# Product: ParkED
**Team: Back Benchers**

## Abstract

Our team is developing a mobile park bench which can move to and from the bench owner's desired locations. For this demo we have implemented, using our own custom built hardware and a software architecture, a prototype of the robot that can plan and execute a path between two locations provided by a user through a web interface whilst avoiding unknown obstacles.

## 1. Project management update

### 1.1. Demo 2 Goals:

- Navigational sensors included - achieved
- Global planner / path-finder implemented - achieved
- Obstacle avoidance implemented - partially achieved
- Communication between front-end and robot - achieved
- Market Research completed - partially achieved
- Ethical Research completed - not achieved
- Safety features included - not achieved

### 1.2. Deviations

Although we struggled to reach the goals we set for this demo, the hardest part, we feel, was the hardware and software architecture alterations. See Section 1.6. Our market research is only partially achieved, as we have a meeting lined up with a member of Edinburgh council to whom we have reached out for a discussion about the implications of our product in a real park setting. The meeting, which will take place on 10th March, we are sure will prove very informative. Due to the time taken up by architecture alterations, we didn't have time for ethical research, determining the necessary safety features, researching relevant weatherproofing, or locally avoiding collisions other than in simulation. However, we believe we will have time to complete these before demo 3.

### 1.3. Group and work organisation:

Group meetings were very important in the lead up to demo 2, especially as we planned to create the navigation stack ourselves. This required close collaboration to ensure that our complicated moving parts harmonized, as well as learning from each other the skills required to work within the ROS architecture.

As the demands of our workload changed, so too did our group structure. We organised ourselves into four subgroups, each of which was responsible for a component of functionality: Hardware, Global Planner, Local Planner, ROS. Once we had agreed on the format in which data would be passed between these components (using custom ROS Messages and Actions which were defined in collective group meetings), our subgroups were able to work asynchronously. This greatly reduced friction and allowed us to complete our work amongst our diverse and busy schedules. For these reasons, agreeing on a protocol for transmitting information through ROS early on was a great decision.

Version control software proved very important for us as our respective functional components began to work together. It was important that any update made in one version of a file would be compatible with another. For this we used git, storing repositories on github.com. This greatly increased our efficiency, especially when much of our work was completed remotely.

### 1.4. Work allocation:

#### 1.4.1. Global Planner

- Rory worked with Emily on the design and implementation of the path finding algorithm. Specifically programming the functionality that reads map data and processes it into format that is suitable for the A* algorithm, whilst adhering to the constraints of the domain. He also reached out to councils and park managers to gain marketing insight. **43 hours**
- Emily worked with Rory on the global planner by implementing the A* search algorithm, testing and debugging it. She also worked with Abdul to enable communication between the planner and the rest of the navigational stack. **32 hours**

#### 1.4.2. Local Planner

- Ziqian initially worked on the frontend and constructed a partially finished login authentication system. Then he moved onto the local planner part of the pathfinding algorithm and worked on the contingency planner. He also worked collaboratively on the simulation test for obstacle bypassing. **35 hours**
- Muiz (Leader) first did some research on security and implementations for the frontend, and then joined the the local planner team to fully implements and integrate the system with other components. He worked on the main flow of the robot movement in the local

planner, and did a lot of simulation to ensure that it works before trying it out on the actual robot. **70 hours**

- Suvi, after recovering from Covid and leaving isolation, spent some time catching up with the progress that the rest of the team had made on the project. Then they joined the local planner team, specifically writing code for the main flow of the local navigation algorithm. **30 hours**

#### 1.4.3. ROS

- Abdul: built upon the basic experience of ROS gained in IVR course and studied further to develop a complex architecture for the system's navigation stack. He then coordinated with all sub-teams to integrate their subsystems into the navigation and control stack. **20 hours**

#### 1.4.4. Hardware

- Jack (Leader), developed our own libraries and communication protocols, requiring extensive research. Much of the work the hardware team made on the project would directly affect the performance of the project in the demo, hence why so much time was spent on this part of the project.
  - 23hrs building libraries (Python)
  - 10hrs developing the GPS system (OpenCV)
  - 21hrs building and repairing robot
  - 15hrs flashing firmware and burning boot-loaders
  - 18hrs hardware communication protocols

