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.
Until now, we have set up a basic user interface that enables one to add items to the database, as well as tracking the location of different shelves. We also set up a basic message exchange system so we can send controls to our robot remotely. Furthermore, we have developed an initial path-finding algorithm the robot can use to navigate from its current position to a shelf. We are already able to control a robotic arm, enabling us to pick up packages using the suction cup. Additionally, we designed a warehouse in Blender and are almost finished with the initial robot design, which includes the scissor lift mechanism in Solidworks.

1. Project plan update

- Locomotion - Not achieved
- Proper testing and debugging - Partly achieved
- Initial item database - Partly achieved
- Pickup mechanism - Achieved
- Chassis - Achieved
- Platform - Achieved
- Warehouse layout - Achieved

1.1. Deviations from original plan

Originally, our main focus for milestone one and two were purely focused on getting the simulation and initial robot design ready. We quickly found out that this was unrealistic, as many people had no experience with Webots or modelling software like Solidworks and Blender. Therefore, we adjusted our initial milestone to include software-related goals such as getting a database and a connection to it up and running. We pushed back the goal of the full robot design to the 19th of February. This allows Mike and Rohan, who have experience in modelling, to focus on the robot design while Fredrik and Jiaqing focus on the integration of it in Webots.

Even though we made a lot of progress, there is still a lot left that needs to be done, such as enabling our algorithm to work with multiple robots

1.2. Group organisation

In the modelling and simulation subteam, Rohan designed the shelves and warehouse in Blender, while Mike was responsible for designing the initial robot in Solidworks, as they have both much experience with the respective software. Furthermore, Jiaqing and Fredrik familiarized themselves with Webots. Jiaqing developed a lane following mechanism, which enables the robot to follow a line. Fredrik developed some code to control the robotic arm, pick up a box and get it to another location.
In the software team, Ufuk set up a basic web interface that enables admins to access, edit and add shelves and items. Ryan worked on the networking protocol that allows the central server to send commands to the robots. Reece developed the A-Star algorithm we will use for navigation, as well as a class for database access using SQL. Dave developed the initial database design, which still needs to be integrated fully into Ufuk's website.

During the week, we had two meetings: one with the subteam only and one with the whole team. In each meeting, we talked about the progress since our last meeting and made plans on what we will work on until the next meeting. Furthermore, whenever a task was finished, we marked it as "finished" in Asana which enables us to easily track what still needs to be done. We are using GitHub as our version control system. Each feature, e.g. the robotic arm control, had a specific branch associated with it which was merged onto the main branch. Before it was merged, one of the team leaders or subteam leaders reviewed the code and suggested potential changes and improvements.

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

As the goals for our milestones until the first demo were mostly related to hardware, the software team managed to pull a few tasks forward in order to supplement the next few weeks of development. For example, we already have a simple web interface which allows for simple management of our warehouse as well as a protocol that allows us to connect the robot with a central server. As the design of the robot turned out to be more difficult than expected, we were not able to fully design it before this demo, but are already finished with the scissor lift and chassis in Solidworks. We plan on having our robot design fully working and integrated in Webots until Milestone 3, which is on the 19th of February.

We still think that we are able to reach all of our milestones by the deadlines we set for them, as the software team has already completed some tasks associated with the upcoming milestones while the hardware team has enough leeway as the robot design is the only major goal of them.

2. Technical details

2.1. Hardware

2.1.1. Robot design

The initial design of the robot has been created with the scissor lift as the centre of the machine. The chassis surrounds the scissor lift with space below for batteries, vacuum pump, navigation electronics and equipment needed to communicate with the server. The scissor lift has a base of 700m x 500mm which is enough room for a robotic arm and the items. The lift can also reach up to 1.5m above the top of the chassis in its current state. However, this can be adjusted with changes to the link length and number of links. Mecanum wheels will be used for maneuvering the robot around the warehouse. These allow for easy control over the position and orientation of the robot in all directions just by changing the rotation direction of the wheels.

