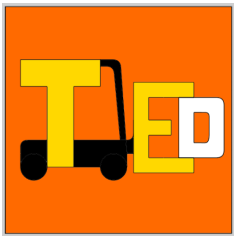
Product: TransportED
Team: AutomatEd



Abstract

TransportED is a mobile robot whose main functionality will be to pick up boxes from shelves and to bring them to a target location while coordinating with other robots to avoid congestion and collisions.

Since the last demo, we developed our localisation mechanism, finalized the robotic arm, imported the robot design into Webots and integrated the components to finish the robot. Furthermore, we set up the central server that enables us to communicate with the robot. We also developed a way to generate simulations efficiently using only little configuration, which allows us to test our robot on many different maps and will help our customers to test our system for their specific warehouse layout. This includes early-stage development of a web interface the customers can use to set their warehouse up. We integrated robot movement that allows us to move from node to node and adapted our database design.

1. Project plan update

- Robot in simulation - Achieved
- Initial map generation - Achieved
- Final robotic arm control - Achieved
- Localisation mechanism - Achieved
- Navigation - Partially achieved
- Simulation working with single robot - Partially achieved

1.1. Deviations from original plan

By this demo, we wanted to have a working simulation with a single robot. We were able to achieve almost all tasks to achieve this goal, but still need to integrate the routing algorithm that creates the path needed to get from one node to any other. This was caused due to difficulties we faced when implementing the line-following and localisation mechanism, but are sure that we will complete it by the 15th of March. We took on additional challenges, such as the map generation. This decision was made as it is vital for our customers to easily see that our solution will work in their warehouse, and fitted in nicely as it allowed us to get more work done in parallel.

1.2. Group organisation

In the modelling and simulation subteam, Rohan worked on constructing a scissor lift mechanism for the robot in the simulation, as research indicated this was more challenging than expected. Furthermore, he created a Shelf proto node that easily allows us to change the dimensions and the number of shelves to allow for more thorough testing. Mike imported the robot design into Webots. Jiaqing together with Divy successfully integrated the line-follower with the robot and used the localisation method to route the robot from one node to another. Divy developed the localisation mechanism, which is based on RFID tags. Furthermore, he worked on the navigation mechanism and made sure it works with the central server. Fredrik decided on the final robotic arm, integrated it with the robot and worked on the central server that communicates and coordinates with the robot. He worked together with Dave, integrating Ryan's communication protocol to establish a connection between the central server and the robot.

In the software team, Reece created a program that can easily generate a warehouse map according to specific parameters. Ryan worked on developing a web interface that will be used by a user to draw paths between nodes to allow traversal. Finally, Ufuk worked on the user interface, creating and communicating with the database and connecting frontend to backend.

Each week, we had an all-hands meeting in which everyone had a few minutes to talk about their progress since our last meeting and what they will work on until the next meeting. This ensured that everyone is up-to-date with the latest changes in development. It led to people asking clarifying questions about the work done by others, which resulted in them setting up meetings when needed to make parts of the code work together.

Most tasks we worked on are from our original project plan. After each milestone, the team leaders came up with additional tasks to ensure everybody has something to work on. During the meetings, we often found additional challenges we had to solve in order to make our subcomponents work together, finding more tasks in the process.

Additionally, whenever somebody made some major contribution such as getting the warehouse generator to work, that person posted a small video showing the functionality he had achieved. This boosted morale and motivated others to do the same.

1.3. Current budget spent

At this point in time we have not incurred any costs. So far, we have spent 15 minutes of our technician time with Gary to talk about the hardware feasibility of our robot.

1.4. Modifications to future milestones

By the next demo, we want to achieve a multi-agent simulation and strongly believe that we will be able to reach this milestone by the 15th of March. Furthermore, we want to improve on our web interface and make it look nicer, as even though the functional requirements are met, there needs to be more work done in terms of design and usability. Additionally, we will work on the multi-robot coordination algorithm that makes sure robots do not collide with each other. If we have time, we will also attempt to allow for multiple boxes on the same platform, increasing the throughput of our system. We will also start collecting information that can be used for the website for the industry day.