  **Total 87 hours**

- Abdul: Assisted Jack in many of the aforementioned tasks, sharing the difficulty in research on protocols necessary to support the development of hardware systems. He also spent significant amounts of time developing and fine tuning the GPS system.
  - 10hrs on GPS system (OpenCV)
  - 7hrs flashing firmware and burning boot-loaders
  - 9hrs coding the Arduino
  - 18hrs hardware communication protocols

  **54 hours**

- Youwei worked on the 3D modelling of the robot. He designed and 3D printed the wheels, as well as sourced the tracks and added wheel sleeves. He also designed, modelled and laser cut the outer casing, back rest and arm rests, and created the enclosing boxes for both robot models. **40 hours**

#### 1.4.5. Application

- Emily spent a large amount of time working on the front-end application. She created multiple web-pages for the website, and spent most of her time on the user account page which interacts with ROS. She implemented a live map using mapbox, which shows the park and all of the contained benches, updating their positions in real-time by subscribing to a ROS topic. She enabled sending instructions from the frontend to ROS, telling a specific bench which location to move to. **33hrs** - total **65hrs**

Abdul + 5 hours in all sub teams - **Total: 79 Hours**

### 1.5. Budget Allocation

- Wooden sheet for box, straight gear, armrest and backrest (£9.5)
- Laser cutter technician time for the enclosing box of the robot, armrest and backrest (£2 @ £15 per hour)
- Plastic for 3D printing of 6 track wheels and 6 hollow cylinders (£4)
- 3D printing technician time for 6 track wheels and 6 hollow cylinders (£5)

  **Total: £ 20.50**

### 1.6. Plan Modification

ROS navigation stacks rely on LiDAR sensor and odometry data. These don't meet the domain which we operate in (Zang et al., 2019). Modifying the existing stack to meet our needs seemed, based on online research and discussions with Thomas (expert), to be more difficult than designing and developing our own. See Section 4.2.

A lot of our original architecture and hardware was redesigned. For local obstacle detection, we swapped out LiDAR which costs roughly £250, for an ultrasonic sensor which is £5. A similar functionality in a different domain can still be achieved at a much cheaper price (Azeta et al., 2019). We originally hoped the Raspberry Pi would be sufficient to communicate with the new hardware, but after discovering bugs in the hardware implementation of clock stretching, we had to introduce an intermediate Arduino to translate. See Section 4.1.

## 2. Quantitative analysis and testing

**Global Planner** For the global planner, which finds a global path through the premapped area, we have designed a test suite using python's `unittest` library. As the system is large with many moving parts, it is important that the code fails in predictable and readable ways if provided with invalid input. Tests like `test_reject_out_of_bounds` and `reject_if_end_in_obstacle` assert this, ensuring that an appropriate error message is given and the path returns 'None'. This None value is interpreted by the node that calls the global planner, which sends the message concisely to the user.

It is difficult to directly test for correctness in a pathfinding algorithm, as finding the optimal path by hand is very

| TEST | SUCCESS |
|---|---|
| NODES IN PATH NEVER CROSS GRAPH CONSTRAINT | √ |
| PATH RETURNS 'NONE' IF START NODE IS ON THE BOUNDARY | √ |
| ALL NODES IN PATH ARE WITHIN BOUNDARY | √ |
| ALL CONSECUTIVE NODES IN PATH DON'T INTERSECT BOUNDARY | √ |
| PATH RETURNS 'NONE' IF GIVEN END NODE CLOSE TO OBSTACLE | √ |
| PATH RETURNS 'NONE' IF GIVEN START NODE CLOSE TO BOUNDARY | √ |
| PATH IS FOUND IN UNDER 10 MILLISECONDS | √ |
| STRAIGHT LINE MOTION IN GAZEBO | √ |
| OBSTACLE BYPASSING IN GAZEBO | √ |

*Table 1.* Results for unit tests run on the global planner software.

resource intensive and error prone. Instead, to test this functionality, we have ensured that the path provided was valid under all input conditions including many edge cases (e.g. constraints such as empty list and various start and end positions). We have defined a valid path as a path that does not enter the buffer zone, the nodes of the path are all within the boundary, and the path never crosses a constraint that it has found. The outcome of these tests can be found in Table 1.