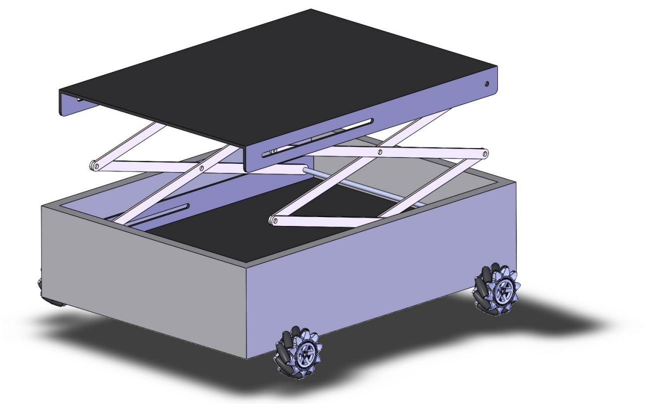


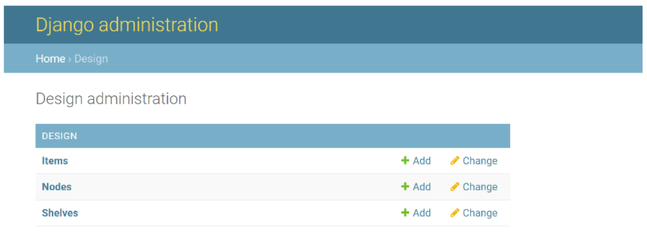
Figure 1. Initial Robot Design

2.1.2. Robotic arm

As we have not yet fully settled on the robotic arm we want to use, we are currently experimenting with the control of the IRB 2400 (IB2) industrial robot. It has a reach of 1.55 meters and would therefore be able to pick up boxes even from far away. The issue with it is that it is also huge with a size of 1.45 meters and is therefore not viable on a small robot, especially considering that it has to be lifted up with the platform. We are looking into trying out smaller robotic arms such as the IRB 1300 (IB1), as it only has a size of about 1 meter while still having sufficient reach of about 90cm. We were so far unable to test the new arm, as we could not find an accompanying URDF file, which is used to describe the elements of a robot and is needed to load the robotic arm into Webots.

2.2. User interface

We built a web application to easily interact with the database. Admins can access and edit nodes, shelves and items. A guest user can only edit and add items. The design of this application is not polished and its purpose was to provide a simple way of interacting with the database for testing and developing purposes, which will be revised and polished at a later date.



Django administration

Home › Design

Design administration

DESIGN		
Items	+ Add	Change
Nodes	+ Add	Change
Shelves	+ Add	Change

Figure 2. Basic user interface

2.3. Software

2.3.1. Robotic arm control

For the robotic arm control, we are making use of the ikpy (IKP) library. It enables us to quickly calculate the joint configurations needed to have the end-effector (EEF) at a specific position. We decided to use it as it has a simple interface. Furthermore, developing our own inverse kinematics solver (IKS) would be complicated, as we have limited knowledge in that area. Other IK libraries such as trac_ik, or Orocos need ROS (ROS) as backend, which we currently do not plan on using, as it would add an additional layer of complexity. As we currently do not have a method of localizing a parcel, we use the ground truth location of it. We feed the relative location of the parcel to our robot together with our current joint angles into ikpy and receive the desired joint configurations to reach the target with our end-effector. Then, we update the motors with the new joint angles. If we are close enough to the parcel to pick it up, we simulate the suction cup mechanism using the Connector (CON) node in Webots, and lift it then to a predefined location.

2.3.2. Networking

We developed the networking protocol between the robots and the central server using the socket (SOC) library in Python to communicate via TCP (TCP). Although UDP (UDP) is the faster protocol, we decided to use TCP instead as it guarantees the transmission of Packets (PAC). This is important as the robots are working in an environment in which safety is the highest priority. Minimal packet loss is vital to ensure predictable and controllable functionality of the robots. This cannot be guaranteed if packets are lost and fail to be re-transmitted, introducing more room for error and uncertainty.

TCP allows an exchange of messages between the server and the robot without depending on physical access to the drones and will allow management of multiple robots and their functions all from one central location. This also allows seamless integration of updates to their software and makes it much easier to distribute this across all robots as the number of drones increase.

