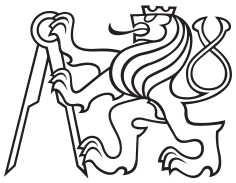
Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Trajectory planning for a heterogeneous team in an automated warehouse

Tomáš Rybecký

Supervisor: RNDr. Miroslav Kulich Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Cybernetics and Robotics
May 2020

Acknowledgements

I would really like to thank my supervisor RNDr. Miroslav Kulich Ph.D. for more than two years of inspirational leadership and mentoring. A large part of this work was developed within the Safelog project, which also gave me the opportunity to test the developed Fleet Management System with real industrial AGVs and gain valuable experience from working in the applied-robotics environment. Finally, I would like to acknowledge the co-authors of the Fleet Management System and its components, mainly Ing. Lukáš Bertl and Ing. Jakub Hvězda Ph.D..

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date 18. May 2020

.....
signature

Abstract

This thesis develops the topic of route planning for a group of cooperating robots, specifically in the automated warehouse environment, which has its specifics in the field of multi-agent path-finding. The thesis discusses possible modifications to existing planning algorithms based on these particularities, such as replanning based on the knowledge of existing plans. Furthermore, the thesis describes the development and operation of a management system for controlling a fleet of automated guided vehicles (AGVs) that operate an automated warehouse, which is also extended with the possibility of safely introducing a human worker. The planning algorithms and the management system were implemented in C++. Their functionality is then discussed, including the comparison with other methods.

Keywords: route planning, cooperative robots, automated warehouse

Supervisor: RNDr. Miroslav Kulich Ph.D.
CIIRC,
Jugoslávských partyzánů 1580/3
Praha 6

Abstrakt

Tato diplomová práce rozvíjí téma plánování trajektorií pro skupinu kooperujících robotů, a to konkrétně v prostředí automatizovaného skladu, které má pro multiagentní plánování svá specifika. Práce se zabývá možnými modifikacemi plánovacích algoritmů na základě těchto specifik, jako například potřeby lokálních úprav již existujících plánů. Dále pak popisuje vývoj a fungování systému pro řízení flotily robotů v automatizovaném skladu, a to včetně zahrnutí lidského pracovníka bezpečně se pohybujícího mezi roboty. Plánovací algoritmy i kontrolní systém byly implementovány v jazyce C++. Jejich funkčnost je na závěr diskutována včetně srovnání s jinými metodami.

Klíčová slova: plánování trajektorií, kooperace robotů, automatizovaný sklad

Překlad názvu: Plánování trajektorie pro heterogenní tým v automatizovaném skladu

Contents

Figures

Chapter 1

Introduction

The field of multi-agent path-finding (MAPF) is highly covered in many publications, and the solutions to the general problem are already on high levels of effectiveness. However, specific applications introduce different obstacles to deal with, but also opportunities to improve the performance of the path-finding process.

An application this thesis aims at are automated warehouses, where the usage of mobile robots[1] proves to be beneficial, as described in Chapter 5. A specific motivation leading to this topic then comes from the EU Horizon 2020 project Safelog[2], which presents a novel solution for human-robot cooperation within automated warehouses and will be further described.

In my bachelor thesis, I showed that sequential assigning of tasks for the agents introduces new constraints, but can be extended with techniques to improve the results, meaning the ability to find a solution or improve its quality [1]. Methods presented in the bachelor thesis were also published in a conference paper [3]. For convenience, only the bachelor thesis will be referenced regarding this topic.

To provide a wider background of the topic, Chapter 2 introduces the MAPF in detail and presents several state of the art methods, along with the

[1] The naming convention in this thesis uses the term mobile robots (or specifically AGVs, Autonomous Ground Vehicles) for the embodied robotic vehicles. Agents, on the other hand, is a term used for the planned virtual entities seen by a planning algorithm. These can represent both robots or humans.

[2] For more information about the project, see http://safelog-project.eu/

approaches from [1], and, finally, algorithms that serve as an inspiration for the problem aimed at later in this work. That is the possibility of distinguishing the agent priorities, as it proves to be beneficial to enable a human worker to enter the otherwise restricted area without the need to stop the work altogether.

This problem can be generally formulated as a higher priority agent appearing in the sequence of assignments coming to the route planner. While the higher priority agent could be planned first and the others afterward from scratch, Chapter 3 describes ways to simplify this process. The result of this re-usage of old planning knowledge promises a significant planing-time save-up. Furthermore, the need to produce new plans for the agents enables another possibility to apply the improvement heuristics described in Chapter 2 without any cost. Along with them, more new sorting mechanisms are proposed and compared.

The proposed algorithms were experimentally evaluated, and the results are presented and discussed in Chapter 4.

A significant part of this thesis describes the development of an actual management system for the mentioned automated warehouse that enables the presence of humans. The Fleet Management System (FMS) that is described in detail in Chapter 5 was developed within the Safelog project and incorporates task allocation, route planning, communication with the AGVs, and several safety and control mechanisms. It is also designed to cooperate with other Safelog components to operate the automated warehouse. It is important to mention that significant parts of the FMS, namely the raw layout implementation, the graphical user interface, and planner structure, were created by several other people. The system is presented as a whole, but the main contributions of the author were in reworking the system and its components to function without errors and in extending it with the human workers.

Chapter 6 then describes several tools developed together with the FMS to test and prove its functionality. The problem of a higher priority agent, namely human, in an automated warehouse is introduced in Chapter 7. The specifics of this application are described, as well as the necessary safety precautions. The chapter also describes methods to ensure and possibly anticipate human behavior, since violations of the given instructions should be expected. The research cooperation with the University of Zagreb (UNIZG) also led to a successful publication of a conference paper [4].

Performed experiments and their results are presented and discussed in Chapter 8. It also discusses the application of the FMS in a real warehouse demonstrator. Chapter 9, finally, wraps up the developed algorithms, as well as the management system. Possible future improvements and research interests are also discussed.

Chapter 2

State of the art

Planning of trajectories for multiple agents moving in a shared area is a widely studied field with several branches of solver approaches. The optimal solvers for the multi-agent path-finding problem (MAPF) can be classified as follows [5]:

- A*-based
- Increasing Cost Tree Search (ICTS)
- Conflict-Based Search (CBS)
- constraint programming

The A*-based algorithms use the A* algorithm to search a constrained state space for each agent. The state space is extended with the time domain in order to achieve collision-free plans[1] for the agents. The A*-based algorithms are addressed in detail in Section 2.1, since this thesis builds mainly on one of them.

The ICTS algorithms decouple MAPF into finding the cost of each agent and then retrieving a solution based on the costs. In the first part, they attempt to find an optimal solution for every single agent and retrieve its size - the cost. This vector of costs is taken as a root for an Increasing Cost Tree

[1] In this work, the term plan is used for the timed trajectory of an agent. The term path is used for the trajectory itself without the time domain.

(ICT). Its children then add 1 to each element in the vector, and the same applies to their children. The second part then uses a breadth-first search to go through the ICT. For each of the nodes representing a vector of costs, it attempts to find a valid (= collision-free) solution which fits the expected costs. That means, it searches for individual plans with the given costs and evaluates the validity of the full set of plans. By searching in a breadth-first manner, the optimality of the first found solution is verified. Some speedup extensions of the ICTS then prune the ICT by quickly identifying individual plans with no valid solution [6].

The CBS approach is very similar to ICTS. It also solves multiple single-agent path-finding problems, but with the addition of constraints that ensure their completeness and optimality. The process is again divided into two levels, and a binary tree is constructed. Within that, constraints (blocked states in the state space) are added on each level for agents to resolve collisions. Then a breadth-first search is applied on the tree to find the least constrained collision-free setup and retrieve its solution [7].

The MAPF problem can also be solved by a general-purpose solver that is given a set of constraints, ensuring a collision-free solution. For this approach, the MAPF problem is modeled as either a Constraints Satisfaction Problem (CSP) or a Constraint Optimization Problem (COP). The main benefit of these methods is that current general-purpose constraints solvers are very efficient and constantly further improved [5]. A beneficial property is also in the ability to choose the optimization domain, e.g., *makespan* (see Section 2.2).

2.1 A*-based algorithms

The A* algorithm was originally used to solve the general problem in discrete time. Such an approach leads to a large state space with a high branching factor. In the worst case for a graph with $|V|$ vertices and $|E|$ edges, the size can be $|V|^k$, where k is the number of agents, and the branching factor would be $\left(\frac{|E|}{|V|}\right)^k$. Several methods then attempt to improve the algorithm performance - the *Operator Decomposition* reduces the branching factor by iteratively adding actions for the agents in a given order in new steps [8], the *Independence Detection* decouples the MAPF problem to smaller problems with fewer agents by grouping those which would collide on their optimal paths [8], and M* reduces the branching factor by attempting the progress of all agents by one step in their optimal path and expanding only in the case of a collision [9].

Coming from mobile robotics applications, where the robots can be allowed to wait in one place, the need for time discretization creates an even larger downturn. Furthermore, the planning time needs to be reduced as much as possible. Planning in continuous time therefore brings in the idea of time intervals or windows. However, even though the solvers based on these approaches are simpler and faster, they no longer provide optimality.

The Safe Interval Path Planning algorithm (SIPP) works with the idea of planning for one agent in a dynamic environment [10]. The nodes in the search graph are extended with time intervals, which are either $safe$ or $collision$. The A* algorithm expansions are then constrained to enter only the $safe$ intervals. The dynamic obstacles in the environment can be either considered as some predicted, but not controlled entities, or as the previously planned agents. Note, that this differentiation also brings in the idea that the planner should take into account the reliability of the trajectories it expects the other agents to follow. This uncertainty will be an important topic in Chapter 7.

The Context-Aware Route Planning algorithm (CARP) similarly works with so-called time windows. These, either $free$ or $occupied$, form an occupancy table of the environment [2]. When expanding the states, the A* algorithm checks their accessibility in the occupancy table the same way as it is done in SIPP. CARP generally considers all entities in the area as controlled and attempts to produce plans for all of them sequentially. The algorithm will be described in detail in Chapter 3, as it is used as the base for the proposed approach.

The sequentiality is then also a variable option since, due to the nature of planning in time, the plans cannot be produced by parallelized planners. However, it can be distinguished whether the whole set of assignments is known in advance and their order can be changed, or if the assignments come in sequentially and any change of order needs to be applied in runtime - after some agents were already planned. The influence and benefits of such order changes will be described in Section 2.3.

2.2 MAPF solutions evaluation

In order to evaluate the qualities of the planning algorithms and solvers, several criteria can be measured and compared. These will also be used to compare the qualities of the proposed methods in Chapter 4.

Firstly, it is essential that the algorithm finds a solution - a set of collision-free plans for all given agents - and it finds it in a reasonable time. The ability to find a plan is measured by the *failure rate*. That stands for the percentage of setups from some representative set that the algorithm fails to solve. The *planning time* then determines how much time the algorithm needs to find a set of plans successfully. In order to maintain the validity of the value, unsuccessful planning attempts should not be included in the computation of an average *planning time*.

The second evaluated property is the quality of the plans. Again, only full sets of plans should be considered. The main measure is the cost of the plans. For a general planning algorithm, the cost can simply be the length of the resulting path, however, when planning in the time domain, the agents can wait on nodes, and this needs to be considered as well. Duration of the plans is usually denoted in two ways:

- *makespan* - the time by which the last agent in the set reaches its goal
- *flowtime* - the sum of execution times across all agents [11]

In this work, *makespan* will be used to describe the duration characteristic of the plans. The actual covered distance can be useful as well to see how many nodes the agent visits. This can help to compare whether it is more beneficial to wait for a long time or choose a longer path. This measure will be regarded as *length* of plans.

2.3 Agent planning order

The order in which the assignments are taken - in which the agents are planned - can have a significant influence on the planning and its results. Enabling more complicated and longer paths to be planned with a higher priority than short ones seems to provide a larger chance of finding the set of plans [12] [13]. The ability to change the order, however, is either restricted to planning the whole set of agents together with the knowledge of it in advance, or it tends to increase the planning time for each agent, as this section shows.

The CARP author presented the idea of order changes along with a highly effective *Longest First* (LF) heuristic, which sorts the agents based on their optimal single-plans costs [12]. The procedure plans each agent with an A*

algorithm to retrieve the costs and then sequentially uses the CARP algorithm to plan them in the order going from longest to shortest. The cost can either be considered as the *makespan* or the actual path *length*. Increasing the priority of the costlier plans has proven to reduce the *failure rate* highly. However, the *makespan* of the resulting plans may get worse with the original order, since the ordering is based on the *length* of the optimal paths and therefore optimizes it [1].

In my bachelor thesis, it was shown that changes in the order of sequentially assigned tasks during runtime also tend to improve the overall results without exceeding a reasonable planning time limit given by the application [1]. The proposed methods attempt to sort the assignments during the planning process heuristically. In order to do so, the idea of influence between agents' plans was introduced. The influence is derived from the overall distance between the agents on their routes in time.

During the sequential adding of agents to the process, a group of the most influencing agents was found for each new assignment. The group was formed using a distance matrix, and taken from the planned group to be re-planned afterward. Two procedures used this approaches, the *Direct group* (*DG*), taking the group of the most influencing agents directly, and *Growing group* (*GG*), which repeatedly adds the one most influencing agent to the group up to some limit. In a set of experiments, it was shown that after changing the order of the agents (moving the influential agents to the end each time) and replanning, the quality of plans, and the chance they will be found rises. That is achieved even with a small size of the group, namely 4-6 agents out of 100.

A comparison of *failure rate* between the named algorithms is shown in Figure 2.1. The results were produced in a set of experiments with the setup, that is described in detail in Chapter 4, since it is used for this work as well. The two versions of CARP numbered 10 and 100 worked with a set of assignments known in the beginning and performed a corresponding number of shuffles of the order to find the best result among them. This approach was, of course, time costly. The *DG* and *GG* methods are numbered by the size of the replanned group. These methods received the assignments sequentially [1].