Although the speed of the software is not of great importance, `path_found_in_under_10_milliseconds` was chosen to ensure that there is not a gross misuse of resources. For example, there could be a mistake in the software causing a very long loop. The result of this test assures us that that is not the case. See Section 4.3

**ROS Architecture** First integrating all components such as the global and local planner into ROS, we tested each of our components' communication through ROS, for example by checking that the correct message is published to the correct topic. Then, we tested if the messages subscribed to and received by any components are successfully received, and parsed correctly in all components. For every ROS Action, Service Server and Client, we also wrote dummy clients to thoroughly test if the servers output intended results.

**Local Planner** For the local planner which creates the kinematic trajectory for the robot, we decided to use Gazebo, which is a robot simulation that is able to rapidly test algorithms. We created a world with free spaces and obstacles, which allowed us to test the local planner functions without having to be provided with the global plan. We gave the robot a route which replaced the global planner input, and tested both the straight line and obstacle encountering movements. After spending some time debugging the code, we met our requirements. Even though we soon realised that in real world the measurements would not be as accurate, this is a decently effective approach in assuring the reliability of the pathfinding and obstacle avoidance algorithm.

**Hardware Tests** Prior to integration, testing the hardware required files with scripted movements and function calls to check for expected values or outputs to motors. With some perseverance, most devices were functioning correctly. Still, the motors proved to be the most difficult, as there were many low level processes requiring debugging, and at times outwith our control such as the Raspberry Pi's 3 B+ faulty hardware implementation of clock stretching. While the motors are still a major factor, other integration tasks could continue with some compensation for their intermittence.

**GPS simulation** We optimized our simulator for detecting **red, blue, yellow** and **green**, then did tests to ensure that for each of these colour settings we are able to fully detect the an object of the said colour, and no object of any other visibly different colour. We then defined a function to map dimensions in image pixels of the SDP demo area to centimeters, and tested our simulator by detecting an object of a particular colour at different positions in the demo area. These measurements were then mapped from image pixels to centimeters, ensuring that we get linear and consistent mapping after distortion correction.

## 3. Budget

- RTK-GPS transceiver (£93-157)
- Casing (£234 - £410 | Metal and Designation fee)
- Integrated Motor Encoder Board (£50)
- Onboard computer (£30-50)
- Battery (£50-£200)
- Track wheels (£40-£150 | 4 Wheels)
- Track links (£400-700)

**£987-£1832 in total**

## 4. Miscellaneous

*Following feedback from demo 1, we have made significant changes to how we implement our solution.*

### 4.1. Hardware

The Wild Thumper 6WD with suspension demanded that we create our own motorboard firmware and libraries to control it. Burning an Ardunio bootloader to the at-Mega328T Chip on-board allowed us to flash I2C control .ino files onto the motor board. We then flashed an Arduino to interface with I2C, which meant we could avoid the Raspberry Pi I2C clock stretching bug. The created Python library allowed us to connect over TTL serial to the interfacing Arduino and send commands directly to the motor-board. However, because serial does not preserve any of the original motor-board clock frequencies, the RasPi can become out of sync.

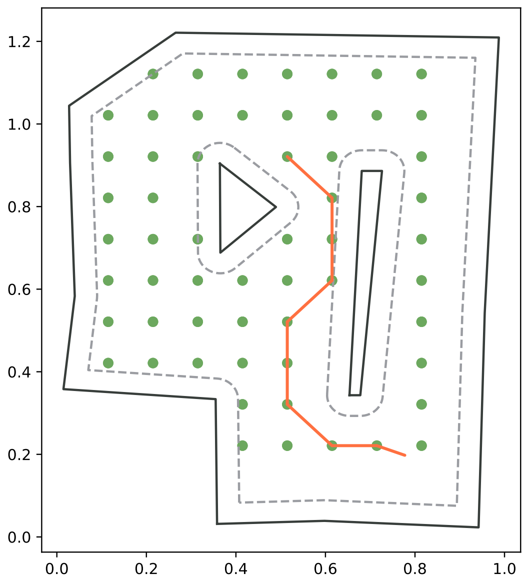

*Figure 1.* Visualisation of our path finding algorithm given no constraints on the graph. Notice the buffer area around obstacles to prevent bumping into them

