**Knowledge Transfer Document**

**Social distancing robot**

**Fontys**

**Eindhoven**

|  |
| --- |
| **Project owner: Cees van Tilborg** |
| **Date: 24-09-2020** |

|  |
| --- |
| **Project team:** |
| **Yves Lardenoije, Atanas Marchev, Bram Odenthal, Hugo Vaessen, Nicky Verhees, Martijn Vissers** |

|  |
| --- |
|  |

Contents

[1 Introduction 3](#_Toc61516332)

[2 Background 3](#_Toc61516333)

[3 Initial problem 3](#_Toc61516334)

[3.1 Questions 3](#_Toc61516335)

[4 Research 4](#_Toc61516336)

[4.1 Robot navigation 4](#_Toc61516337)

[4.1.1 Hybrid A\* 4](#_Toc61516338)

[4.1.2 Jump Point Search 4](#_Toc61516339)

[4.1.3 D\* 4](#_Toc61516340)

[4.1.4 Conclusion 4](#_Toc61516341)

[5 Process 5](#_Toc61516342)

[5.1 Design 5](#_Toc61516343)

[5.1.1 Robot design result 5](#_Toc61516344)

[5.2 Implementation 6](#_Toc61516345)

[5.2.1 Talking Skeleton 6](#_Toc61516346)

[5.2.2 Robot Movement 6](#_Toc61516347)

[5.2.3 Robot Pathfinding 6](#_Toc61516348)

[5.2.4 Message protocol 6](#_Toc61516349)

[5.2.5 Order system 7](#_Toc61516350)

[5.2.6 Database design 9](#_Toc61516351)

[5.2.7 Gazebo 10](#_Toc61516352)

[6 Conclusion 10](#_Toc61516353)

[7 Recommendations 10](#_Toc61516354)

[8 Bibliography 11](#_Toc61516355)

# Introduction

This document is a compilation of all the knowledge the team has collected during their Embedded Systems Minor for the HTES Social Distancing Robot project. It will contain the information they have gathered while researching, designing, and Implementing a robot that will help waiters in bars and restaurants.

# Background

Since the outbreak of the COVID19 virus, people who meet many others are potential super infectants. Examples of such people are waiters in restaurants and terraces, or nurses who distribute meals in hospitals or nursing home. These people have tasks that can be automated by robots which will in turn lower infection rates and unnecessary contact will be avoided.

Robot solutions for social distancing are already on the market, but they are either expensive to purchase or provide limited in functionality. The professionally deployable solutions also require an elevated level of technological infrastructure and specialist personnel. (Teade Punter)

# Initial problem

The development of a low-budget robot for use within organizations for contact-rich distribution tasks, such as serving in a restaurant. The assignment is to develop the software, based on ROS, for this platform. The platform is: a 2-wheel (Segway/Oxboard) or 3-omniwheel robot.

The control of the robot should be simple enough for use by non-technically trained personnel. The robot will have to be able to work both indoors and outdoors, and to be able to bring orders to the customer's table. This also requires a form of stabilization mechanism, so that loose or liquid items also arrive properly. (Teade Punter)

## Questions

* How to detect a person and navigate to this person?
* What is a minimal technical configuration (sensors) needed, to enable a low-cost robot?
* How to have a good overview on the terrace to be able to pay attention to all attendees?
* How to map the sound on the (robot) map? The robot can register its own position, and to position this in its map. But what about the origin of the sound, and more important: how to map the sound on the map? Research is needed to find the actual source and map it on the robot map. Which data / signal type is required to add to the HERE tracker to enable accurate mapping?

(Teade Punter)

# Research

## Robot navigation

Here we have compiled a summary of the research we have done for the possible path planning algorithms we could use, and a final choice which was based on the complexity of the algorithm and how much it fit our project. For more details on every algorithm check out the Robot Navigation – Research document in the repository.

### Hybrid A\*

Hybrid A\* is a real-world algorithm that uses A\* as its base and is used for path planning for autonomous vehicles. This algorithm considers the maneuverability of the vehicle to plan out possible next movements. (Nordeus, 2015)

**Pros**

* Only algorithm that considers turning possibilities of robot
* Fast recalculation
* Possibilities for memory and CPU usage settings

**Cons**

* More complex than the other algorithms

### Jump Point Search

The JPS algorithm improves on the A\* algorithm by exploiting the regularity of the grid. The Jump Point Search has a general direction, and it scans multiple nodes in the same direction until it finds a “Jump point” where it starts a new scan with a new direction to the end goal. (Hofkamp, 2015)