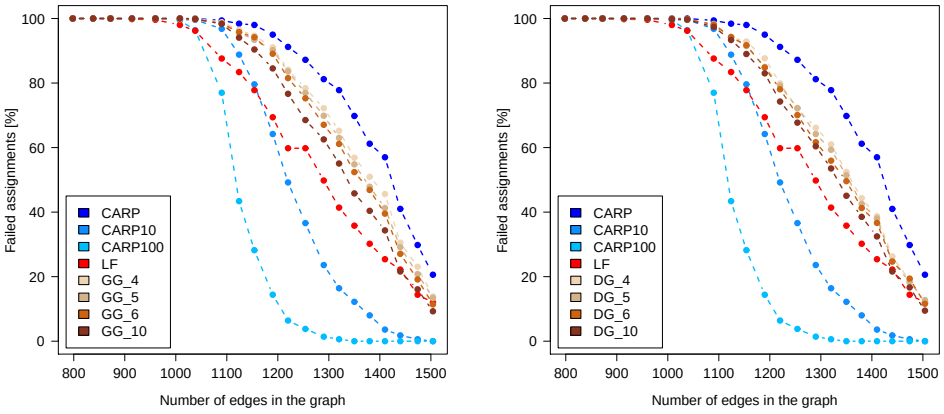


Figure 2.1: Comparison of *failure rate* of CARP with no sorting of assignments, random shuffles and several sorting heuristics, *LF*, *DG* and *GG*. [1]

2.4 Anytime and lifelong path planning

A different area of study aims at algorithms, which can work with old planning information and reuse it. The anytime algorithms usually quickly produce a sub-optimal path and then reuse the search knowledge to improve the solution. They do so by inflating the heuristic used by an A* algorithm, making it a weighted A*, which often provides speedups, but it is at the cost of solution optimality [14]. However, if the heuristic is consistent, multiplication by an inflation factor $\epsilon > 1$ is proved to bound the resulting cost to ϵ times the optimal cost, which is the idea behind the Anytime Repairing A* (ARA*). The algorithm repeatedly searches the state space while it lowers the inflation factor. During that, it reduces the number of expansions in new iterations by expanding only the states whose cost may no longer be valid with the new ϵ value [15].

The Anytime Safe-Interval Path Planning algorithm (A-SIPP) even applies the anytime approach to the multi-agent planning problem [16]. It does so by introducing a time horizon beyond which it ignores the dynamic obstacles/other agents and therefore speeds up the plan production while guaranteeing completeness of the resulting set of plans. Another possible benefit of the planning knowledge preservation for multi-agent planning will be proposed in the next chapter.

The lifelong planning algorithms, on the other hand, store their old planning knowledge to incorporate changes in the environment and adjust the path only locally, which highly reduces the planning time needed. The most widely used

algorithm continuously maintaining a least-cost path in a partially unknown or dynamic environment is D*Lite [17].

Chapter 3

Memory-based Context-Aware Route Planning

The possibility of reusing old planning knowledge to either improve or adjust a plan has an interesting potential for multi-agent planning and some specific situations. A setup that could make use of this idea is replanning of a set of already existing plans. That can either be triggered by the need of changing the order in which the agents get their plans during the sequential planning or in the case when a strictly high priority agent enters the environment.

In both cases, a poorly effective way would be planning all other agents again from scratch. At that point, any reusable information from the previously planned routes seems to be beneficial. To prove this idea, an algorithm called Memory-based CARP (MCARP) is proposed. The algorithm very simply uses the old plan for each agent that is anyhow replanned to improve its heuristic. By that, it strongly reduces the number of expanded states, which directly leads to a reduction of the planning time. Furthermore, with the planning time reduced, the algorithm can aspire to improve its qualitative results as well. It will be shown, that even with the change of order of the agents, which leads to improvement in both *failure rate* and *makespan*, the planning time will still be held bellow the time of the original CARP algorithm planning from scratch.

In this chapter, the multi-agent path finding problem will be formulated, as well as the situation of the replanning. After that, the original CARP algorithm will be described in detail as it is an effective tool for solving the original problem. As a solution of the replanning problem the MCARP algorithm will be finally proposed with a detailed description of its function.

3.1 Data structure

To achieve the desired algorithm performance and properties of the generated results, which is especially that they are collision-free, a suitable state-space notation is necessary. Based on the real-world motivation, the automated warehouses applications, a promising representation is a graph. Mainly it is because the agents, AGVs in the real world, are expected to follow defined roads and pass through set intersections. A graph representation then easily fits the warehouse area.

The graph $G = \{V, E\}$ consists of vertices V, matching intersections of roads, and edges E, the roads themselves. Each edge has a defined length and the vertices have a defined time for passage through them. For agents with a defined constant speed, it is then possible to compute the time for which they occupy each vertex or how long it takes for them to pass an edge to the next vertex. This is vital for working with continuous time, since there are no set time steps. Instead, the times of reaching and leaving a vertex form the so called time windows.

The formulation of an occupancy table is then apparent, since for each vertex, it stores a set of *occupied* and $free$ time windows denoting the usage of the vertex in time. For convenience, each time window contains not only its start and end times, but also the identifier of the vertex v it belongs to. A time window is therefore a tuple $tw = t_s, t_e, v$, where t_s and t_e are the start and end time respectively and v is the corresponding vertex identifier.

3.2 MAPF formulation

Let us have a set $A = \{a_1, a_2, ..., a_n\}$, where $a_j = \{r_s, r_g, t\}$ is an assignment for the $j - th$ agent stating that at time t it appears on the start vertex r_s and desires to go to the goal vertex r_g. Based on this set of assignments, an algorithm solving the MAPF problem should produce a set of collision-free plans $P = \{p_1, p_2, ..., p_n\}$, where the $j - th$ plan p_j is a set of time windows tw_k. As noted above, each time window consists of the vertex which the agent passes through, and the times of entry and departure. To maintain the collision-free property, each of the time windows in the plans should appear only once in the whole set and the windows for the same vertex in two different plans must not intersect. Furthermore, it needs to be ensured that agents don't attempt to switch positions on two neighbouring vertices.

The set of assignments is assumed as finite, but repeated adding and planning of new assignments is also possible [1]. In any real application, the agents usually do not stay on the final vertex, but rather they continue with another assignment. However, for evaluation of the algorithms, let us assume all of the agents start at time $t = 0$ and stay in their target forever.

3.3 CARP

The Context-Aware Route Planning (CARP) is an A*-based algorithm for solving the MAPF task. A pseudocode of the CARP algorithm is shown in Algorithm 1. At the first 3 lines, it attempts to find a starting window in the starting vertex r_s, so that the agent can be deployed in the graph in the given time t. The algorithm then searches the graph G with a standard A* search with the cost function $f(w) = g(w) + h(w)$, where $g(w)$ is the real cost of the partial plan to w and $h(w)$ stands for a heuristic estimate to the goal. Since CARP needs to distinguish the time at which the vertices are visited, the *open* list contains the time windows.

The choice of the value $g(w)$ has to be made based on the criteria that should be optimised. It can either represent the distance or node count covered so far, optimising the covered distance. However, the shortest path can last longer and while planning in the time domain, optimisation of *makespan* as defined in Section 2.2 can be more useful. Especially in the autonomous warehouse application, where time is highly valued. Therefore, the compared algorithms based on CARP work with $g(w) = \tau_{start}(w)$. The cost of a state is the time at which it is reached.

The algorithm pulls the time window with the lowest heuristic cost $f(w)$ from the *open* list at line 5, adds it to the *closed* list and retrieves the vertex identifier. If it is the goal vertex r_g and it will be $free$ forever, the algorithm finishes at line 8 and reconstructs the found path using backpointers stored in the visited time windows.

Otherwise, it computes the time at which the agent will be able to leave the vertex from the entry time - the start of the window - and the duration $d(r)$ of the vertex r, which is the time to pass through it. After that, the expansion is performed at line 10. The set $\rho(r)$ stands for all time windows in the neighboring vertices of r. For each of them, the entry time is computed and the algorithm checks whether the window is reachable.

The reachable windows are then given the entry time and a backpointer to the window that led to them and are put in the *open* list.

Algorithm 1 CARP - Plan route [2]

Input: $a = \{r_s, r_g, t\}$ − assignment; start and goal position, start time
$G = \{V, E\}$ − a graph; vertices (free time windows) and edges
Variables: open, closed
Output: shortest-time conflict-free route plan for a
$G = \{V, E\}$ − an updated resource graph

1: **if** $\exists\ w\ [w \in V \mid t \in \tau_{start}(w) \wedge r_s = resource(w)]$ **then**
2: $add(w, open)$
3: $entryTime(w) \leftarrow t$
4: **while** $open \neq \emptyset$ **do**
5: $w \leftarrow argmin_{w' \in open} f(w)$
6: $add(w, closed)$
7: $r \leftarrow vertex(w)$
8: **if** $r = r_g \wedge \tau_{end}(w) = \infty$ **then return** $followBackPointers(w)$
9: $t_{exit} \leftarrow \tau_{start}(w) + d(r)$
10: **for** $w' \in \{\rho(r) \backslash closed\}$ **do**
11: $t_{entry} \leftarrow max(t_{exit}, \tau_{start}(w'))$
12: **if** $\tau_{start}(w') \leq t_{entry} \wedge t_{entry} < \tau_{end}(w')$ **then**
13: $backpointer(w') \leftarrow w$
14: $\tau_{start}(w') \leftarrow t_{entry}$
15: $add(w', open)$
return null

3.4 Replanning in MAPF

As described above, the standard planning process for a group of agents in a graph-based environment works with the graph $G = \{V, E\}$ and the set of assignments A to produce a set of collision-free paths P. Apart from some possibly pre-computed properties of the graph, which will be applied in Section 3.5, the algorithm does not receive any further information. This changes in a special case, when a set of plans already exists and the task is to adjust them.

Specifically, this happens when the environment is accessed by one or multiple agents, that have a higher priority than the previously planned ones. These higher priority agents will be assumed as planned externally, with their set of collision-free plans H representing an additional set of dynamic

obstacles in the graph G. The replanning should then produce a new set of plans P_{new}, so that the joint set $\{H \cup P_{new}\}$ is collision-free.

The replanning process works with the same graph G and the set of assignments A as the standard planning. The occupancy table is not completely free at the beginning, since it already contains the set of plans H, so the old plans may no longer be valid due to possible collisions. Yet, the set of plans P exists and might provide useful information for the replanning.

3.5 MCARP

The proposed MCARP algorithm is an extension of CARP aiming to solve the replanning MAPF task. The main idea behind it is in using the previously produced plans. Therefore it cannot solve the original MAPF task. The MCARP also utilizes the opportunity to sort the agents before their replanning in order to achieve better results. A simple modification of A* estimate to goal heuristic is applied as well.

3.5.1 Distance to goal

As it was already mentioned in Section 3.3, the cost $g(w)$ is considered as the time the vertex w is reached, so that the algorithm optimizes the duration of the plans, the *makespan*. The first improvement attempt done by MCARP is adjusting the second part of the state cost computation, the heuristic estimate to goal $h(w)$. This value is usually computed as the Euclidean distance from w to the goal vertex, which is also the case of CARP. The relevancy of this value depends on the area. If it is possible that the path can aim almost directly to the target, it is effective. However, in an area with directed roads, such as in an automated warehouse, the Euclidean distance can be misleading. Since the area is known, estimating the distance can be replaced by the exact distance, which can be pre-computed. Based on that, an optional metric using the exact distance to the target has been added to MCARP.

Before planning, a distance matrix of the graph G is computed with the application of the Floyd-Warshall algorithm [18]. This algorithm is used for finding the shortest paths between all pairs of nodes in a weighted graph with positive or negative edge weights, but with no negative cycles. That

condition is fulfilled in a graph with positive edge weights representing the traverse time.

The time it takes to pre-compute this matrix is not included in the planning time since the graph is the same for the whole process (planning and replanning of all agents), and the distance matrix is considered as an accessible attribute of the graph. This assumption holds for the automated warehouse application as well, since the structure of it does not change, and the distance matrix has to be computed only once. The distance matrix provides the optimal path length to goal $h_d(w)$. Then $h(w) = h_d(w)/v$, where v is the agent speed, is the optimal time to reach the goal. However, the optimal path cannot be expected to be collision-free. Therefore the time $h(w)$ is still only an estimate.

3.5.2 Plan reusing

The main extension of the state cost $f(w)$ computation is the incorporation of old plans. Since the number of the incoming high priority agents is expected to be limited, so can be expected that the plans P will need only local modifications, and most of them will still fulfill the collision-free property. If the algorithm would only modify the path in the area of a potential collision and kept most of it, the same way as D*Lite does for single-agent path-finding, the replanning might be achieved in a significantly shorter time. The algorithm is therefore motivated to stay close to the old path by using the old path of the $j-th$ agent P_j to extend the cost computation formula to the form

$$f(w) = g(w) + h(w) + d(w, P_j). \tag{3.1}$$

The added element $d(w, P_j)$ is the Euclidean distance of the vertex w from the old path P_j, i.e., from the closest vertex in it. If the vertex of the newly expanded time window is in the old plan, the cost is increased by 0. However, the further the search proceeds from the old path, the larger increase in cost is applied. An optional version of the heuristic adjustment also adds a multiplication of $d(w, P_j)$ by the length of the old plan $l(P_j)$, i.e., the number of resources it enters. That ensures that until it is possible to expand the old path, it will be prioritized in the open list. The benefits and downturns of this approach will be shown in the experiment part in Chapter 4.

It should be noted that this modification causes the heuristic to lose its admissibility property. This can be proven by a simple situation with two agents A_1 and A_2 originally planned in lexicographical order. Agent A_2 could be at some point, avoiding the path of agent A_1 by following a longer path, which would be faster in that case. Then, if the order of the agents

changed for the replanning, this path and the resulting plan would no longer be optimal. However, the modified MCARP heuristic would penalize the currently optimal plan while trying to support the old one.