2. Technical details

2.1. Hardware

2.1.1. Robot Design

The initial design for the robot included a scissor lift which raises and lowers the platform to retrieve items. However, experimentation inside Webots showed that connecting two *Solids* together that are both independently driven by separate motors - as is required in a normal scissor lift - is extremely troublesome. We ended up on the following simplification, which does not compromise physical accuracy: rather than hinging the scissors in the midpoints of the rods as is done in standard scissor lifts, they are hinged at their endpoints. This operates very similar to a standard scissor lift, much like the kind we would use if building in-person. The platform can be raised up to 0.8m (with only two scissors and a small (0.4m) test scissor length), and naturally more height can be achieved with the addition of more and longer scissors. The lift still needs to be integrated with the main robot. We were unsatisfied with the visual appeal of the initial robot design and changed the robot base to be the one of the KUKA youBot.

2.1.2. Localisation method

We settled on RFID tags as our primary localisation mechanism. They are placed at each intersection and have a unique ID that allows us to identify where the robots are at any point in time. The RFID tags are currently configured to have a transmission range of 15cm, perfectly in line with what is reasonable for that technology. We also thought about using NFC tags, but with a transmission range of usually only 4cm and costs roughly five times higher per tag (0.24£ vs 0.05£) there was no real benefit. We also looked into using QR codes, but came across problems when they were not in the center of the image and therefore abandoned them. During the last Q&A session it was brought up that our lines would wear out after significant use. We do not see an inherent issue with that, as one could simply apply another layer of paint whenever necessary whenever necessary, which is cheap and easy to do. The wear of the RFID tags are also not a huge problem, as our robots turn exactly above the tags, never touching them with their wheels. For harsher environments with high humidity or humans possibly stepping on the tags we will protect the tag by putting them into a clear plastic casing.

2.1.3. Line-following

There are three infrared sensors placed at the front of the robot, each sending five beams in an aperture of 1.2 radians. Using these, we can infer the position relative to the line. This allows precise movement, which further compliments our choice of Mecanum wheels.

In case of an unforeseen circumstance, where the robot gets off the line there is a safety mechanism in place. In this, it looks for a line in a small area and follows it until it reaches the node. This is then communicated with the server, and determine the location.

We added friction to our floors, and a noise parameter to our Infrared. This is to ensure our system works with different floor/ warehouse environments. (INF)

2.1.4. Robotic arm

We abandoned the idea of using an IRB industrial robotic arm, as they are too heavy to fit on our platform. The IRB 1200 – one of the smallest robotic arms – has a weight of 25 kg, which too heavy to lift with our platform.

We now settled on the robotic arm of the Kuka youBot (YOU), as it is small and fits on the platform of the robot without taking up too much space. It is relatively compact at a size of only 65.45 cm and lightweight, weighing only 5.8 kg. It has an reach of 54 cm and can hardly reach something that is not on our platform as the platform is 50 cm wide. Therefore, we also use a linear joint to get the robotic arm to the other end of the platform. When the parcel is connected to the suction cup, we retract the linear joint again to get the robotic arm back to its original position and get the parcel onto the platform.

2.1.5. Fixing ikpy

As mentioned in our previous report, we are using ikpy (IKP) to control our robotic arm. When we added the linear joint into our configuration, we noticed that the joint configurations returned by the inverse kinematics function of ikpy returned incorrect results. After some debugging, we found out that ikpy is currently lacking support for prismatic joints. We added this functionality and are currently waiting until it gets merged into the main branch of the repository.

2.2. User interface

2.2.1. Web Interface

The web interface allows the user to see the current tasks directly in the main page. There is information about the package that is carried, which robot is carrying that package and whether the robot is on its way to the package, or it already picked the package up. The user can use the navigation bar at the top to go to 'Requests Packages' and 'Warehouse Generator' pages. Package requesting page is used to pick the needed packages from the list. Once the packages are requested, the backend is run asynchronously to get the them. The requested packages are no longer visible to the user to prevent them from requesting the same one again. 'Warehouse Generator' page contains the warehouse generator that will be mentioned in 2.3.3. If the user runs into any problems, there is an admin page for debugging, which can see and edit everything in the database. This page will be used by the team that is sent by our company.