**Pros**

* Fast
* Does not take up a lot of memory

**Cons**

* Recalculations of path might be slower than D\*

### D\*

 The D\* algorithm expands on the A\* by switching the end and start points and from there on out expand the open list until the start point has been found. (Stentz, 1995)

**Pros**

* Fast recalculations of path because it stores node directions

**Cons**

* Slower path calculations than JPS
* Takes more memory than JPS

### Conclusion

The Hybrid A\* was chosen because it is a real-world solution that is developed specifically for robot navigation. The robot is going to navigate in a dynamic environment in which it must be responsive and fast. Each change to this environment must be accounted for and checked if it would mean the robot’s path to alter. If that were needed the Hybrid A\* promises fast calculation for the next possible path.

# Process

The project that was given to us was in its first developmental stages. This meant that we had little to no previous information on it. We worked with another group for the design of the actual robot and had help of other students for matters that were not in our scope and domain of work. Our job was to research how to develop this robot and its accompanying functionalities – the order system and the back end.

## Design

At first, the group worked on designing the robot. This would eventually prove to be time wasted, since the mechatronics group was also working on a design. Our group was not aware of this because of poor communication of the project definition. When this became clear, our design was eventually scrapped, and we started to work with the mechatronics group to finish developing the robot together. The current design is the result of both the groups work.

This group also designed the product on the software side. For information on this, please refer to the git repository where our design files are stored.

### Robot design result

The current robot design is as seen in the image below:

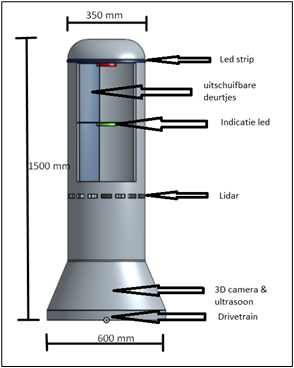


Figure 1: The current robot design

The most notable features of the robot are the lidar, the camera and the ultrasonic sensors. For the wheels, a differential wheeled base was chosen, which can move around its own axis. For more details on all the components, please refer to the “TotaalConcept.docx” file in the git repository.

## Implementation

For the implementation of the robot, we developed a talking skeleton, the robot’s movement functionalities and a pathfinding algorithm. For the ordering system we developed a protocol for the messaging between the clients, the bar, and the robot, a system for the infrastructure and a database design. For testing we build a simulation environment in Gazebo.

### Talking Skeleton

To prove the functionality of the communication between the robot, server, and order system we have developed the talking skeleton of the project. The talking skeleton features all the communication between the systems, without any other logic. All the messages defined in the message protocol (see *Message protocol* ) are implemented in the talking skeleton with the correct sequences.

The database is currently being simulated with a text file which the server maintains. For the product, this should be converted to a database as described in *Database design*.

### Robot Movement

When driving the robot around, the movement should happen smoothly. Accelerating, breaking, and steering should not be done abruptly to avoid the spilling of drinks and/or the robot tipping over. For this, an algorithm which limits the g-forces on the robot to an upper limit was developed. The algorithm also features an emergency brake method which does not avoid the spilling of drinks but does make sure the robot will not tip over.

The code of the robot movement needs to calculate the desired speed. This happens by using the distance to the endpoint to define when to start slowing down. The desired values for linear and angular speed are then each plugged into a discrete PID, this will smoothen the movement even more. After this is done it is checked whether the angle is reasonable to already start driving. This is done so that the robot can maneuver in tight spaces. Even if the linear speed is acceptable it can still be limited by the angle error. All off these different processes are controlled by variables in the code. There is also an emergency brake to make sure the sensors have a way to tell the robot to stop. Lastly, the movement is programmed in a ROS (noetic) node to make external communication easier.

For further information on the code please read the README in:

*$(git clone folder)/catkin\_ws/project/movement.*

### Robot Pathfinding

We chose the Hybrid A\* path planning algorithm because of the research we did. The path planner is a standalone program written in C++. It is controlled through a sample program that shows its capabilities by itself. When in deployment this program should run in cooperation with the other program in the robot’s navigation systems – the vision and the movement programs.

Before it is started the algorithm must have information about the world and the target location. The map should be a combination of a static map build specifically for the restaurant or bar and a dynamic map that is created from scanning the robot’s environment using its vision systems. After we have a target the program calculates a next best node and sends the information of the movement to the movement system. This movement will then displace the robot closer to the target.

### Message protocol

For communicating between the tables, the bar and the robot, sockets will be used for sending and receiving data. This can be either a simple TCP/IP socket or a WebSocket solution.