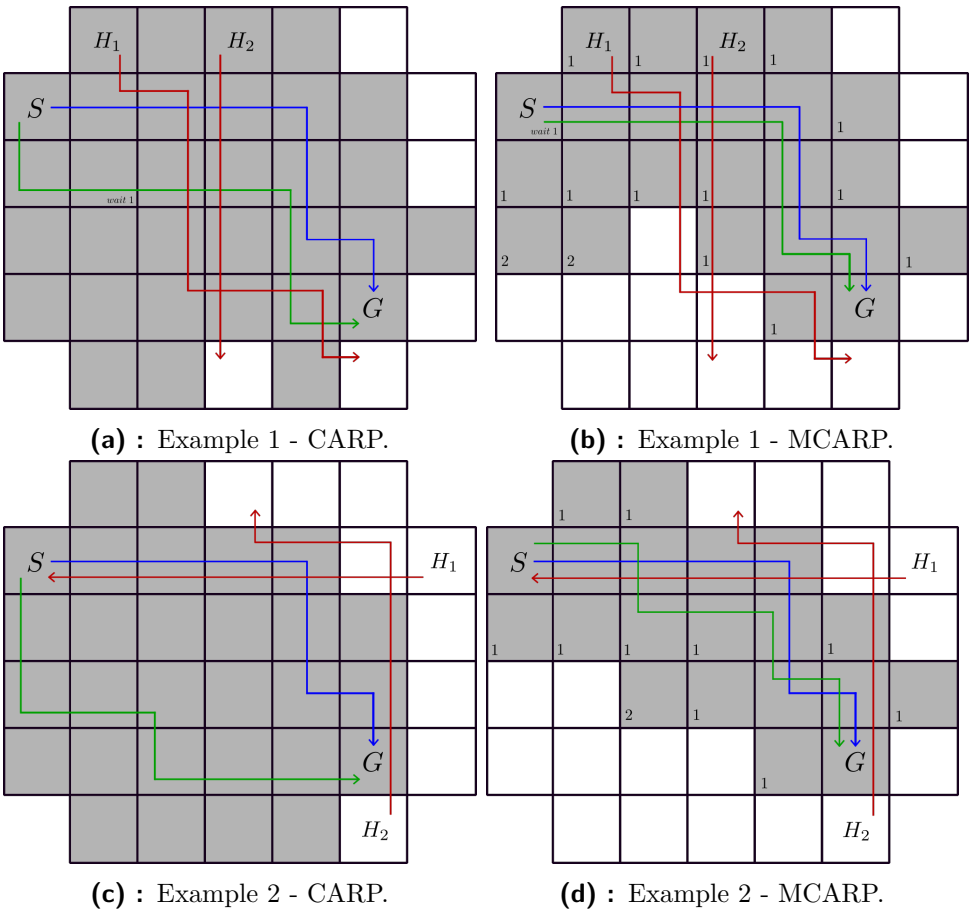
This property of the MCARP algorithm needs to be taken into account, and the sorting should ensure the reduction of the possibly increased plan cost.

An essential part of the extension is the calculation of the distance from the old path. This is performed by finding the closest vertex in the old path for each newly expanded time window. The complexity of the minimum value search is $O(l(p))$ and, if the evaluated vertex is present in the old path, the search can be quicker. The distance is then retrieved from a pre-computed distance matrix, so the complexity is constant.

This plan reuse should lead to the reduction of expanded vertices since the algorithm can search only a limited area in which it is led by the old path. This result is shown in two examples in Figure 3.1. An agent is planned from node S to node G in a 4-connected grid, the original plan in an empty map is marked by a blue line and is the same in both examples (**(a)**, **(b)** and **(c)**, **(d)**). After this plan is produced, two higher priority agents, H_1 and H_2 are planned. Their assignments are different in the two examples, and the resulting paths are marked by red lines.

Figures **(a)** and **(c)** show replanning by CARP with the resulting path marked by green line and Figures **(b)** and **(d)** then show the result for MCARP. The grey tiles are those which the algorithms added to the $OPEN$ list - at least one time window in them was evaluated. For CARP, it is 47 windows in the first and 43 in the second example. MCARP added 24 and 23 windows, respectively. Additionally, these tiles are marked with their distance from the original path (which is zero for tiles of the path). The number of time windows actually expanded in the first example is then 23 by CARP and 10 by MCARP. In the second example, it is 19 windows expanded by CARP and 9 by MCARP.

The comparison shows that MCARP narrows the search to the area of the original path and even returns to it after it moves the agent away from the higher priority ones. The number of expanded time windows shows that in these cases, MCARP expanded only the new path, reducing the number of expansions to half when compared to CARP. A promising attribute is that the path produced by MCARP is of the same *length* and *makespan* as from CARP, but this cannot always be expected due to the non-admissible heuristic. In Chapter 4, a larger scale statistical comparison is presented to



(a) : Example 1 - CARP. **(b) :** Example 1 - MCARP.

(c) : Example 2 - CARP. **(d) :** Example 2 - MCARP.

Figure 3.1: Replanning using CARP and MCARP. Blue path shows the originally planned path from S to G, red paths are consequently added higher priority agents H_1 and H_2. In green is the replanned path by each algorithm. Tiles each replanning algorithm added to $OPEN$ list are marked grey. Numbers in **(b)** and **(d)** denote the distance from the original path.

show that this way of old-path-prioritization significantly reduces planning time while providing similar qualitative results as replanning from scratch.

3.5.3 Sorting of agents

As it was described in Chapter 2, the order in which agents are planned has a significant influence on the quality of the plan and the possibility to find some at all [1]. An example of the influence is shown in Figure 3.3. If agent A_1 plans first, it receives its path and follows it. Agent A_2 will not be successfully planned in that case, because it cannot evade agent A_1 and its starting node is the goal of A_1. However, if agent A_2 would be planned first, it can execute its plan while A_1 will wait on its start until A_2 moves away.

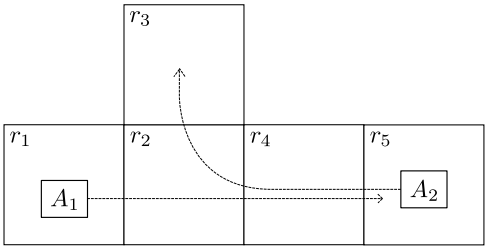


Figure 3.3: Two agents in a narrow corridor example. [2]

The situation of replanning due to a high priority agent arrival provides an opportunity to apply the change of order as well since all plans need to be revised to maintain the collision-free property. Several methods for sorting the agents were presented in Chapter 2. Their application in replanning can be much simpler since all of the agents are replanned at once. There is also the full knowledge of their old plans. Those can be used as a better estimate of costs for the LF heuristic, or for pre-computing the distance matrix of agents in time. That can serve for a simplified version of the DG method, which will sort all agents by their influence with others.

The distance matrix that denotes the influence between agents is produced by the Algorithm 2, which was proposed along with the DG and GG heuristics [1]. The algorithm was optimized by removing unnecessary for-loops and incorporating the computation of the sums of the matrix columns in the matrix generation to avoid further iterations later. The algorithm also saves half of the processing time by only computing half of the matrix, which is above the main diagonal (performed at lines 8-10). The other is copied since the matrix is symmetric.

The set of sums $S = \{s_1, s_2, ..., s_n\}, s_i = \Sigma_{j=1}^{n} d_{ij}$, where d_{ij} is the mean Euclidean distance between $i-th$ and $j-th$ agent, then provides a numeric representation of the influence on each agent. The lower this number is, the closer the agent was to the others and the larger influence on him is expected. Therefore, the set S is used to sort the agents. Since the resulting method differs from the original group-replanning methods (DG and GG), the heuristic sorting all agents by the influence on them will be called $Influenced\ First$ (IF).

Several other methods for the sorting of agents are proposed as well. These build directly on the old plan knowledge. The first sorting mechanism comes from the idea of the LF heuristic, but it aims at reducing the long waiting times among the old plans. A situation can be imagined, where an agent could pass through a narrow passage and proceed to a less occupied area, but due to reaching it later, it has to wait a long time before several other agents pass. If the agent had a higher priority, it could pass through earlier, and the

other agents could be slowed down insignificantly. To test this hypothesis, the $Maximum\ Wait$ (MW) heuristic sorts the agents by the longest waiting time present in their old plans. Similarly, the $Overall\ Wait$ (OW) heuristic sorts the agents by their overall waiting time sum.

Algorithm 2 Distance matrix [1]

Input: $P = \{plan_i\}_{i=1}^{n}$ − a set of n planned routes
Output: M − a square matrix of size n representing mean distances between routes
S − a set of column sums of the matrix M

```
1:  for j ∈ 0, 1, ..., n do
2:      for k ∈ 0, 1, ..., n do
3:          if j == k || P[j] == ∅ || P[k] == ∅ then
4:              M[j][k] = ∞
5:          else if k < j then
6:              M[j][k] = M[k][j]
7:              S[j] += M[j][k]
8:          else
9:              d = ∅
10:             for step_j, step_k ∈ length(P[j]), length(P[k]) do
11:                 notSet = true
12:                 while notSet do
13:                     tw_j = P[j][step_j].timeWindow
14:                     tw_k = P[k][step_k].timeWindow
15:                     if tw1 ∩ tw2 then
16:                         resource_j = P[j][step_j].resource
17:                         resource_k = P[k][step_k].resource
18:                         notSet = false
19:                     else
20:                         if tw_j.start < tw_k.start then
21:                             step_j + +
22:                         else
23:                             step_k + +
24:                 d ∪ euclid_dist(resource_j, resource_k)
25:             M[j][k] = d̄
26:             S[j] += M[j][k]
    return M, S
```

Chapter 4

Algorithm evaluation

In order to evaluate the contribution of the proposed adaptations to the CARP algorithm in the replanning task, a set of experiments was performed on a computer equipped with Intel Xeon E5-2690. The proposed MCARP algorithm is compared to the original CARP algorithm, while also only partial modifications are applied, to evaluate the effect of each of them properly. The comparison is performed based on the characteristics described in Section 2.2, that is:

- *Failure rate*
- *Planning time*
- *Makespan*
- *Path length*

4.1 Experiment setup

The ability of the algorithms to produce a set of collision-free plans is mostly tested when the state-space consists of fewer states, graph vertices in this case, and therefore the agents are less operable. In order to test the methods' capabilities as much as possible, the experiments were performed on a set of 21 maps. All of the maps, in the form of a graph $G = \{V, E\}$, are built on

the same set of 400 vertices V. The sets of edges E_j, where $j \in (0; 20)$, in the maps then form a set ranging from a spanning tree to a mostly 4-connected graph. These maps were created by generating a complete square graph with the size of 20*20 vertices. The graph was then simplified to a spanning tree, and, finally, approximately 50 random edges from the complete graph were added 20 times, to create the set of 21 maps of density ranging from 800 to 1500 edges in the graph. An example of a generated map with 1190 edges is in Figure 4.1.

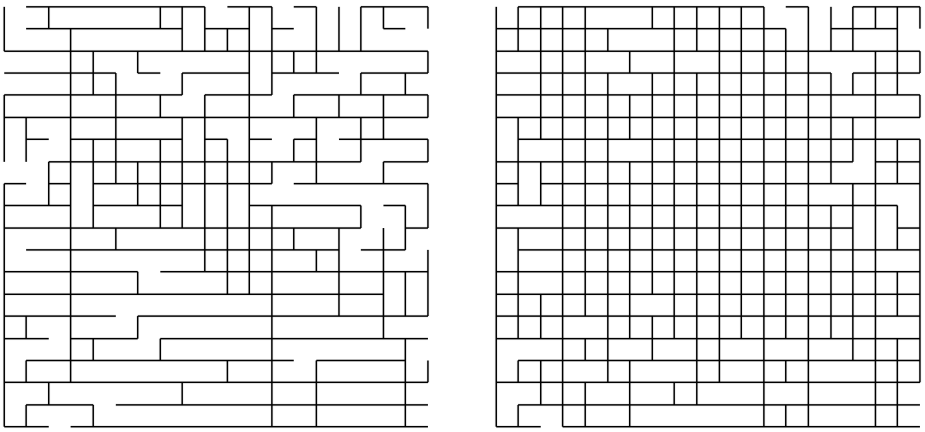


Figure 4.1: Generated maps with 1008 and 1350 edges (504 and 775 two-way edges respectively).

A set of 500 random assignments, each for a group of 100 agents, was then generated for planning on each of the 21 maps. The sets of maps and assignments are the same as those used for the experiments in [1]. The sets of assignments were then split into two subsets - the standard agents and the high priority ones. To evaluate the influence of the size of this group, it's size ranged from 0 to 10 agents.

The experiments were then performed in two waves, as the task suggests. Firstly, the agents were planned by the original CARP algorithm (using a distance matrix proposed in Section 3.5 for a heuristic estimate to goal computation). Then, the high priority agents were planned, with the possibility of a failure here taken to account as well - such cases were not considered as the failure of the replanning algorithms. Especially in the less dense maps, the inability to find a plan even for a lower number of agents is expected. Finally, the main group of agents was replanned with the constrained occupancy table and with the availability of the old plans for the methods that use it.

4.2 Compared algorithms

We study several variants of the MCARP algorithm, which can be categorized by the choice of memory use, estimate to a goal used, and the sorting. Firstly, the versions use either no memory, purely the distance from the old path added to heuristic, or the forced prioritization of old path nodes. These memory versions are labeled as follows:

- M_0 - no memory applied
- M_1 - distance from the old path added to heuristic
- M_2 - old path nodes hard-prioritized

Differentiation was also made by application of the duration to the goal metric used for the heuristic estimate to the goal, which is compared to the standard Euclidean distance used. Both were used to pre-compute a distance matrix later used by the planner. The labeling is δ for the duration to goal and ε for Euclidean distance to the goal.

Finally, MCARP versions are compared based on the sorting heuristics. The applied were:

- NS - not sorted
- LF - based on the *makespan* of each old plan
- IF - based on the influence between agents in the old plans
- MW - based on the longest window (waiting) in each old plan
- OW - based on total excess waiting in each old plan

The proposed methods are compared to the original CARP algorithm. It solves the replanning task the same way as it does for the standard planning; it plans all assignments from scratch. Generally, it is the same algorithm as MCARP with the Euclidean distance to the goal estimate, unsorted agents, and no memory used. Therefore, the results of the replanning CARP will be named M_0-ε-NS according to the above notation.

4.3 Evaluation

To evaluate the influence of the A* heuristic adjustments (memory and distance to goal estimate), the algorithms were first grouped based on the sorting versions. Each of these groups was then compared to the results of the unsorted versions and between each other. Based on this comparison, the best settings could be selected to perform the final comparison also between the sorting heuristics.

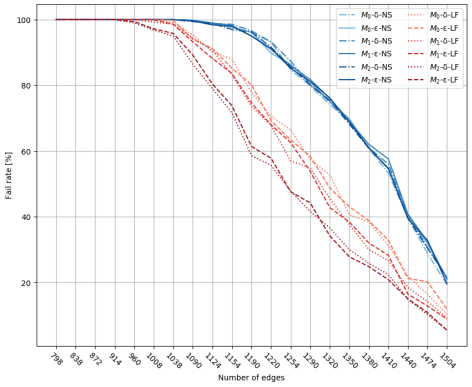
This processing was based on the growing density of the maps. The number of incoming high priority agents for this set was 5, resulting in a group of 95 planned agents. The influence of the high priority group size will be shown as well, but the differences between algorithms' results were not that significant to compare them. Other characteristics and trends in the results will be addressed, as well.

4.4 Grouped results

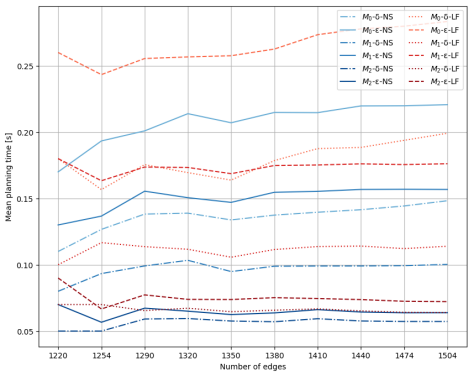
4.4.1 LF

The first evaluated sorting heuristic is the *Longest First* method. As can be seen in the Figure 4.2, in the combination with M_2 it significantly reduces $failure\ rate$. This version of memory usage also expectedly reduces expansions and therefore the *planning time*. However, application of the LF sorting strongly increases the *makespan* leading to worse results than those achieved by the original CARP algorithm. This applies for all kinds of memory usage.