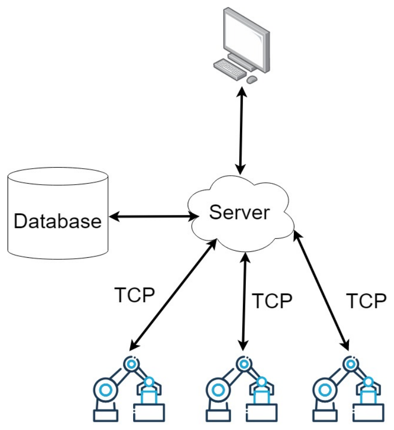


Figure 3. Network Diagram

To initiate a command, a server-side function will send a request to the robot client in the format "msglen:robotid:function:args", where msglen is a 5 character representation of the length (e.g. 00010 would tell the receiver to wait for 10 characters before interpreting the message). This message triggers the corresponding function on the robot with the specified parameters. A response will be sent once the function is executed to keep the server updated of the robot's status.

2.3.3. Navigation

We created an initial design of the robot's navigation systems through the implementation of an A* path-finding algorithm (A*) that is able to find the shortest path from a specified starting node to a specified end node.

We decided to use A* over other algorithms such as Djikstra as it employs a heuristic function that allows it to explore less nodes in order to reach the target node.

The A* algorithm implementation takes the following four arguments; "node_positions", "node_connections", "start_node_index", "end_node_index". "node_positions" and "node_connections" are both 2D lists with the first index of both defining the node number with the second index of "node_positions" storing the x,y position of the node and the second index of "node_connections" storing what nodes the current node is connected to. "start_node_index" and "end_node_index" are used to define the start and end point of the path search respectively.

2.3.4. Database

A database was designed to store information on the robot, shelves, package information and positioning. This is used to plan the routes that a robot will take, which item the robot is required to collect, which shelf it is on and its location on the shelf. The robot will use this data to plan its trip and will update any information on its status via the server.

Before deciding on a relational database, we considered using a graph database. While this could be beneficial for the constant evolving real-time data, none of the members in the software team has any experience with this. All members have prior knowledge with relational databases and three out of four have taken the introduction to databases course, which focuses on relational databases with PostgreSQL. Therefore, we collectively agreed on using relational databases to focus more on the other software aspects.

The entity-relationship (ER) diagram can be seen in fig 3. Note that this is an initial database design and may change as the project progresses.

There are 5 entities: Robot, Task, Package, Shelf and Node. A robot can be assigned one and only one task at a time. This task involves the picking of one and only one package. In a shelf, there can be zero or many packages. The position of a shelf and robot is identified by a node. The Robot table may be redundant for now, as we currently assume that we only have one robot in our environment. However, it will be of use when we start implementing the swarm aspect of our system.

One shelf has x number of layers/units, indicated by the attribute *num_of_layers*. Since we currently assume that one shelf unit can store only one package, the *layer_number* attribute in Package table will tell the robot which compartment the package is in. Together with *layer_height*, the robot can estimate the location to pick up the package. The robot has a *state* attribute, which will indicate whether the robot is picking up the package, dropping off the package at destination, or unavailable due to charging.

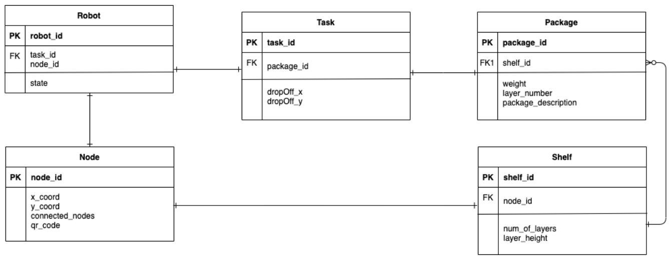


Figure 4. Database Schema

3. Evaluation

3.0.1. Navigation

The A* algorithm was initially tested on a 5x5 grid with the starting node of six and end point of 23.

The graph shown in figure 5 shows the algorithm output as well as the path taken as red lines on the graph. The output matches the expected one, verifying the algorithm is working correctly in this case

To further test the algorithm, we ran 10 different tests on the graph above to see if the expected cost matches the returned cost.