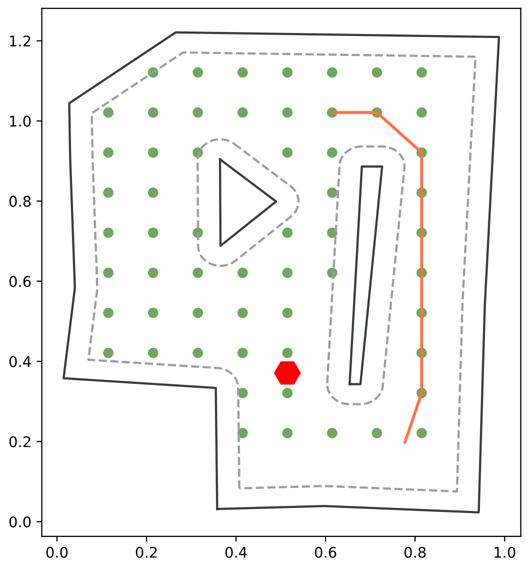

*Figure 2.* Visualisation of our path finding algorithm which, having received an unknown object, constrains the graph and finds the next quickest route.

We hope to later implement a Arduino specific PID controller to interface in the event of these stutters, for which no library is currently available. Using our unconstrained hardware frees us from many of the Turtlebot limitations, and makes for a more domain-appropriate prototype that reflects our final product. Another remarkable implication of this decision is that it reduces the cost of our prototype by £1000.

### 4.2. ROS Navigation Stack

As a team, we are implementing a navigation stack from scratch. This, allows us to make the greatest use of our team's expertise and will provide results for the prototype which are more adaptable and appropriate to our domain. (Gill, 2018) (Hansen, 2019) (2, 2020)

### 4.3. Navigation

The global planner receives the starting position and a goal position of the bench from the frontend and, according to a representation of the park, calculates the most efficient path through the space. It sends this list of coordinates to the local planner, see Figure 1. If the local planner detects an object blocking the path that it cannot move around, the global planner is called again, this time with a constraint which represents this obstacle. The global planner, if called with a constraint, modifies its graphical representation of the space and returns, as before, a list of coordinates representing the new most efficient path to the goal, see Figure 2.

The graph representing the space is generated automatically from a geojson map file, where nodes are generated in a grid of valid locations in the space, and edges between nodes are only created if there is a valid straight line path from the location of one node to another (Zeyad Abd Algfoor, 2015).

We use the A* search algorithm as which finds the optimal path between two nodes (Hart et al., 1968). The A* algorithm requires that its heuristic function (the function that estimates the potential value of a node) always underestimates the true cost of a path. This is known as an admissible heuristic. In our graph, each node object represents a location in the space and contains its spatial coordinates as an attribute. The straight line distance between the current node and the goal node is therefore very easy to calculate. This makes for a great heuristic function as it approximates the length of a path but never overestimates it, as there will never be a route to the goal node that costs less than a straight line.

## A. Appendix

## References

2, Ros. Navigation concepts, 2020. URL https://navigation.ros.org/concepts/index.html.

Azeta, Joseph, Bolu, Christian, Hinvi, Daniel, and Abioye, Abiodun. Obstacle detection using ultrasonic sensor for a mobile robot. *IOP Conference Series: Materials Science and Engineering*, 707:012012, 12 2019. doi: 10.1088/1757-899X/707/1/012012.

Gill, Jasprit S. Setup and configuration of the navigation stack on a robot, 2018. URL http://wiki.ros.org/navigation/Tutorials/RobotSetup.

Hansen, Matt. Navigation2 overview, 2019. URL https://roscon.ros.org/2019/talks/roscon2019_navigation2_overview_final.pdf.

Hart, Peter E., Nilsson, Nils J., and Raphael, Bertram. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136.

Zang, Shizhe, Ding, Ming, Smith, David, Tyler, Paul, Rakotoarivelo, Thierry, and Kaafar, Mohamed Ali. The impact of adverse weather conditions on autonomous vehicles: Examining how rain, snow, fog, and hail affect the performance of a self-driving car. *IEEE Vehicular Technology Magazine*, PP:1–1, 03 2019. doi: 10.1109/MVT.2019.2892497.

Zeyad Abd Algfoor, Mohd Shahrizal Sunar, Hoshang Kolivand. A comprehensive study on pathfinding techniques for robotics and video games. *International Journal of Computer Games Technology*, 2015. URL https://doi.org/10.1155/2015/736138.