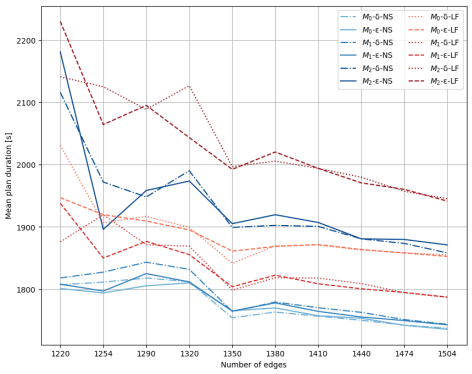
The overall best performance can be seen with the usage of M_1. Even though in *makespan* it is outperformed by the versions not sorting the agents at all, the $failure\ rate$ reduction is still significant, and it also provides the shortest paths among all versions. Of the distance to goal metrics, the duration to goal provides slightly better results.



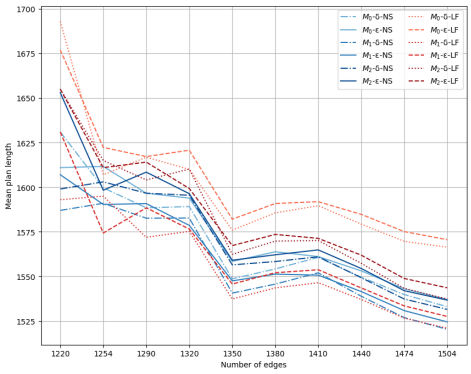
(a) : *Failure rate* depending on edges count in map.



(b) : *Planning time* depending on edges count in map.



(c) : *Makespan* depending on edges count in map.



(d) : Path *length* depending on edges count in map.

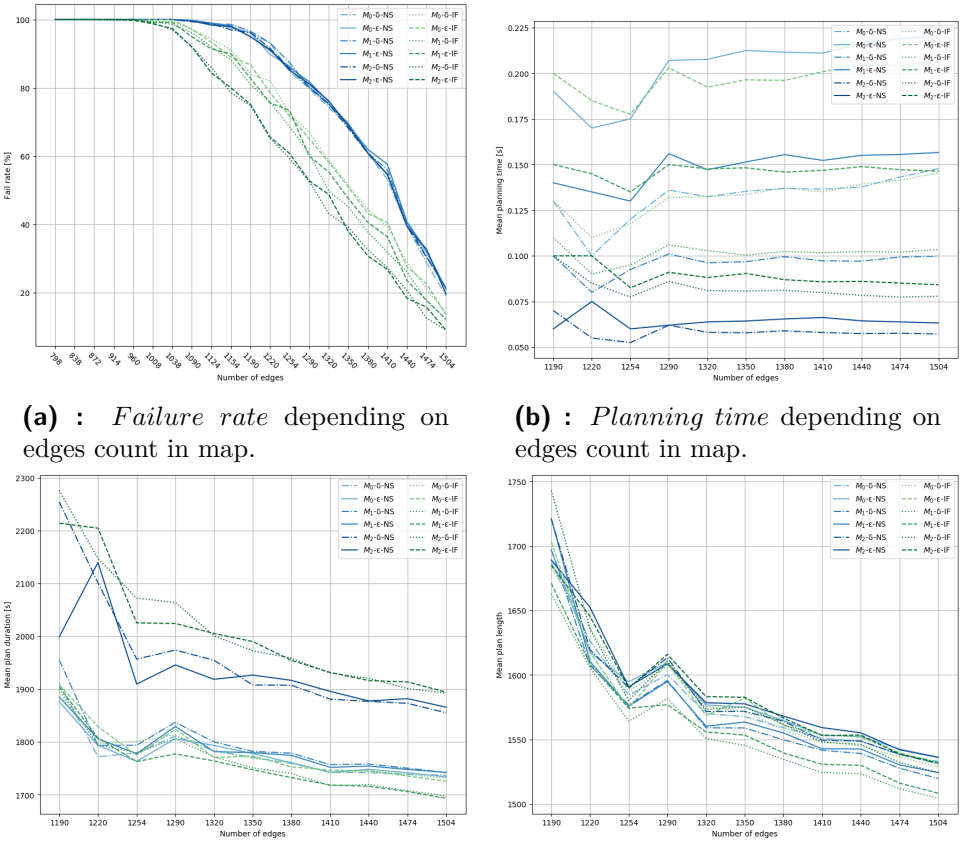
Figure 4.2: LF heuristic

4.4.2 IF

Similarly as for LF, the IF heuristic utilizes M_2 for $failure\ rate$ reduction, but forcing the agents to stay on their old paths causes bad qualitative results again, as seen in Figure 4.3. For this sorting algorithm, the softer memory usage achieves the best *makespan* and *length* results. For the IF case, the better metric is again the duration to the goal, although the difference is much smaller.

4.4.3 MW

Figure 4.4 shows that the MW heuristic performs the same or worse regarding $failure\ rate$, in some cases the increase is up to 10%. However, the MW



(a) : *Failure rate* depending on edges count in map.

(b) : *Planning time* depending on edges count in map.

(c) : *Makespan* depending on edges count in map.

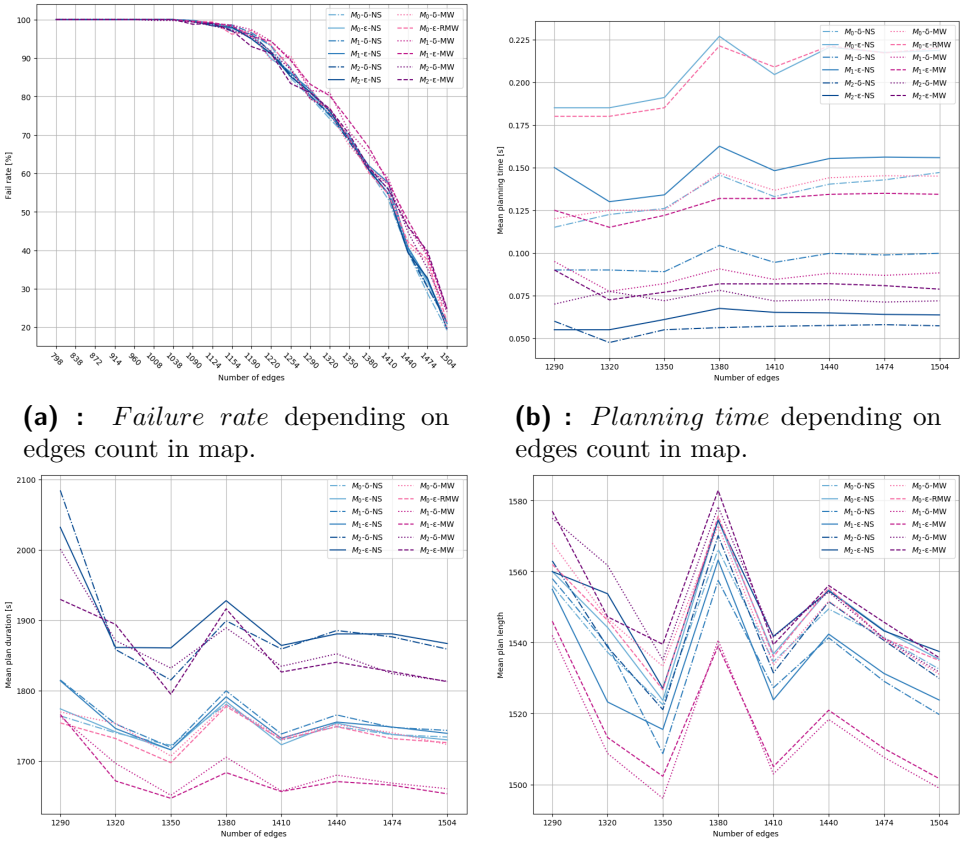
(d) : Path *length* depending on edges count in map.

Figure 4.3: IF heuristic

heuristic aims to optimise plan duration by reducing the waiting times. With the help of the old paths M_1, it is successful in this metric. The difference between distance to the goal estimates is again very small, although the Euclidean distance performs slightly better in *makespan*. Although the differences in path *length* may seem large between the maps, range of the y-axis is relatively small and the differences are within a 5% range from average.

4.4.4 OW

Reducing the overall waiting time leads to generally better results in *failure rate*, again mainly with M_2. The main benefit of this approach is, however, in *makespan*, where OW has even better results than MW, as Figure 4.5 shows. The estimate to the goal difference is again negligible and M_1 performs best overall.



(a) : *Failure rate* depending on edges count in map.

(b) : *Planning time* depending on edges count in map.

(c) : *Makespan* depending on edges count in map.

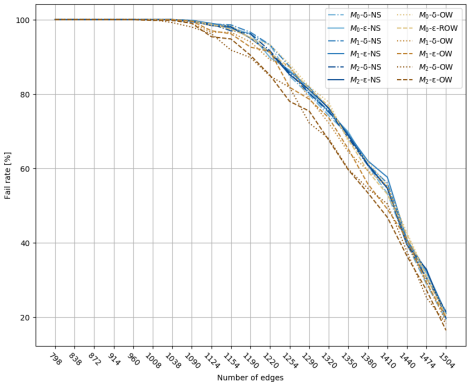
(d) : Path *length* depending on edges count in map.

Figure 4.4: MW heuristic

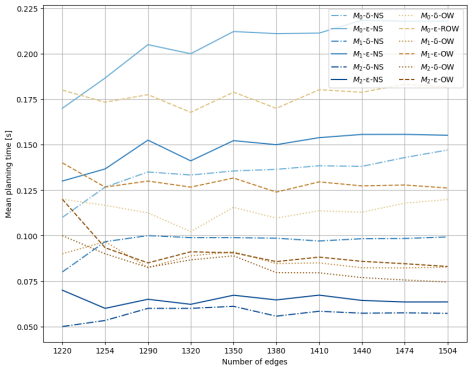
4.5 Sorting comparison

Figure 4.6 shows the comparison of the most promising algorithms. It is necessary to point out that the values for the algorithms can be different from those in their grouped comparison. For each set of plots, separate filtering was performed, so that they incorporate results for all assignments that all of the algorithms were able to plan.

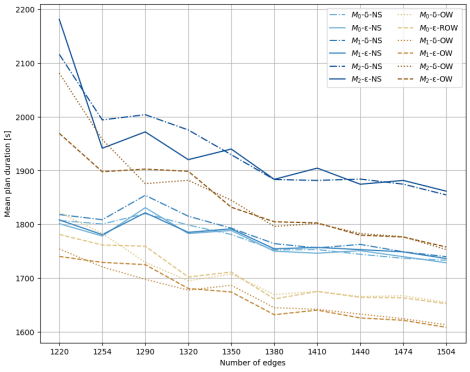
The combined benefits of M_2 metric combined with LF sorting in *failure rate* are clearly visible here, reducing the number of failed assignments by up to 20% compared to the original CARP ($M_0 - \varepsilon$). While these are also the fastest algorithms to find the results, the quality of the plans is significantly worse. Even with the otherwise successful M_1 metric, the *Longest first* sorting does not achieve the *makespan* results of the original CARP. However, the



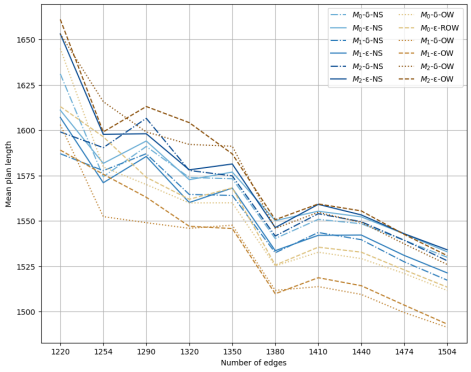
(a) : *Failure rate* depending on edges count in map.



(b) : *Planning time* depending on edges count in map.



(c) : *Makespan* depending on edges count in map.



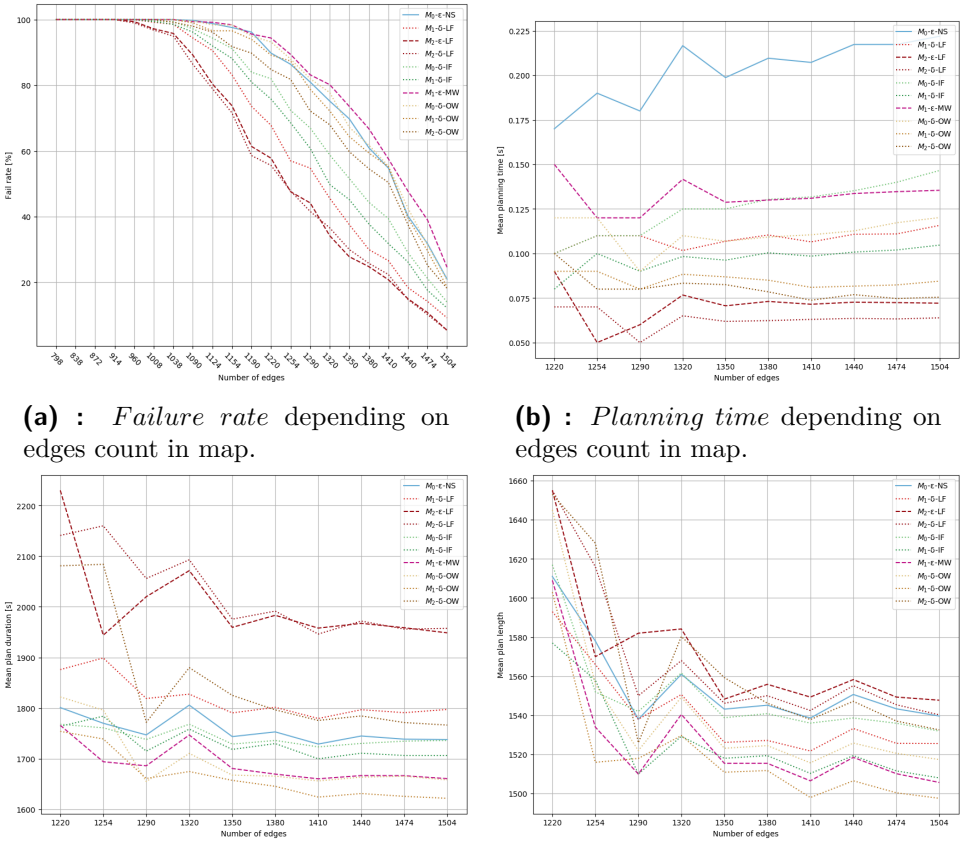
(d) : Path *length* depending on edges count in map.

