconsistent hardware, especially our motors. Additionally, one of the newer problems that we find hard to break through is that even if we keep correcting the code to make it more and more refined, it still does not solve the whole problem, it is more of a band aid solution not an actual one. And at the same time, even with the same more precise code, there are still different cases of contradiction, sometimes the AV will be deflected to the left by a smaller angle, and sometimes it will be deflected to the right by a slightly larger angle. The AV does not provide a consistent output so we are left guessing what will be the output every time we make changes to the wheel power ratio.

Another significant challenge we faced while developing our program was deciding if we wanted to utilize the built in compass feature on the micro:bit in order to determine where the robot should point towards when going forwards and when turning. We wanted to utilize this feature in order to ensure a precise 90 degree turn each time our robot turned while the wavefront algorithm was being executed. We even thought about utilizing the compass feature to adjust for any discrepancies when the robot turned since we noticed that the robot would sometimes turn a couple degrees too far to the left or right which is not by itself horrible, but overtime this error would add up and our robot would inevitably not be in the correct spot at the end of our algorithm. However, despite trying to make it work for numerous days, we were unable to make the compass feature work successfully. The overall goal was to align the robot perfectly after each turn, matching the grid's layout in order to limit errors. We tried to code it to read the heading before and after a turn, then adjust motor power to correct any deviation but we found it difficult to use because the orientation of the compass would always change whenever the AV moved positions resulting in an even greater inaccuracy than before, without utilizing the compass. We found that the compass lagged by at least 15-30 degrees due to processing delays, and inconsistent motor response which likely amplified general error. We also noticed that the battery charge also threw off readings, as low power seemed to lag the compass by at least 30 degrees while in high power it lagged by around 20 degrees. We tried to play around with it and make it work but no matter what we did we could not get a reading that was accurate enough to use, so we ended asking for help to the TAs but we quickly found out that this compass feature would most likely not work as we needed it to, so we decided to scrap this feature and instead rely on the initial "wheels turning 90 degrees" code despite its inaccuracy, since the compass feature was even more inaccurate. Overall, we were unable to properly utilize the compass feature on the microbit and were forced to use the "wheels turning 90 degrees"code because we could not find a better solution to the accuracy problem. If we were to have more time on this project, I would like to revisit this feature with better and more accurate hardware because I think it could at the very least be used as an efficient calibration method and potentially successfully even used to correct the discrepancies in turning.

Results and Discussion:

Our AV performed with partial success. The vehicle turned and moved in the appropriate cartesian directions, but the motors failed to provide accurate turns. We ended up 3 squares away from the target. In this analysis, I will discuss the result, what went wrong, what we did, and what we could do in the future.

Algorithm Result

The algorithm succeeded. It was effectively able to traverse around the obstacles in its path to the solution. Our wavefront algorithm effectively scoped out the shortest path from the start to the finish and provided the appropriate instructions to travel. It would take the map given, with 0's representing open spaces, 1's representing boundaries, 2 representing the start, and 99 representing the end. The algorithm would shoot a 'wave' from the "2" integer position, creating an incrementing step for each open square. After the wave covers all squares, the algorithm calls another function to mark the pathway from start to finish. The robot then is given instruction on how to traverse the pathway, with move_forward() and turnRight() or turnLeft(). Our code worked perfectly, but the only downside was the sizing of the map. When we presented our vehicle, the map size was different from the one I originally used. And as the map dimensions in my code were immutable, we could not adjust the grid to account for the larger map size. Although with the smaller map, the robot was still able to map out the solution, it is something to make sure to account for in future projects, so the robot can traverse through larger mazes.

Hardware Issues

As for the vehicle hardware, our motors were inconsistent and inaccurate. Due to motor wear, charging differences, or some other reason, the motors would not turn straight. Sometimes on the same code, the orthogonal turns would be 95 degrees, but on other times, the turns would be around 85 degrees. This issue caused our AV to inaccurately turn left, and end up three squares away from the planned end point.

Tests and Trials

When we started programming, our initial assumption was the motor issue could be solved through solutions. We tested a compass, an ultrasonic sensor, and multiple variations in wheel speeds. However, all tests failed. The compass provided an inaccurate orientation each time we calibrated due to magnetic interference. The ultrasonic sensor would not help straighten the vehicle due to its imprecise nature. The individual wheel speeds were clearly different due to differences in friction. We implemented a ratio of right to left wheels to sync the motors. While this method slightly helped, it still did not completely ensure orthogonal movement.

Considerations

For the future, we could replace the motors. The BBC Micro:Bit is an old device, and the motors most likely have heavy wear. This could be the reason why we faced inconsistencies for the motors. Also, the motor that we used, the DC motor, has a short lifespan and has an open loop, meaning no correction for drift. The DC motor itself could explain our inaccurate turning and moving mechanisms, and could be the reason why we couldn't get accurate orthogonal turns. Instead, we could use an encoder, a motor attachment designed for precision. This motor tracks shaft rotation and provides position/speed feedback. For the future, we should consider changing the motor that we use to try and get more accurate turns.

Conclusions & Recommendations:

Conclusions

In conclusion, while our AV system demonstrated the functionality of its pathfinding algorithm, several hardware limitations prevented it from achieving its intended level of accuracy. The wavefront algorithm itself performed as expected, successfully identifying the shortest path from start to finish by avoiding obstacles. However, the inconsistencies in motor performance, including issues with motor misalignment and inadequate power distribution between the left and right wheels, significantly affected the vehicle's ability to complete the task accurately. This resulted in the vehicle drifting to the left and finishing 3 grid cells away from the target, despite the algorithm's success.

Our attempts to mitigate this issue by using a motor speed ratio, ultrasonic sensors, and a compass did not yield satisfactory results. The motor mismatch remained the primary cause of inaccuracy, as the mechanical system could not provide consistent straight-line movement or precise 90-degree turns. In addition, the compass feature failed to deliver reliable orientation data due to magnetic interference and processing delays, further complicating our calibration efforts. Ultimately, while the software (specifically the algorithm) proved effective in solving the problem, the hardware challenges prevented the vehicle from reliably executing the pathfinding instructions.

Recommendations:

The biggest improvement would be to replace the current motors with more precise ones. Motors with encoders would be ideal because they give feedback on how much the motor has moved, which would help make the vehicle move more accurately and reduce issues like veering off course.Also for the better Calibration.The vehicle's performance could improve with a better system for calibrating the motors. This means adjusting the motor power ratios and wheel speeds as needed during the robot's operation. We should also test the vehicle on different surfaces to make sure it works consistently in various environments.

Additionally, we still need to make the compass more clear. The compass didn't work as expected because it had delays and didn't give accurate readings. In future projects, we should try using other sensors, like gyroscopes or accelerometers, that might give better data. If we decide to use a compass again, we could combine it with other sensors to improve its accuracy.