We do have to keep in mind that the order dashboard also needs to open a connection to the server when a customer wants to order, this means the picked solution for communicating should also be available for web projects (for example in JS), in case a web solution is chosen.

When placing an order, the active session of the customer will send the order specifications to the server. All communication data is converted to a byte array and based on its incoming header the server will handle it accordingly. After the order has been prepared by the staff, another message will be transmitted to the socket of the robot to instruct the robot to start handling the order(s).

The current project uses the default socket implementation.   
<https://man7.org/linux/man-pages/man2/socket.2.html>

#### Headers

Each received message contains an identifier (header) to specify what kind of data or action is received and the system will call the correct function to process it. The headers are defined in an Enum which is shared between the client(s) and server.

|  |  |  |  |
| --- | --- | --- | --- |
| *Header* | *Value* | *Description* | *Direction* |
| ***PLACEORDER*** | 0 | Used to create a new order | Staff à Server  Customer à Server |
| ***CANCELORDER*** | 1 | Used to delete an existing order | Staff à Server |
| ***STARTORDER*** | 2 | Used to let the robot start bringing certain order(s) | Staff à Server  Server à Robot |
| ***EDITORDER*** | 3 | Used to edit an existing order | Staff à Server |
| ***RETURN*** | 6 | Used to let the robot return to its starting position | Staff à Server  Server à Robot |
| ***STATUSUPDATE*** | 4 | Used to update the status of an order when the robot has an update available | Robot à Server |
| ***RESULT*** | 5 | Used to indicate how handling the previous header went | ALL |
| ***BILL*** | 7 | Used to request the bill from an order. | Client à Server |
| ***ID*** | 8 | Used to identify the client(s) on the server. | Robot à Server |
| ***GETOPENORDERS*** | 9 | Used to get the current open orders. | Client à Server |

*Table: Header definition*

Further information on the protocol, headers and their body content can be found in the communication specification document.

### Order system

To handle the orders of the customers at each table, we agreed on the usage of a website for showing and managing the orders.

The choice of a website gives us flexibility, we could redirect users to the menu or list of drinks using the (usually) already available QR code. These codes are (post COVID) commonly used for registering at the restaurant or bar the person is at.

We produced three ideas for the development of the system. They have not been chosen or developed upon, because of time constrains.

#### Order dashboard

In this design, only 1 component is required. The single component contains the database and the website. This means the bar personnel and customers both use this component for ordering and for managing orders at the bar.

The pro of this setup is that only a single component is necessary, which makes it easier to setup and use, this also means the price would be significantly less than solutions that use multiple components.

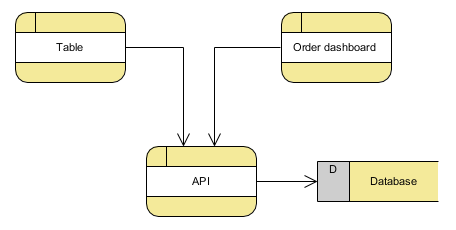
The downside is that in this scenario that it is not possible to react instantly to events that take place on the website, such as sending instructions to the robot or to instantly notify the bar that a new order has been placed.

#### Order dashboard (API)

This is slightly the same design as the first proposal (Order dashboard). But uses a central API for sending and retrieving information from the database. Information will be obtained from the API and new orders will be stored using the API endpoint. Using polling at a certain interval the dashboard for the bar personnel will be updated to list newly placed orders at the tables.

The pro of this environment is that the API can be accessed from any component (if we would like to), which gives us more scalability for the dashboard. The limitation is again reacting instantly to events, there will usually be some delay.

In figure 2 you see an example of this setup, please note that this solution can be a built as single component if necessary (The table also uses the order dashboard for placing an order, but on the customer side of the dashboard).



*Figure 1. Example of the API setup with the dashboard*

#### WebSockets

In this design web sockets are used so it is possible to have a two-way communication between a webserver and a client which works in real time.

As previously mentioned, the guest at tables will order their drinks by scanning a QR-code. When scanning the code their device will connect to the server as a client of a table (see Figure 1). When an order is received by the server, it will be sent to the bar right away. This means very little waiting time between receiving orders.

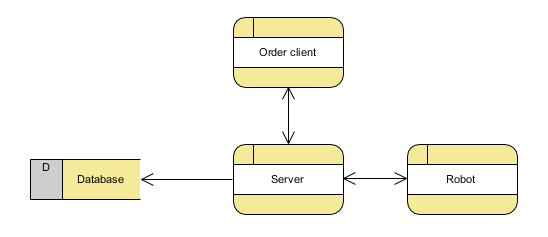
The server can send and retrieve data from the database whenever this is needed.