Figure 4.5: OW heuristic

difference in *length* of the plans is smaller, and M_1 metric performs better than CARP.

The MW sorting with M_1 memory has promising results in *makespan* and plan *length*, but it struggles to find the plans at all. The modified version that considers overall waiting time instead of maximum, OW, has much better results. With both M_0 or M_1 memory, it is the best regarding *makespan* and M_1 tops in *length* as well. Both of these then maintain a similar $failure$ *rate* to CARP, cutting down the *planning time* by half.

The overall best performing sorting is IF. Even with no memory it reduces the $failure$ *rate* by 10-15% and by up to 20% with M_1. Furthermore, it also succeeds in decreasing both the *makespan* and *length*.



(a) : *Failure rate* depending on edges count in map.

(b) : *Planning time* depending on edges count in map.

(c) : *Makespan* depending on edges count in map.

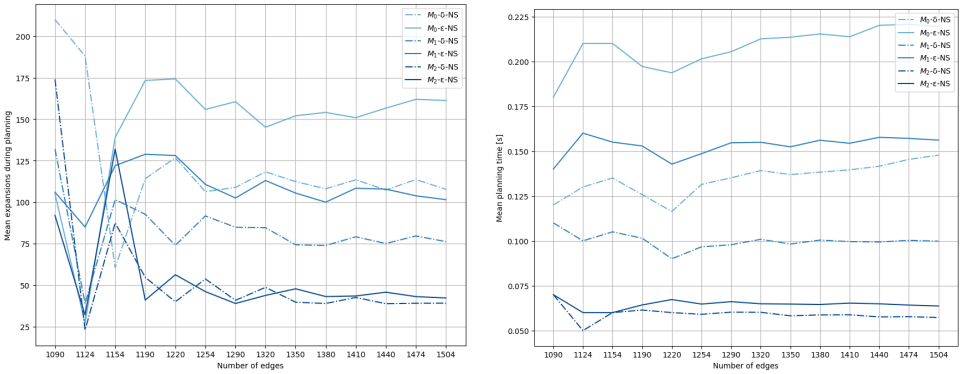
(d) : Path *length* depending on edges count in map.

Figure 4.6: Algorithm comparison

4.6 Expansions reduction

Algorithms using the old path memory achieve lower *planning time*. As expected, prioritizing the old path greatly saves up expansions needed to search the state space, and Figure 4.7 shows the correlation with the final *planning time*. Namely, the M_1 memory reduces around 30% of the expansions needed for planning, while M_2 cuts up to 60% expansions. The great reduction achieved by M_2 is, however, paid by the limited ability of the algorithm to improve the plan in terms of *makespan* and *length*. An interesting attribute is the usually top performance of M_2 in *failure rate*. The explanation is that these algorithms use the full knowledge of the old path, i.e., they already have a solution in terms of a path. They only have to add more waiting to follow it, which explains the increase in *makespan*.

Figure 4.7 also shows the significant difference in *planning times* between algorithms using the Euclidean distance and the time to reach the goal. Although it could be seen that the qualitative results of these two metrics do not differ that much, directing the search by the duration to target reduced the expansions. This result is expected since, with a more accurate estimate of a state's cost, the A* algorithm needs fewer expansions to reach the goal. With no memory applied, the reduction is around 20-30%, while with more memory applied, the individual effect of the estimate to goal gets smaller.



(a) : Number of expansions depending on edges count in map.

(b) : *Planning time* depending on edges count in map.

Figure 4.7: Comparing the number of expansions performed by planning algorithms and the resulting *planning time*.

4.7 High priority agent count

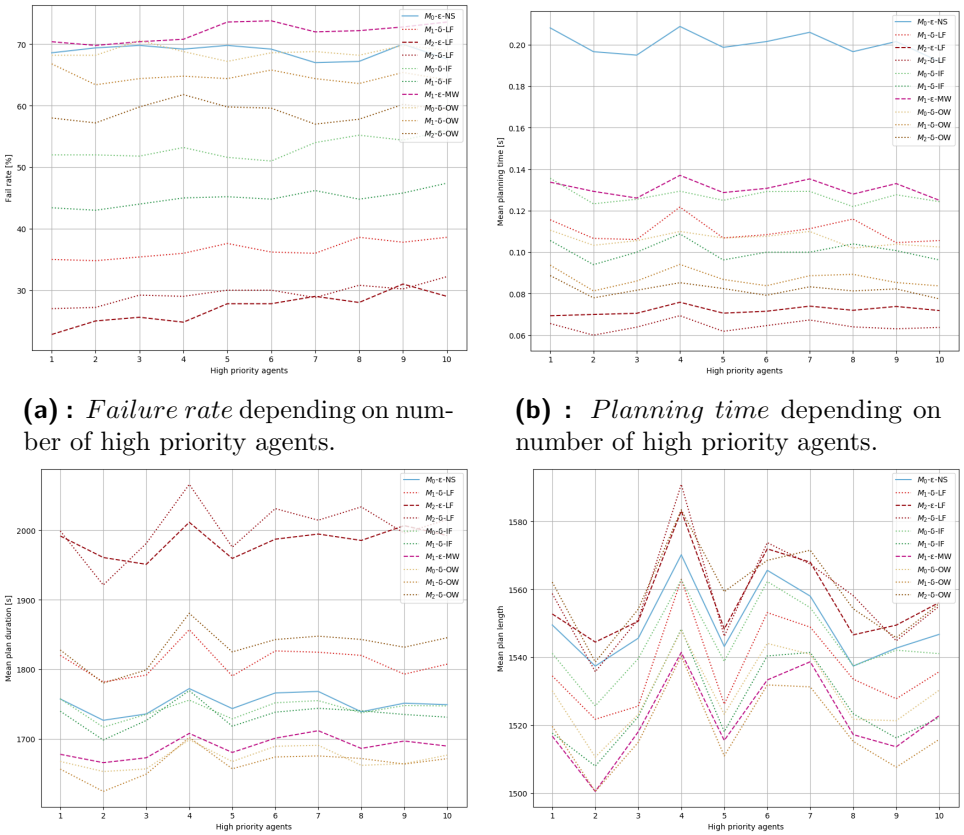
Figure 4.8 shows a comparison of the results of the top selected algorithms from Section 4.5 regarding the number of high priority agents introduced before replanning. Note that the resulting set always consists of 100 agents. The results are from planning on a map with 1350 edges.

This setup can be apprehended as a MAPF solver having gradually lower control over the situation, which remains the same. An increasing set of agents is planned from an external source, and the planner has fewer agents to control in a more restricted state-space. Based on this, it is expected that the results for each algorithm should not differ too much with the increasing number of high priority agents. Therefore, the results in Figure 4.8 may seem meaningless, but they confirm this assumption.

It can be seen that algorithms suffer a decrease in quality since the effect of the sorting gets smaller. With more pre-planned agents, this trend would

bring results of the sorting algorithms to the values of the unsorted version. In the end, all of the agents would be planned by the external source, the standard CARP, in this case.

The results in path *length* may seem noisy, but the differences of each algorithm are within a 5% range from the average *length* values among all high priority agent counts.



(a) : *Failure rate* depending on number of high priority agents.

(b) : *Planning time* depending on number of high priority agents.

(c) : *Makespan* depending on number of high priority agents.

(d) : Path *length* depending on number of high priority agents.

Figure 4.8: Algorithm comparison regarding the number of high priority agents in process.

4.8 Estimate to the goal metrics on a warehouse map

The motivation to use the real duration to goal δ instead of the standard Euclidean distance ε comes from the deployment. The warehouse area consists mainly of one-way routes, and in such case, Euclidean distance might be

misleading regarding the real-time it will take to reach the target. A set of assignments was therefore tested on a map of an actual warehouse shown in Figure 4.9 **(a)** consisting of 490 nodes connected with 1520 partly one-way routes. The testing set again consisted of 500 sets of random assignments for 19 agents each, with only one high priority agent introduced. The number of agents corresponds to the real warehouse deployment.

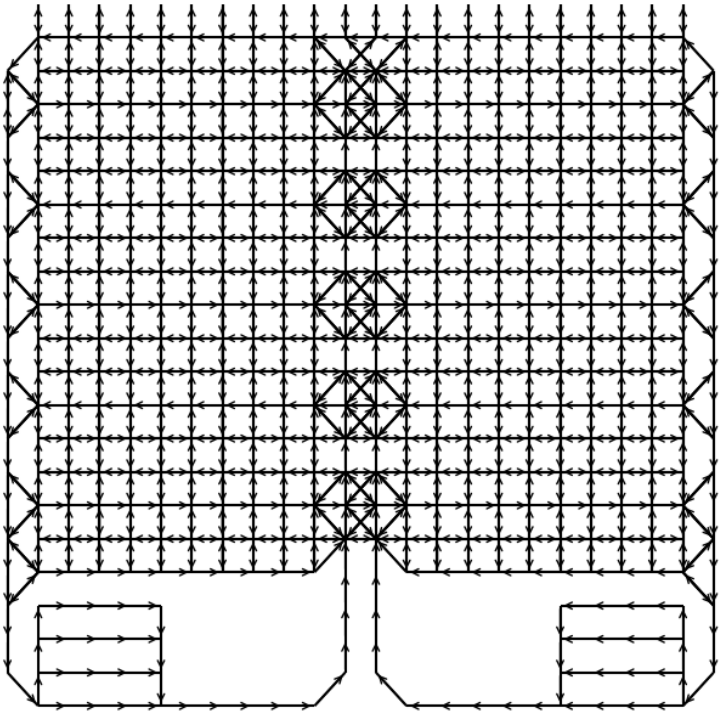
Figure 4.9 **(b)** shows the results of the experiment which are sorted by *makespan* for better readability. The plot shows the effect on the planning of an unsorted set of agents with no memory and also on the best performing algorithm regarding *makespan*, OW sorting with M_1 memory. It can be seen that the assumption was incorrect, as the difference between δ and ε metrics is insignificant, similarly to the undirected graphs results shown in Section 4.3.

4.9 Evaluation conclusion

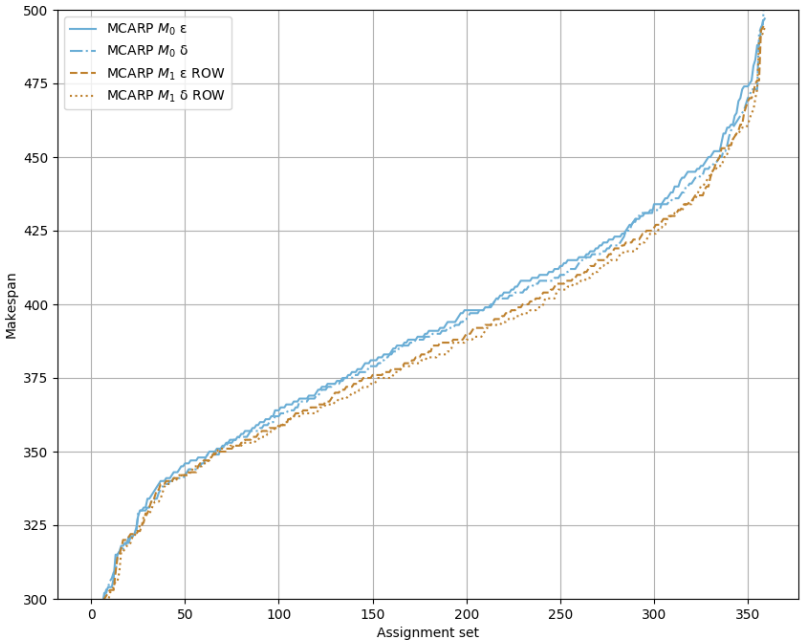
This chapter has shown the main specifics of the various versions of the MCARP algorithm. Two of the variable metrics achieved their original goals. The usage of old plan memory proved to save up *planning time* during replanning and even improve the ability of the algorithms to find a solution. The sorting heuristics proved to be a promising way to the improvement of all qualitative characteristics of the plans. On the other hand, the alternative metric for a heuristic estimate to the goal did not achieve better results than the Euclid distance metric, even on a mostly oriented graph.

Out of the evaluated combinations, the overall best is M_1-δ-IF. Even better results could be, however, achieved by a combination of two setups. Since the significant time saving achieved by the combination of memory and δ metric, two runs of the planning would be affordable.

The best combination could consist of the best performing setup for *makespan* and *length*, M_1-δ-OW, and the least failing version, M_2-δ-LF, which would attempt to find any suitable solution in case of failure of the first. The combined *planning time* could still be held below half of the value achieved by the original CARP.



(a) : Oriented graph of a warehouse map for 20 robots.



(b) : *Makespan* comparison between δ and ε metrics on warehouse map.

Figure 4.9: Warehouse map experiment.

Chapter 5

Fleet Management System

With the growing market of e-commerce and the resulting demands on logistics ([19]), there are various designs of automated warehouses already in operation in the world. The most widely spread automation implementations are [20]:

- Autonomous Ground Vehicles (AGVs)
- Autonomous Mobile Robots (AMRs)
- Automated Storage and Retrieval Systems (AS/RS)
- Aerial drones

The AGVs, AMRs, and AS/RS solutions use some kind of mobile robots equipped with sensors and a specific ability to transport and manipulate the stored merchandise. The movement of AGVs is limited to a given preset route, and they mostly use magnetic tracks, tags, or other external localization systems. AMRs, on the other hand, are far more autonomous and able to plan their tasks and routes through the warehouse. The AS/RS are complete solutions of the warehouses, which use AGVs to move whole storages around the warehouse. This solution will be further addressed in this chapter. The last-mentioned aerial drones are usually used to maintain inventory and monitoring of the warehouse. They can quickly reach distant locations while not taking up the space used for deliveries.

The AGV/AMR designs can be based on human pickers using the help of autonomous carts, some use autonomous or partially manually operated

forklifts with pallet detection, or the warehouse can be a completely robot-occupied workspace with forklift-like mobile robots. The usage of such mobile robot solutions proves to be [21]

- Flexible - thanks to easy reprogramming and rescheduling of the robots,
- Highly efficient - due to the possibility of adding more robots to cover an increase in demand, by reducing the labor costs and work time limits,
- Safe - by application of collision-avoidance mechanisms.

The AS/RS automated warehouses that this chapter aims to use AGVs that can drive under the storages, lift them, and move them from their positions to picking stations. The picking stations are operated by human workers or robotic manipulators that take the goods from the storages and assemble the orders for delivery.

