**Robot Navigation Research**

**Social distancing robot.**

**Fontys**

**Eindhoven**

|  |
| --- |
| **Project owner: Cees van Tilborg** |
| **Date: 09-10-2020** |

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

Version

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Editor(s)** | **Edits** | **Status** |
| V0.1 | 05-11 | Atanas Marchev | Create document, start writing results of research | For review |
| V0.2 | 13-11 | Atanas Marchev | Fix feedback points | In progress |
|  |  |  |  |  |
|  |  |  |  |  |

## Distribution

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **To** |
|  |  |  |
|  |  |  |

Table of Contents

[Distribution 2](#_Toc56163697)

[Introduction 4](#_Toc56163698)

[Problem Definition 4](#_Toc56163699)

[Research Methods 4](#_Toc56163700)

[Main Research Question 4](#_Toc56163701)

[Sub-questions 4](#_Toc56163702)

[Research Results 4](#_Toc56163703)

[Dijkstra’s Algorithm 4](#_Toc56163704)

[Pros 5](#_Toc56163705)

[Cons 5](#_Toc56163706)

[A\* 5](#_Toc56163707)

[Pros 5](#_Toc56163708)

[Cons 5](#_Toc56163709)

[Hybrid A\* 5](#_Toc56163710)

[Pros 5](#_Toc56163711)

[Cons 5](#_Toc56163712)

[Maze Solving Algorithms 6](#_Toc56163713)

[Pros 6](#_Toc56163714)

[Cons 6](#_Toc56163715)

[Jump Point Search 6](#_Toc56163716)

[Pros 6](#_Toc56163717)

[Cons 6](#_Toc56163718)

[D\* 6](#_Toc56163719)

[Pros 6](#_Toc56163720)

[Cons 6](#_Toc56163721)

[Conclusion 6](#_Toc56163722)

[References 7](#_Toc56163723)

# Introduction

This document describes a short research that compiles a list of possible pathfinding algorithms and chooses which one would be best for the robot that the group is building for their Embedded Systems Minor.

# Problem Definition

The goal of the project is to design an autonomous robot server for a restaurant or a bar. This robot will have to navigate around the venue carrying drinks, food or going to pick up points so it can receive its next order. This means that it will have to safely deliver the orders to the clients without much delay and problems for the other clients. The platform design is created from the group and the actual robot will be received from another group that builds robots. The specific requirements for the robot pathfinding will be:

* Fast delivery to clients
* If problems occur the recalculation of the path should not take much time

# Research Methods

The research techniques that were used are:

* Problem analysis
* Explore user requirements
* Community Research
* Literature study