2.3. Software

2.3.1. Central server

We set up the base code that dictates which tasks our robot has to execute in order to deliver a certain functionality, such as moving a parcel. This can be scaled easily to handle the swarm functionality. When somebody wants to move

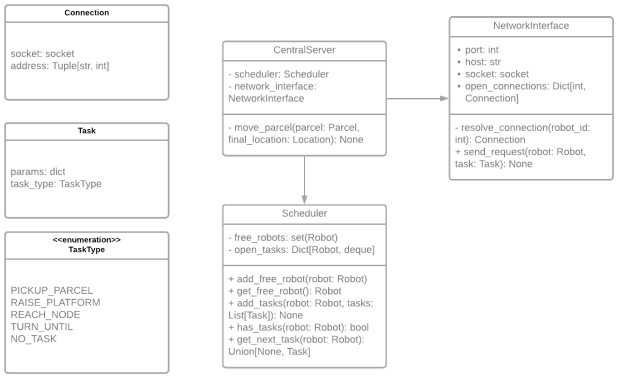


Figure 1. Class diagram on the server side. This does not include all classes, only the ones exhibiting major functionalities currently

a parcel from one location to another, we pick a free robot from the scheduler, tell the scheduler to add the necessary tasks to the robot, and then send each task one by one to the robot until all tasks have been achieved.

2.3.2. Networking

Using the networking protocol developed in the first demo, we have integrated this to work with a robot in Webots. This is done by creating a network interface and a central robot server to which the robot connects to. Each robot has a thread that listens to incoming tasks and puts them into a task queue. Then processes them one by one until completion. When a task is completed, it sends the task type back to the server to signal completion. The server then sends the next task to be done. It makes use of the asyncio library to send tasks to robots concurrently.

2.3.3. Warehouse Generator

Warehouse generation can be broken into 3 main stages: user input, file generation and database population. The user can define a warehouse environment via the web interface we have designed. This takes information such as nodes, types and connections along with other values for generation, i.e. node coordinates, and exports the grid in the form of a JSON string to be received and parsed server side into a format that can generate a valid Webots map file. It also stores all the relevant navigation information for the map in the database to allow A* algorithm to function correctly. A grid sample that the user can produce is displayed in figure 2.

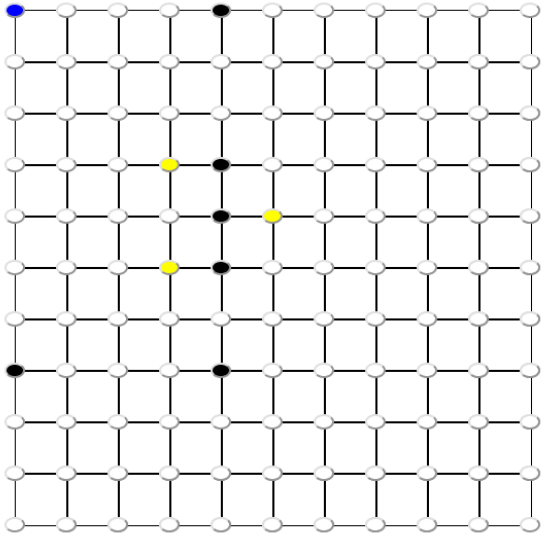


Figure 2. A graph representation of an arbitrary warehouse configuration, where the blue node is the robot start position, a yellow node is a shelf and a black node is a robot junction.

Currently, the system successfully handles creation and parsing, but we currently need to manually import the data from the website as there still needs to be some integration done. However, this code should be implemented in the back end and the environment will be auto-generated when the user uses the web interface to plot their warehouse.

The utility of this generator is to allow the user to get a visual representation of how the warehouse would be in prior. It can allow managers to test new efficient configurations or even allow them to test scalability in a consequence-free environment. This reassures potential customers of the robustness of our product.

Furthermore, it will help us save time during testing as we

can quickly create and populate different warehouse layouts as opposed to creating these by hand.

2.3.4. Navigation

Navigation is done by giving the robot commands from the central server (i.e. turn until two lines have been crossed), the robot will then perform that task and when it is done will send a request to the server for it's next task.

The robot knows when it has finished a 'go straight' command as at each junction there is an RFID tag that once scanned by the robot will lead the robot to request its next task from the server.

3. Evaluation

3.0.1. Warehouse Generator

The warehouse generator was first evaluated by creating an image for a warehouse layout we wanted to represent and see if the map generated matched this image (i.e. the image shown in figure 3). The image was then used to create the input arguments for the generator function in the same format that will be generated when it is implemented in the back end in the future.