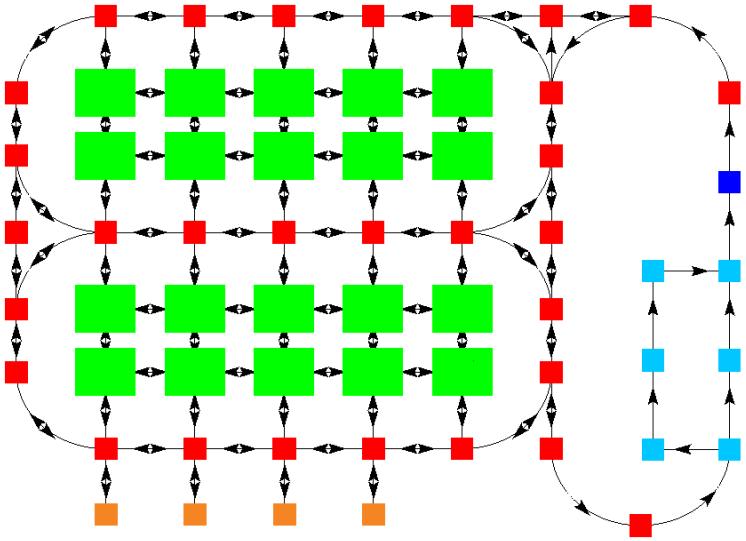


Figure 5.1: AS/RS warehouse schema displayed in the Safelog Fleet Manager Terminal.

A scheme of an AS/RS warehouse layout is shown in Figure 5.1. The layout comprises of road nodes (red) and directed road segments (black lines with arrows). Four special kinds of nodes are distinguished apart from the road nodes. These are the picking stations (one in the scheme, dark blue), nodes of the queue in front of the picking station (light blue), charging stations for the AGVs (orange), and the storage locations. In the schema, the storage locations are occupied by outlines of the storages (green).

For such a solution, a complex management system is necessary to control the AGVs, plan their routes, and ensure they fulfill them correctly. Such a management system, the Fleet Management System (FMS), has been developed as a part of the SafeLog project.

5.1 Description

In the structure of AS/RS warehouse control, the FMS is located between the Warehouse Management System (WMS), which acquires new orders of which picking station requires what goods, and the AGVs themselves. When seen as an I/O system, the input of the FMS is a sequence of jobs from the WMS through a so-called NorthInterface. The jobs say what storage should be delivered to which picking station, while the AGV, which will move it, has to be chosen by the FMS. As an output, the FMS sends commands to AGVs through a SouthInterface while ensuring they fulfill their plans at the correct time to avoid collisions. The components are displayed in Figure 5.2.

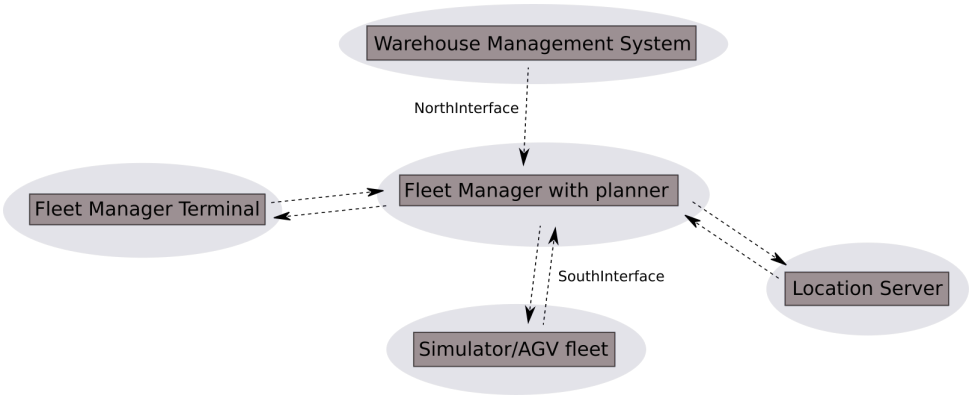


Figure 5.2: Schema of the WMS.

The optional output of the system is a stream of updates of the warehouse state to the Fleet Manager Terminal (FMT), which is a graphical user interface. In a real deployment, the FMS also updates all AGVs' and storages' positions to the Location Server (LS), which is another SafeLog component. Both of these will be described later.

Figure 5.3 shows the components in detail with a schema of their implementation, the transferred data streams, and used communication interfaces. All components and modules will be described in this and in the following Chapters.

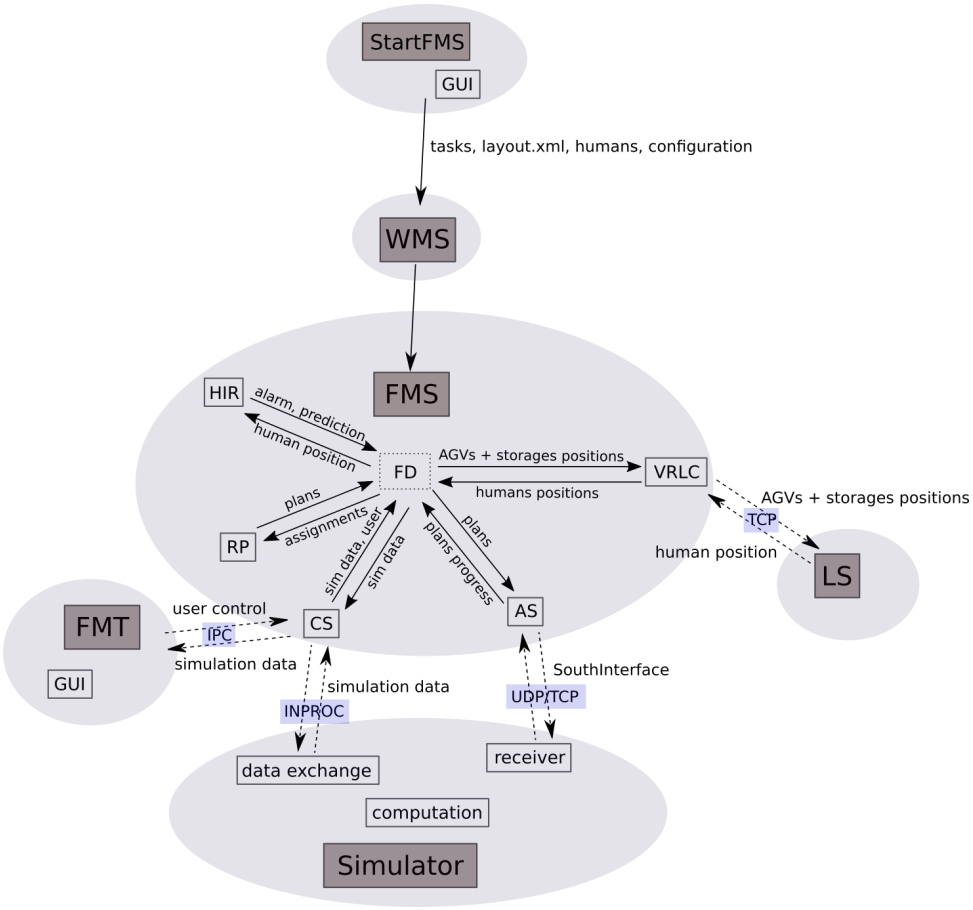


Figure 5.3: Schema of the FMS. A dashed rectangle marks a data object, solid rectangles mark thread. Solid lines mark data access within a program, dashed lines mark data transfer between processes with the communication interface noted in blue.

5.2 FMS modules

The FMS consists of several modules which will be described in this section and are displayed in the center of Figure 5.3. Three of the main modules are implemented as threads of the program, as they continuously work with their assignment. These are the Component and Agent servers and Route Planning. Fleet Data is then a data object equipped with mutexes that ensure safe data access from the threads.

5.2.1 Fleet Data

Fleet Data (FD) serves as a storage of all data used by the FMS. It holds representations of all agents (AGVs as well as humans, which will be introduced later in Chapter 7) with their current states, future plans, and service data. It also maintains data that is on the way from the Simulator to the FMT GUI, both external components that are described in Chapter 6.

5.2.2 Component Server

The Component Server (CS) module is the core of the system. It communicates with all external components of the FMS, reacts to them, and updates the FD. Most importantly, CS extracts current simulation time from the simulator data, keeping the system synchronized. In the case of real-world deployment, another source of clock is applied.

The CS maintains binary communication with the components. For the Simulator that is in-process (INPROC) transport mechanism, for the Fleet Manager Terminal user interface that is inter-process (IPC). Both can be substituted by the secondary TCP network communication. Each component is handled by a separate thread that calls proper static handlers upon reception of new messages.

In case the system is in a simulation mode, the simulation state data is received by CS. The content of it is not meant for the FMS, because the data describes states it would not have the knowledge about in a real deployment, such as exact positions of agents and storages. This data is only stored and then optionally sent to the Fleet Manager Terminal for user visualization. The FMT, if running, delivers information about possible user actions in the GUI. That is stopping or changing the speed of the simulation, new jobs for the WMS, and actions regarding humans. All of these functions will be described in detail in Chapter 6. The simulation related data is redirected to the Simulator, the new jobs are stored in FD for the planner.

Finally, CS invokes the synchronized functions of the AgentServer, which are described in Section 5.2.3.

5.2.3 Agent Server

For communication with the AGVs, an interface called South Interface is used. It defines the format of the messages sent to and from the AGVs, as well as the procedures of the communication. Since the South Interface description is a classified document provided by the real AGVs manufacturer, it will be described only in general. The South Interface distinguishes messages going from the FMS to the AGVs (down) and from them (up). The first group consists of messages providing a new command, a request for status, or an acknowledgment of the received status. The AGVs send only the status messages.

The Agent Server (AS) handles communication with the AGVs (and in the simulation also with humans) w.r.t. the South Interface. Upon reception of a status message, a handler is called and firstly sends an acknowledgment. Then, based on the data in the status message, it determines the AGV status, which leads to the next action. The AGV could be

- responding to a new command,
- it can be announcing the completion of its current command,
- or it could have just been requested to send the status.

If a command was received, AS only acknowledges the reception and erases the command from the queue of commands for this AGV. The queue and command creation, as well as the idle queue, will be described in Section 5.2.4. If the AGV has completed all of its commands, which corresponds to the second and third cases, the AGV is added to the idle queue until the time of the next command, in the case that there is such. Otherwise, the AGV has completed its plan, and the planner will plan its next phase.

From the status message sent from the AGV, the FMS also gets the knowledge of the current position of that AGV. Since the AGVs are not tracked externally, they localize themselves using ground markers and add this information as a part of the status. They also inform about current deviation from the node, battery status, etc.

The AS also provides several time-synchronized functions invoked by CS. That is sending new command messages to all AGVs that ran out of their idle time - they are popped from the queue. To ensure reliable communication

over the UDP or TCP network interface, AS also maintains a memory of the last broadcasted message to each AGV, and if no response is received (if expected) until a set timeout, the message is re-sent.

Furthermore, to ensure an efficient workflow of the warehouse, AS attempts to lower the unnecessary waiting times of the agents. That is done by checking the next idle time for each AGV (time until the next command is to be executed) within a defined range from the target picking station queue. If the time exceeds a defined threshold, AS asks RP to attempt a re-plan of the AGV in order to achieve a shorter waiting and delay, possibly. This possible delay is caused by the planner not knowing the future state of the picking station queue at the time it plans an AGV path. It, therefore, adds a safety delay at the end of the plan to ensure avoiding collision with the AGV that might be occupying the queue start. But once the AGV is within the range of the queue and it will not be overtaken by other AGVs, more efficient timing can be reached for it.

5.2.4 Route Planning

A new job from the WMS is directed to the Route Planning (RP) module. That consists of a state machine handling new jobs, as well as new phases of the ongoing jobs. RP is equipped with a planner that holds the necessary occupancy table and a resource graph describing the warehouse map. The planner was described in Chapter 3 in detail. The AGV that will be given a new task is chosen based on its distance from the job start - the target storage. A route plan is then produced for it and passed to the system. For convenience, the process of fulfilling a task is split into five parts, namely:

1. To a storage
2. To a picking station queue
3. Movement within a queue
4. Returning to a storage
5. To a next storage/back to a charging station

When an AGV is in one of the first four states, it is considered *busy*, and each time it completes one of these phases, the planner plans the next one for it. Once the AGV returns the rack, it turns $free$, and the planner leads it

back to its charging station, or if there is another job, and this is the closest AGV, it is assigned to it.

The FMS then maintains a set of timed plans (trajectories) for the AGVs. The RP produces the plans in the South Interface format. That means the plans are split to single actions, such as proceed one node forward, turn, pick up storage, etc. These timed command messages can be serialized and sent directly to the AGVs. To maintain collision-free execution, the AGVs should execute each step of the plan at a time given by the planner. Therefore, AGVs are moved to the idle queue until the time of their next movement - command. The idle queue sorts the AGVs waiting for their commands by the time they will go on with their plan so that checking and then commanding of these idle AGVs is faster. After the plan is produced, a request for the status is sent to the corresponding AGV. This will make it report itself to AS, which will then send it the commands.

5.3 Customization

The FMS provides several customizable options. Firstly, multiple serialization formats and communication protocols and mechanisms are implemented.

Apart from the predefined messages in the South Interface, the FMS primarily exchanges binary messages with its components. The other option is the significantly larger JSON format. To be compatible with the real AGVs, the FMS uses the UDP network protocol for the South Interface. The TCP network protocol can be used optionally in simulation. For communication with the components (Fleet Manager Terminal and Simulator), the FMS uses shared memory transport mechanisms from the `nanomsg C++` library[1]. The Simulator runs in the same process as the FMS, so the in-process (INPROC) mechanism is used. The Fleet Manager Terminal is a separate program and communicates with the inter-process (IPC) mechanism. Both components can optionally communicate over TCP network protocol.

The other main customizable option is whether the FMS runs with a simulated or a real warehouse. Also, the introduction of humans is available with the possibility to assign their tasks by hand in Fleet Manager Terminal or to load a prepared assignment file. The user can also select whether to enable distractions of humans - a simulated free-willing behavior of humans sometimes not following given paths. The features regarding humans will be

[1] For more information about the library, see https://nanomsg.org.

described in Chapter 7. Finally, it is possible to launch an online plotting tool that displays statistics and graphs about the warehouse operation. This tool will be described in Chapter Chapter 6 along with the FMT.

Chapter 6

FMS testing tools

Several tools were developed to test the function of the implemented FMS. The main idea was to avoid the need for testing with a real warehouse demonstrator and real robots, which would be costly and ineffective. Therefore, the FMS was equipped with two external components, the Simulator and the Fleet Manager Terminal. These tools allow us to simulate, display, and control the process of an automated warehouse, providing useful feedback for debugging and other improvements of the system before the real-world deployment.