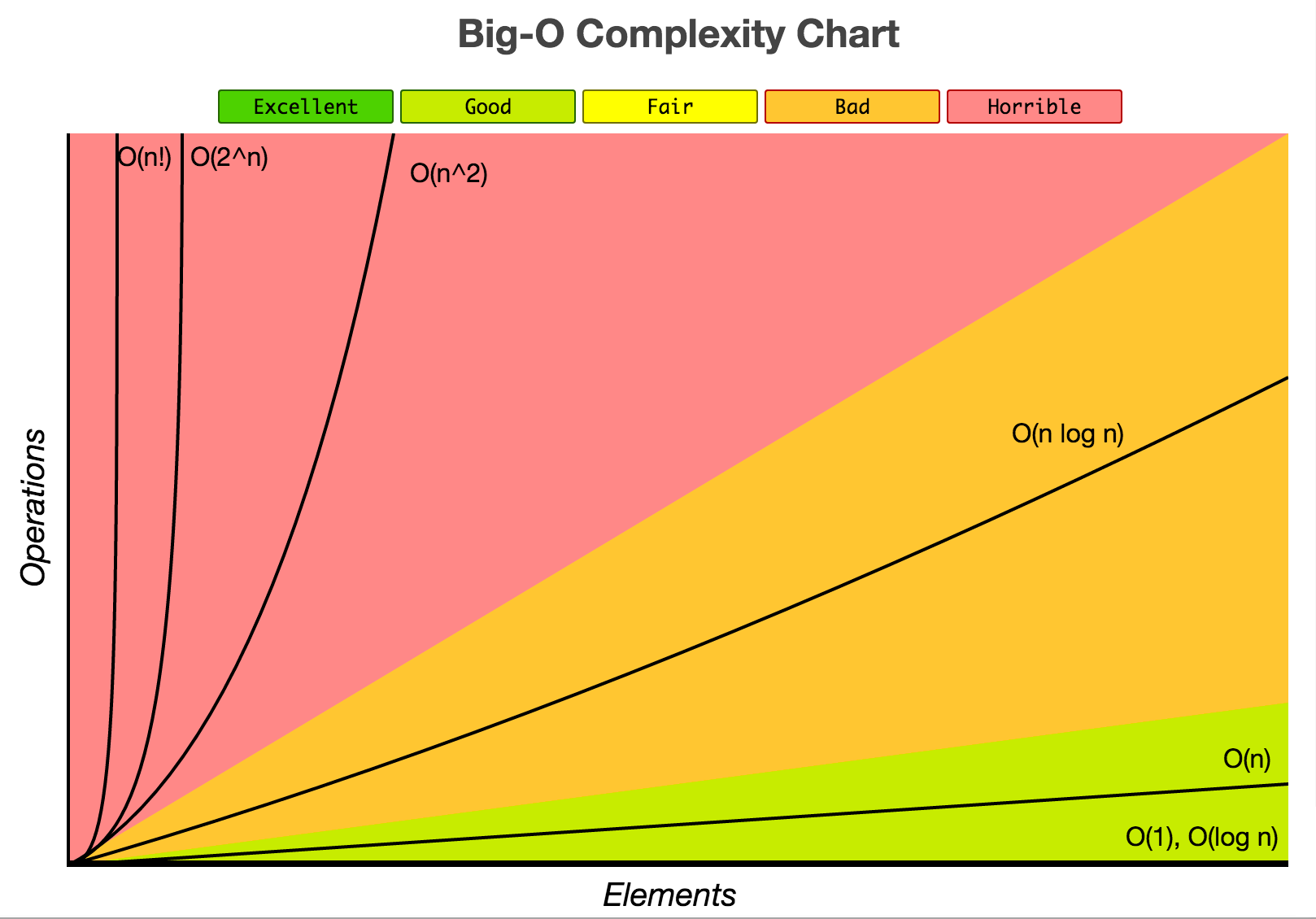
# Main Research Question

What is the best pathfinding algorithm for a robot server in a dynamic environment such as a bar?

## Sub-questions

* What pathfinding algorithms there are?
* What pros and cons do the algorithms have?

# Research Results

There are many algorithms today but most of them follow one base algorithm and derive from it. We will use the Big O notation to classify the space and processing power used for the size of the problem for each of the algorithms. 

## Dijkstra’s Algorithm

Dijkstra’s Algorithm is used to find the shortest path between nodes in a graph. The algorithm is slow, but sure to work. It starts at a Begin point and checks all its neighbors for the next shortest path, keeping a record of the sum of the length of the paths, and a shortest path so far. It keeps checking neighbors until it reaches the destination. After the algorithm has reached the destination it finds the shortest path from the end to the beginning, by backtracking its way to the start node, and chooses it. Dijkstra’s Algorithm has Horrible complexity (Θ(E+V^2)).