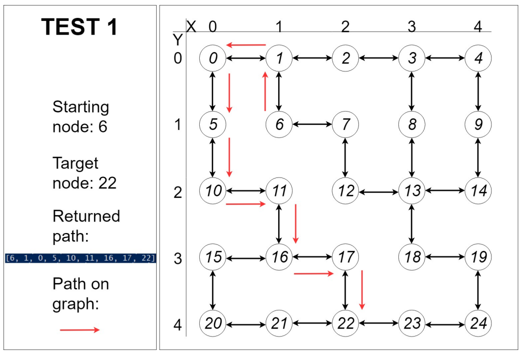


Figure 5. Test 2 results

Test Number	Starting Node	Target Node	Expected Cost	Returned Cost
1	6	22	8	8
2	6	23	7	7
3	2	14	4	4
4	20	4	10	10
5	10	14	8	8
6	3	12	3	3
7	19	0	7	7
8	0	21	7	7
9	4	12	4	4
10	12	4	4	4

Table 1. Results for 10 tests on A* algorithm

3.0.2. Networking

As we don't have a functional robot to test the client-server interactions on, we tested the scripts that would be deployed locally on a development machine. One script was run to start the server and multiple client instances were successfully able to connect to the server simultaneously, simulating multiple robots. Due to local deployment, any quantitative testing metrics such as response time and packet loss would not be a great reflection of the system therefor we have decided to push this to a future report.

3.0.3. Robotic arm control

To test the robotic arm, we first had to configure the noise of our positional sensors. We looked into the accuracy of positional sensors according to the estimates given by this manufacturer of sensors, and adjusted our noise to be 0.02 degrees per joint. Furthermore, the resolution, which determines the granularity in which we can measure joint angles, was set to 0.0055 degrees. Those correspond to the values of one of the lower-end sensors.

The test we ran had the following structure: the parcel was positioned either 20 or 40 cm straight away from the end-effector, and was displaced along the perpendicular axis by either 0, 10, or 15 cm. We counted the test run as a success if the robotic arm was able to pick up the parcel and pull it to a predefined target location which is close to the robot. With this, we wanted to test whether the arm is still able to pick up a parcel even if it is not perfectly lined up in front of it.

Test	Distance along X	Distance along Y	Success
1	20cm	0cm	√
2	20cm	10cm	√
3	20cm	15cm	√
4	40cm	0cm	×
5	40cm	10cm	√
6	40cm	15cm	√

Table 2. Results for 6 tests of the robotic arm.

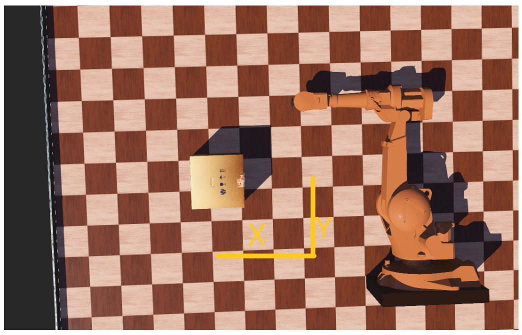


Figure 6. Test setup

The error resulted because the suction cup did not align with the orientation of the box, resulting in no pick up. Another thing noticed in the tests was that although the box ended up in the right position, the orientation of it was not always the same. The reason for those issues is that we currently do not have full control of the end-effector orientation. We will look into how to control it and will give an update on this in our next demo report.

4. Budget

As we planned on developing our robot fully in simulation, we do not think we will incur any additional costs to our budget. The only thing we might be spending money on is a hosting service for our central database, but the costs of this will be limited and are expected to be around 10 - 20 pounds. We plan on researching the potential costs of creating our system when we fully developed the robot design.

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

A* algorithm explanation. URL https://en.wikipedia.org/wiki/A*_search_algorithm.

Connector. URL https://cyberbotics.com/doc/reference/connector.

End effector. URL https://en.wikipedia.org/wiki/Robot_ end_effector.

Irb 1300. URL https://new.abb.com/products/robotics/industrial-robots/irb-2400.

Irb 2400. URL https://new.abb.com/products/robotics/industrial-robots/irb-2400.

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

Inverse kinematics. URL https://en.wikipedia.org/wiki/Inverse_kinematics#:~:text=In%20computer%20animation%20and%20robotics,the%20start%20of%20the%20chain.

Packets. URL https://en.wikipedia.org/wiki/Network_packet.

Robot operating system. URL https://www.ros.org/.

Socket, low-level networking interface. URL https://docs.python.org/3/library/socket.html.

Tcp. URL https://en.wikipedia.org/wiki/Transmission_Control_Protocol.

Udp. URL https://en.wikipedia.org/wiki/User_Datagram_Protocol.