6.1 Simulator

The Simulator imitates the actions performed in the warehouse. It maintains communication with the Agent Server, acting as the AGVs, and performs the received commands by calculating the positions of AGVs and storages in time. Its detailed scheme is shown in Figure 6.1.

The Simulator holds a representation of all AGVs. The separate modules in this set can be easily used as the core for a real AGV control program, as described in Section 8.2. However, in the Simulator, all of them are just data objects without being split to separate threads. This resulting approach was selected for a better simulation performance when compared to using a separate thread for each AGV.

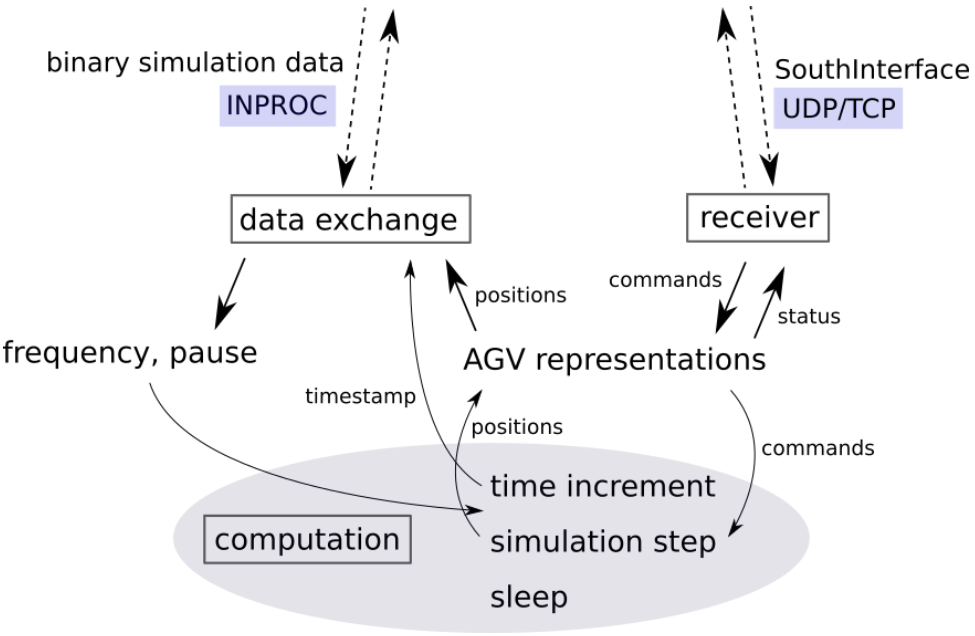


Figure 6.1: Schema of the Simulator.

Processing of all AGVs is therefore done in a pair of threads. The receiver imitates the AGV communication; it receives messages for all the AGVs and calls the corresponding handlers in them. The AGV object than either notes a new command and confirms reception back to FMS, or it only sends a status message. All is according to the described South Interface routines. A mechanism for resending messages after a defined timeout was implemented here the same way as in the FMS. AGV objects are given the current simulation time and record the time of the last broadcast. The computation thread then triggers periodic checks of the time since the broadcast, and after a set timeout, the status message is sent again.

The second thread running in the Simulator performs the computation itself. In a routine shown at the bottom of Figure 6.1, it firstly increases the timestamp of the simulation, the simulation time. Then it performs the response timeout check as described above, and finally, it performs the simulation step by evaluating the state of all AGVs and takes proper action. An AGV can either have no commands - then it is skipped - or have a command for execution. In that case, a handler is called for the particular command type to compute the current AGV's position. If an AGV has completed its command, the status message is sent right after. After all of the AGVs are evaluated, the sleep time is computed based on the duration of the simulation step to maintain a constant frequency.

The last thread in the Simulator serves for data exchange. It communicates with the FMS by requesting a possible pause or speed change and sending

the current positions of all AGVs and storages as a reply. These serialized packages of data are redirected through the CS to the Fleet Manager Terminal.

6.2 FMT

The Fleet Manager Terminal is a graphical user interface (GUI) that displays the process in an automated warehouse, as shown in Figure 6.2. It is not dependent on the usage of the Simulator and is able to display the activity also in a real warehouse or demonstrator operated by the FMS. It can also be used to limitedly operate the warehouse as a substitute of the WMS, or control the simulation.

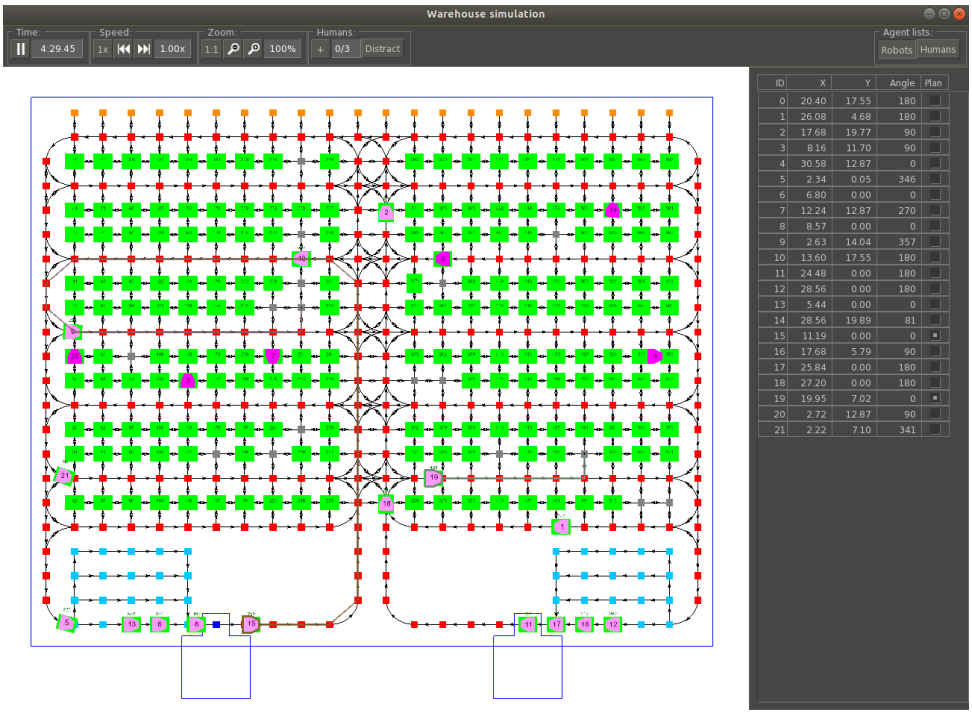


Figure 6.2: Fleet Manager Terminal GUI.

The primary function of the FMT is the visualization of the current execution state of the warehouse. That is based on data that the communication thread of the FMT queries from the CS. Based on the received data, the FMT window content is updated in a defined refresh rate.

The opposite direction of communication serves to deliver information about user actions in the GUI. The user can:

- Pause the simulation or change its speed
- Operate the visualization - zoom and move image
- Display AGVs and their exact positions
- Simulate the WMS - add an assignment for the AGVs by clicking on a chosen storage and then on a target picking station
- Display current routes of the AGVs by clicking on them

6.3 Statistics

An additional GUI feature is a plotter program. It is an optional tool implemented in Python that communicates with the FMS over an IPC interface and receives JSON messages noting new assignments and performed deliveries for all AGVs. From this data, the program counts the new fulfilled assignments in time and displays them in a plot, online. Along with that, it computes and displays the developing time needed for each delivery.

The main purpose of this tool, however, is to show how the warehouse process is affected by the presence of humans and their behavior. This will be described in the following Chapter in detail.

Chapter 7

Human worker in an automated warehouse

The main idea which this thesis is based on is the cooperation of agents with different priorities. Specifically, it can be a human worker and mobile robots working in the same area, while the human must not be endangered by the AGVs. With this multi-priority setup and even the idea of the human entering the area while the robots already fulfill their tasks, the following situation arises. That is, the robots have their plans and mostly do not have to be affected by the moving human. However, they should evade the area the human will occupy.

7.1 Problem definition

Most of the current solutions of automated warehouses enable the possibility for a human worker to enter the area, e.g., to perform a maintenance task. However, this causes the whole warehouse or its significant part to shut down completely since the industrial robots are not equipped with certified safety measures and sensors to avoid causing danger to the human. Such a situation, causing an indefinite delay in delivery completion, is highly ineffective for the whole warehouse process. Enabling most of the AGVs to continue their work while only changing their paths or slowing them down, with a very limited number of AGVs that would need to be stopped, means a large improvement and cost-saving.

From the safety point of view, it needs to be ensured that the AGVs maintain a safe distance from the human, slow down when closer to him, and in case of reaching a close safety zone, they stop their movement completely. From the view of the planner, the human is a highly prioritized agent, as described in Chapter 3, since he cannot be requested to wait for a long time or use a seemingly longer path - human free will should be expected and respected already in planning. The second level of this human anticipation is that even when the path given to the worker seems the easiest, the system cannot fully expect him to follow it. Either the human can exceed the expected mean speed, or he can change the planned direction completely. All these requirements should be taken to account when designing a system that would successfully provide an environment for human-mobile-robot cooperation that is safe and efficient.

7.2 Human safety

Figure 7.1: Warehouse ilustration with safety levels. *Image source* : *Safelog*

Several safety mechanisms are applied in the Safelog project to meet the described requirements. They can be split into hardware and software ones. The hardware mechanisms, developed by other Safelog partners, are:

- Safety Vest
- Augmented Reality (AR) glasses

They provide the human worker the information about his surroundings, including the path he should follow or a notification that an AGV is nearby, and in the opposite way, inform the system about the current human position. Through a Location Server (LS), this data is transferred to and from the FMS. The software mechanisms are the following:

- Safety Levels
- Human-Aware Route Planning
- Human Intention Recognition

The safety levels are three. The widest level is the area of the whole warehouse where the AGVs are planned by the FMS and can move at full speed. When an AGV enters the middle level, the human is informed about its presence, and the AGV slows down. Finally, the closest level acts as a simulated electromagnetic pulse that turns off all AGVs that enter it. The function of planning and intention recognition modules will be described in the following sections.

7.3 Human in FMS

For communication with the humans, i.e., sending them their plans, the FMS distinguishes in its modes. If it is running as a simulation, the simulated humans are handled the same way as the AGVs, which means by the AgentServer. For that, a special type of South Interface message was designed. This command message contains the whole path which it provides to the human directly. The Simulator then moves the human object along the given path with a set velocity, without communicating on each node as for the AGVs. In the case of the real warehouse deployment, the path is handed over to the Location Server, also at once. It is then sent to the AR glasses, which display it to the worker to lead him through the area.

The opposite direction of information, data about the human's position, is also processed differently according to the mode. In the simulation, the human is again grouped with other agents, and its position is extracted from the simulation data. It is the only entity the FMS gets the exact position of. In a real deployment, the safety vest uses a camera to localize the human using the position markers placed in the warehouse. The position data is then sent through the Location Server, which provides it to the FMS.

The FMS Simulator allows two ways of control in the simulation. Firstly, a predefined file with timed assignments for the human or humans can be loaded by the system, which will then execute the tasks autonomously. The FMT, if running, displays messages when the human is assigned, whether the planning was successful, and when his movement will start. Alternatively, the user can operate the human through the FMT during the simulation. A limited number of humans can be added to the process, and upon clicking on some of them, the user can set their new goal. A message shows to inform the user whether the planning was successful and when the movement will start, which can be seen in Figure 7.2.

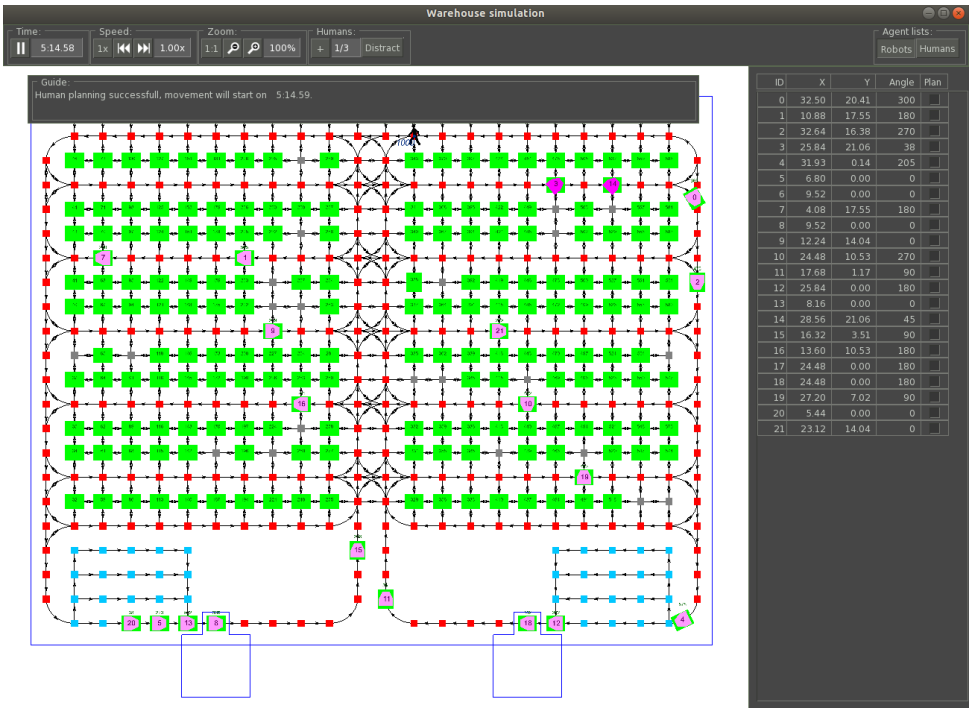


Figure 7.2: Human in FMT GUI.

7.3.1 Human-Aware Route Planning

A special part of the RP handles human presence in the warehouse. Firstly, once a human target is known, it attempts to plan the human towards the goal. For that, all AGVs are stopped, and the highest priority (empty occupancy except for static robots' positions) is given to the human. It is possible that the planning fails - the stopped AGVs still represent an obstacle, and a valid path might not exist. In that case, the planner attempts to move the AGVs away from the desired human plan and retries the planning. If one of the attempts is successful, the plan for the human is either set to be sent to the simulated human at a given start time, or the planned path is updated to

the LS, again with the expected start time. After that, AGVs are planned towards their goals with respect to the human path and the safety proximity. At this point, the old plan knowledge speeds up the replanning, as described in Chapter 3.