The guests will only have to connect to the server if they want to place an order. This means that it is possible to have multiple clients of the same table, which order drinks at the same time. The bar will then get all these orders separately.

A big advantage is that the clients or server push their data instead of polling for updates. This would make the server poll constantly for changes in which it cannot do other things.

A downside of web sockets is that it requires a reliable communication, which usually are not available in outside areas. There will need to be a retry/reconnection strategy that will always make sure an order is correctly send to the server. To prevent frustration, this must be done without cancelling an order and making the guest order again.

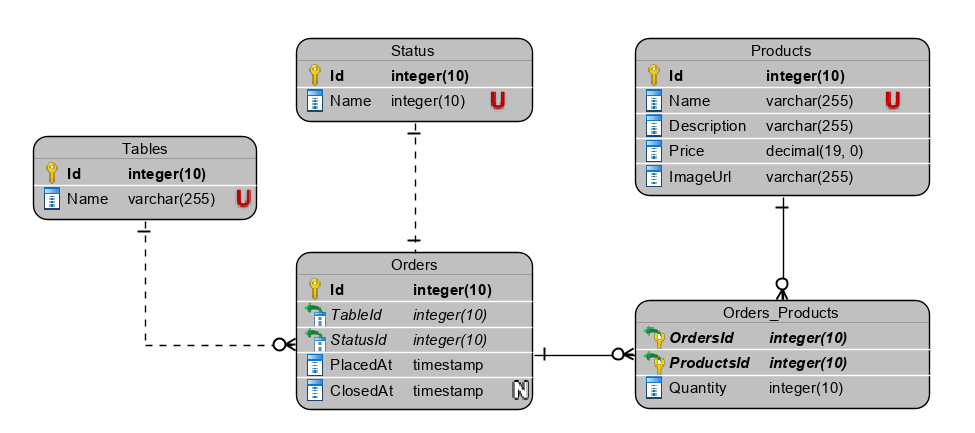
This solution requires the most development time.



*Figure 2. Example of WebSocket setup (the table is the client)*

### Database design

This is a brief description for the entity relationship diagram, this design is still subjective to change and is just to give an example of a potential (small) system for storing and managing orders.



*Figure 1. Simple ERD for storing order and product information.*

There are currently 4 entity tables and 1 pivot table.

When an order is created, the products are attached to the *Orders* record using the pivot table, which also contains the quantity of each product. The order will be attached to the correct *Tables* entity the order was created on.

The status of the order indicates what action(s) can be taken for the order, such as sending it to the robot if it is still open. Example statuses could be: ‘Created’, ‘Processing’, ‘Closed’ etc.

### Gazebo

# Recommendations

We would recommend future groups to:

* Keep close contact with other groups who are working on this project if there are any.
* Do frequent meetings so everyone is updated on his/her teammate’s progress and to avoid misunderstandings or double work.
* Use a project management tool to track and reflect on your progress.
* Have a list of priorities for the parts of the project and do the highest ones first.

# Bibliography

Belwariar, R. (2018, 9 7). *A\* Search Algorithm.* Retrieved from geeksforgeeks.org/: https://www.geeksforgeeks.org/a-search-algorithm/

Hofkamp, A. (2015, 10 31). *Jump Point Search: Fast A\* Pathfinding for Uniform Cost Grids.* Retrieved from Gamedev.net: https://www.gamedev.net/tutorials/programming/artificial-intelligence/jump-point-search-fast-a-pathfinding-for-uniform-cost-grids-r4220/

Navone, E. C. (2020, 9 28). *Dijkstra's Shortest Path Algorithm - A Detailed and Visual Introduction.* Retrieved from freecodecamp.org: https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/

Nordeus, E. (2015, 11 9). *Explaining the Hybrid A Star pathfinding algorithm for selfdriving cars.* Retrieved from habrador.com: https://blog.habrador.com/2015/11/explaining-hybrid-star-pathfinding.html

Stentz, A. (1995). *The Focussed D\* Algorithm for Real-Time Replanning.* Pittsburgh, Pennsylvania 15213: Carnegie Mellon University.

Teade Punter, J. D. (n.d.). *Student proftaak assignments HTES PII semester 7 Minor ES - 2020nj.* Retrieved from 2020nj HTES 4 – Social Distancing Robot serving a drink: https://git.fhict.nl/I397569/htes-social-distancing-robot/-/blob/master/Documentation/Sem7%20MinorES%20Assignments%202020nj%20-v1%20(3).pdf