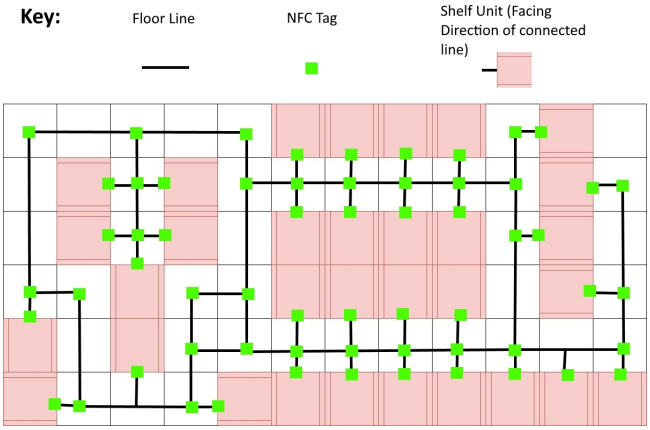


Figure 3. Image created of warehouse that we wanted the code to generate

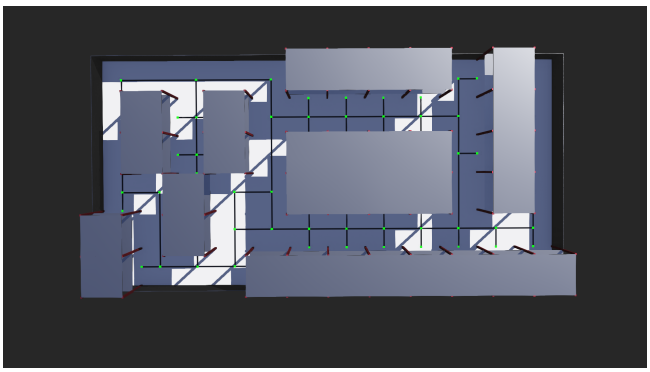


Figure 4. Screenshot of the code generated map being ran in webots

Figure 4 shows the Webots world file that was generated by the warehouse generator. When comparing this to the image created of the desired output, it matches as expected; however, certain tags are not visible in the image due to the 3D nature.

Test Number	Floor Size(meters)	Line Width	Line Errors	World Errors
1	12x6	2	0	0
2	12x6	1	1	0
3	12x6	3	1	0
4	9x5	2	0	1
5	9x5	1	1	1
6	9x4	2	0	0
7	9x4	1	1	0
8	1.5x1.5	2	0	1
9	1.5x1.5	1	1	1

Table 1. Errors observed in each warehouse generation (Shelf size was "1x1x1" for every test)

When given an odd line width, every line file generated had some lines being one pixel off from their neighbour. This is expected to be because of the rotation of the grid images leading to the differences

When given at least one odd value for the floor size, it was noticed that the RFID tags were 0.01 meters off of where they should be. As this is such a low value, it is unlikely to cause issues to the solution; however, it is still worth being noted.

3.0.2. Parcel pickup

We tested how long it takes us to pick up parcels at different shelf heights. We measured the time taken to pick up parcels at 1m, 1.5m and 1.75m. For each height, we ran the code six times to get a good estimate of the performance of our parcel pickup. Each run was a success, but there were some errors.

Height (m)	Average time (s)	Standard deviation	Errors
1	9.159	0.367	0
1.5	9.26	0.194	1
1.75	13.05	0.107	6

Table 2. Average time and number of errors for parcel pickup at different heights. Six test runs on every height

At the height of 1.5m, the weight of the box caused the robot base to slightly lift up from the ground, which often did not cause any problems, but once lead to it colliding with the shelf. At the height of 1.75m though, this problem intensified, and the platform collided with the shelf four out of six times. Although the collision would cause the shelf to be displaced, the robot was still able to pick up the parcel. We will have a closer look at how to prevent such issues and will consider methods such as adding counterweights or adding constraints to the maximum payload we can carry at a particular height.

4. Budget

As we planned on developing our robot fully in simulation, we have not incurred any additional costs and might only be spending 10-20 pounds on hosting services.

5. Video

This is the link to the video. To check the time of the upload, click the three dots on the top in the right hand corner, and then click on details.

References

Inverse kinematics python library. URL https://github.com/Phylliade/ikpy.

Infrared sensor documentation. URL https://cyberbotics.com/doc/reference/distancesensor.

Kuka youbot reference manual. URL https://www.generationrobots.com/img/Kuka-YouBot-Technical-Specs.pdf.