### Pros

* Reliable
* Thorough

### Cons

* Slow processing time
* Takes up a lot of memory

## A\*

A\* is an improvement to Dijkstra’s Algorithm. It Is the most common and successful algorithm. It works by creating trees(paths) originating from the start and extending them until it has reached the destination. In the beginning the A\* adds all possible nodes to a list that is called an open list, it then iterates through the nodes and adds all their neighbors to the open list. At the end of each iteration, it chooses which node to extend to, adds it to a list called the closed list, to shorten the path to the goal by guessing the remaining cost to it. The end of the algorithm is when the goal node is in the closed list. It can be used but many of its derivatives build upon it and make it better in certain cases which we will investigate and pick. A\* algorithm’s complexity’s is horrible (Θ(b^d))

### Pros

* Reliable
* Thorough
* Faster than Dijkstra’s

### Cons

* Slow processing time
* Lots of memory usage

## Hybrid A\*

Hybrid A\* is a more real-world algorithm that uses A\* as its base but has more complexity when it comes to the movement through the grid. It is used for path planning for autonomous vehicles. This algorithm works with a model of the vehicle and considers its maneuverability by using its turning capabilities to plan out possible next movements. The complexity from hybrid A\* is the same as the complexity of normal A\* + the complexity of vehicle dynamics.

The time complexity of vehicle dynamics is not in our scope to define.

### 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

## Maze Solving Algorithms

There are algorithms specifically for solving mazes like Wall Follower or Trémaux's algorithm but since our project is not about that, because they would be either too slow or impossible to use for a dynamic environment like the one we would have, they will not be investigated for the research.

### Pros

* Good for solving mazes

### Cons

* Not applicable for our situation

## Jump Point Search

The Jump Point Search uses the same logic A\* uses but instead of trying every node’s parents it 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. The JPS uses the open/closed lists less often and thus reducing the time taken to store and delete information from the memory. The time complexity of Jump Point Search is the same as A\* except when driving in straight lines, in this case it collapses a list of nodes into one node.

### Pros

* Very 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. After that it gets the minimum length from the expansion points it creates during its search and uses it as the path. Each expansion point knows the length from it to the target and when it searches it adds it with its next expansion points. The time complexity of D\* is equal to that of A\* for the first iteration, however since it remembers information about the nodes, recalculations are faster.

### Pros

* Fast recalculations of path because it stores node directions

### Cons

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

# Conclusion

We have reviewed some of the pathfinding techniques that are used in today’s robotics and gaming industries. Some of them were not what our robot was needed for and were skipped.

Using the Hybrid A\* would be ideal in most situations 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 because it does not iterate through every node to find the goal.

# References

Jaffar Ali Raza, S. (n.d.). Real-World Modeling of a Pathfinding Robot Using Robot Operating System (ROS). Retrieved November 09, 2020, from <https://arxiv.org/ftp/arxiv/papers/1802/1802.10138.pdf>

Jump Point Search: Fast A\* Pathfinding for Uniform Cost Grids. (n.d.). Retrieved November 09, 2020, from <https://www.gamedev.net/tutorials/programming/artificial-intelligence/jump-point-search-fast-a-pathfinding-for-uniform-cost-grids-r4220/>

Stentz, A. (n.d.). The Focused D\* Algorithm for Real-Time Replanning. Retrieved November 09, 2020, from <http://robotics.caltech.edu/~jwb/courses/ME132/handouts/Dstar_ijcai95.pdf>

Think Labyrinth: Maze Algorithms. (2015, November 20). Astrolog.

<http://www.astrolog.org/labyrnth/algrithm.htm#solve>

Navone, E. C. (2020, November 19). Dijkstra’s Shortest Path Algorithm - A Detailed and Visual Introduction. FreeCodeCamp.Org. <https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/>

Explaining the Hybrid A Star pathfinding algorithm for self-driving cars. (2015, November 9). Habrador. <https://blog.habrador.com/2015/11/explaining-hybrid-star-pathfinding.html>

Ryder, A. (2018, April 1). Dijkstra’s algorithm: Application, Complexity, Implementation, Explanation. OpenGenus IQ: Learn Computer Science. <https://iq.opengenus.org/dijkstras-algorithm-finding-shortest-path-between-all-nodes/#complexity>