The RP also checks, based on the current AGVs' status and human position, whether the safety radius is violated and takes the corresponding action. That is stopping the robot and alerting the human or just planning the robot with a lower speed in the area. The emergency stop is doubled, as it can be triggered either by the RP based on the computed distance or by a radio measuring module mounted on the AGVs.

7.3.2 Virtual Reality Location Client

The Virtual Reality Location Client (VRLC) is an optionally launched component of the FMS that handles communication with the Location Server (LS) in a real warehouse deployment. This communication over TCP socket provides the current position of a localized human to the FMS. The FMS, on the other hand, updates the positions of AGVs and storages to the LS. Once it connects to LS, the FMS also informs the server about the positions of all node markers and storages in the warehouse layout. Finally, as was noted above, the FMS updates the LS with new plans for humans.

7.3.3 Human Intention Recognition

The purpose of the Human Intention Recognition (HIR) tool is to check that a human is fulfilling a given plan and, if not, provide a prediction of a possible new path followed by the human. This allows the planner to take actions for the AGVs - update their plans - prior to a possible interrupt caused by a proximity violation.

This part of the FMS, developed mainly by colleagues from University of Zagreb, Faculty of Electrical Engineering and Computing (UNIZG-FER), takes the plans for humans and their actual positions, and in the first level, it checks whether the human is within tolerance from the expected position in time. The second level comes to work at the point when the plan is violated. A simpler situation is when the human stops - in that case, the HIR provides the planner with that information, and the planner adjusts the AGVs paths so that they will avoid the area, or, in case they cannot, stops some of them.

A more interesting situation arises in the case the human changes his direction completely. At that point, the HIR reacts by attempting to detect a possible new target of the human from a set of possible points of interest through the warehouse. It then plans the expected paths to a set of these and provides the planner with the most probable one. The planner takes it as the new plan for the human and again adjusts the routes for all AGVs.

The cooperation in implementation and testing of the HIR module led to a successful conference journal publication [4].

7.3.4 Simulated distractions

To fully test the function of the HIR module, the FMS is able to simulate the human violating the given plan. This can be achieved either by providing a file with the distraction assignments, similar to the one with standard assignments for humans or by the humans being controlled through the FMT GUI. To distinguish, whether the user assigns a standard job, or a distraction, this mode is set by a dedicated button at the top of the FMT screenshot in Figure 6.2.

The so-called distraction job is then passed to the RP, which plans it while ignoring all AGVs. On the contrary to a standard job, however, it does not store the occupancy information or the job itself. It just sends the modified command to the simulated human. The Simulator then switches to following this new plan, and it is the task for the HIR module to detect the deviation, once the human leaves the expected old path.

7.3.5 Plotter tool

Added human assignments, as well as distractions, can be displayed in the plotter tool described in Section 6.3. With the development of completed deliveries and their duration, this provides a useful insight into how human activity in the warehouse affects its production, fully online. The plotter also displays at which time the HIR reacted.

Chapter 8

FMS performance and deployment

The function of the FMS and its components has been evaluated in three areas of application. Firstly, the Simulator setup was extended with a statistical tool described in Sections 6.3 and 7.3.5 that allows evaluating the performance of the system under different conditions. The FMS was also applied in a laboratory demonstrator to operate a fleet of TurtleBot2 robots simulating the AGVs in a real warehouse. Finally, the FMS is currently integrated into a real warehouse demonstrator to operate the Safelog designed warehouse.

8.1 Simulated performance

Although the main benefit of the Simulator is the possibility to develop and debug the FMS without the need for any costly hardware, it can be used for the final system evaluation as well. With the usage of the plotter tool, the performance of the automated warehouse can be compared based on different scenarios. The FMS can work with various warehouse layouts, fleet sizes, and the presence and behavior of humans can differ as well.

Figure 8.2 presents the results of some of these scenarios on the map shown in Section 4.8. The expected number of AGVs for this map is 22, and the results of this standard process are in plot **(a)**. The AGVs start in their charging stations, which causes the slow development of delivery durations in the beginning. After the first delivery wave (one shade of green), the

delivery times stabilize around 150 s since the AGVs already move between the storages, and the closest is selected for each task.

(a) : Warehouse with 21 AGVs.

(b) : Warehouse with 11 AGVs.

(c) : Human worker moves inside warehouse.

(d) : Humans not following orders, HIR switched off.

Figure 8.2: Plotter tool showing the growing number of deliveries during the time and their duration development. The Y-axis displays both time for delivery in seconds and number of completed deliveries. The $n-th$ shade of green marks the $n-th$ delivery completed by each AGV. In the Human Actions plots, the marks denote that at that time, a new job was assigned to any human in the warehouse.

Plot **(b)** shows a similar process with only 11 AGVs. The average time for delivery is lower, around 100, since the robots do not have to wait before the picking stations. The overall speed is lower, however, since there are fewer AGVs to meet the demand.

In the plot **(c)** it can be seen how the process can be affected by a human moving in the warehouse. Note that since the first appearance, the human is still present in the warehouse and therefore serves as an additional obstacle. In the area where the AGVs move mostly on one-way routes, the influence depending on the exact location can be high. The dependence on the human

location can be seen later in the process, where the human starts to slow down the deliveries more significantly.

The impact of an arbitrarily moving human can be seen in plot **(d)**. Since the HIR was turned off in that case, the human quickly caused major delays and completely disabled the warehouse afterward. Since the AGVs are not designed to react to their surroundings, it is the FMS that needs to be informed about the human's location to react in time.

8.2 Laboratory demonstrator

The laboratory demonstrator is shown in Figure 8.3 has been developed by a group of students, including the author, as a team project. This comprised of

- setting the real environment,
- designing and implementing ROS modules to control the robots, and
- assembling the components and designing a way to repeatedly launch the demonstration.

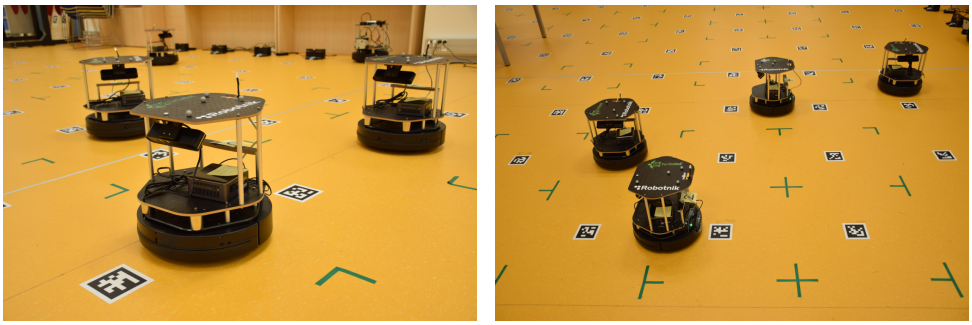


Figure 8.3: Automated warehouse laboratory demonstrator, Intelligent and Mobile Robotics Group, CIIRC CTU.

The center of this demonstrator is the nearly unchanged FMS. In order to avoid any collisions caused by possible delays, it was only equipped with an additional node blocking tool. This reactive node blocking ensures robots access only nodes, which are *free* at the time of execution. Such delays are possible due to the limited precision of the hardware - correction of, e.g., orientation error can take arbitrary time.

The Simulator core is used as well, but only as of the source of the clock for the FMS. An optional function is a connection to the Vicon motion-capture system over UDP protocol and using its data as an external localization of the robots. This data is then packed the same way as in simulation and used for continuous visualization in FMT. It is not used for the FMS itself.

The robots localize themselves using a simplified version of the actual Automated Storage and Retrieval System (AS/RS) approach. The laboratory is equipped with a grid of unique markers - AprilTags [22]. Based on the South Interface, the robots do not get the information about the position they need to reach, and they also do not localize themselves absolutely. Instead, they know the ID of the marked node they need to reach, and that is supposed to be in their route ordered by the FMS. Once the robot has the visual contact with the target marker, it navigates on top of it, while storing its relative orientation. Before reporting command completion, it corrects the orientation error based on the target orientation of the current command.

The robots use standard PCs equipped with ROS [23]. The robot controlling program is split into localization, motion control and communication, similarly as in the Simulator. Also, the base of the communication and data maintaining part is taken from the Simulator as proposed in Chapter 6.1. Retrieving of the AprilTag information from the camera is based on the `apriltag_ros package` [1]. The motion control is based on a proportional controller for speed

$$u_{speed}(t) = K_P e_{speed}(t) \tag{8.1}$$

and a PI controller for turning

$$u_{steer}(t) = K_P e_{steer}(t) + K_I \int_0^t e_{steer}(t')dt'. \tag{8.2}$$

The achieved precision of motion is within 5 cm position error and 0.02 rad orientation error.

8.3 Real warehouse demonstrator

The main purpose of the FMS is its deployment in control of a real automated warehouse along with other Safelog components described in Chapters 5 and 7. This deployment is currently in progress and brings in several other challenges and necessary adjustments of the FMS so that it will be able to operate the real AGVs running their black box firmware. The FMS is also

[1] For more information about the package, see http://wiki.ros.org/apriltag_ros.

adjusted to work with standard industrial safety routines, such as external emergency stop, photoelectric sensor triggered emergency, etc. From the nature of the warehouse operation process, the FMS can no longer be a once ran application, but it should be adaptable to the current state of the warehouse, for example, AGV and storage positions.

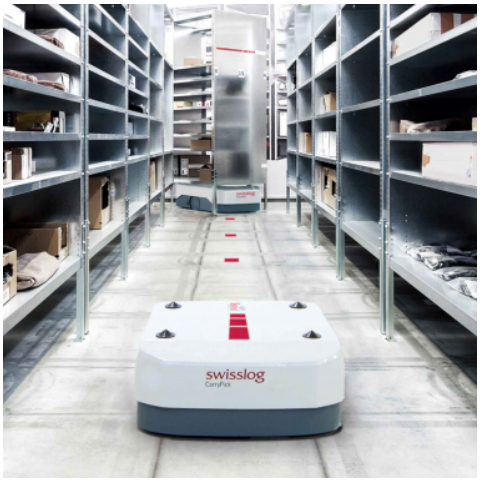


Figure 8.4: Swisslog Carrypick AGV used in the Safelog project.
Image source : https : //www.swisslog.com/

Chapter 9

Conclusions

This thesis deals with the topic of warehouse automation from two different views. In the first half, an approach for the solution of a specific instance of the multi-agent path-finding problem was proposed. The motivation for this came from the development of a real autonomous warehouse management system, which was presented in the second half.

In the structure of the automated warehouse layout where humans can enter the area used by AGVs, replanning of these robots becomes a regular task, and its optimization is in place. From the nature of the situation - a fleet of AGVs that already have collision-free plans - the multi-agent path-finding algorithm-based planner module can make use of the knowledge it already has and computation it already performed.

Several techniques were therefore proposed and combined in Chapter 3. These build on the Context-Aware Route Planning (CARP) algorithm and, based on the usage of memory, are jointly named MCARP. The application of memory, a more precise heuristic estimate of cost-to-goal for states, and heuristic sorting of agents have shown promising results in the experiments presented in Chapter 4. Not all expectations were met since, even in a mostly oriented graph, the precise estimate to the goal did not outperform the Euclidean distance. The rest of the assumptions were, however, confirmed. The usage of memory significantly reduces the time needed for replanning and can also ensure lower *failure rate*. The more precise heuristic cost computation based on the exact travel time to a vertex also reduces the number of expanded states. And finally, two of the proposed techniques for

the sorting of agents served for the best performing planners among those compared.

These sorting heuristics, namely sorting by influence - distance - between agents in the original set of plans and by their overall waiting time during these, may serve as an inspiration for more general usage. For example, sorting of agents can be performed even during the original planning process and has proved to have a major impact on the quality of the plans [1] [12]. Further research is also necessary for the leading idea of the MCARP algorithm, application of the distance from the original path to the A* heuristic. As was already mentioned, this causes the heuristic to lose the admissibility property. In the specific case assessed in this work, this should not affect the quality of the results. Still, the properties of the heuristic should be evaluated to, for example, find suitable bounds to the sub-optimality [15]. The use in the original MAPF solving is then also possible even for memory usage, for instance, in the combination of pre-computed paths in the known environment.

The inspiration for the usage of old plans to speed up replanning came from the D*Lite algorithm. Its modifications for MAPF were already proposed, but are usually highly limited based on the data structure [24]. The first issue arises from the fact that D*Lite builds the paths from the goal to start. Then, in case of a change in the path, it rebuilds only the segment around the new obstacle and keeps the rest of the path. In multi-agent path-finding, the time domain has to be taken into account, and a path segment reached later than originally planned may no longer be accessible without collision. Therefore, both of these properties need reworking in order to comply to that. Instead of these modifications, this thesis aims to assess the problem from the other side - modifying a MAPF solver to incorporate the previously planned path. As it showed as a promising direction, the future work regarding this topic can build on it.

The second half of this thesis presents the FMS and its components. Although presented here as a whole, the project was developed by multiple people in time. The contribution of the author was especially in finalizing the components and processes to work fluently. In the first phase, this consisted of reworking the communication between Agent Server and the simulated AGVs, fixing the command generation process, and debugging several related issues in both Simulator and FMS. Performance optimization was also performed by reduction of the threads in the Simulator and replacing serialization and communication protocols with faster solutions based on binary messages and shared memory, respectively. A whole new area was the introduction of humans, the design of their control, rules, and movement execution.

The development of the FMS is still in progress, especially concerning the human distractions simulation and FMS integration in the real-world warehouse demonstrator. At this moment, however, it is already a useful tool for the evaluation of automated warehouses application and their extension with human workers. Additionally, it is used for full control over a laboratory demonstrator of the automated warehouse.

Three conference papers were published with a link to the work presented in this thesis. The methods *DG* and *GG*, introducing the influence between agents' plans and using it for their sorting, were proposed in a paper for the 21st International Conference on Intelligent Transportation Systems (ITSC 2018) [3]. The joint work with the researchers from the University of Zagreb, Faculty of Electrical Engineering and Computing, on the intention recognition approach with the application in the presented FMS was accepted for the 28th Mediterranean Conference on Control and Automation (MED 2020) [4]. Finally, the laboratory demonstrator originally designed for FMS was modified to perform experiments for a planner using the AA-SIPP planning algorithm [25]. In cooperation with the researchers from the Federal Research Center for Computer Science and Control of Russian Academy of Sciences, the application of the planning algorithm in a warehouse-like pick-up and delivery problem was accepted for the 17th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2020) [26].

Appendix A

Bibliography

[1] T. Rybecky. Planning for a team of cooperating mobile robots. *CTU in Prague, Bachelor thesis*, 2018.

[2] Adriaan W. ter Mors, Cees Witteveen, Jonne Zutt, and Fernando A. Kuipers. Context-aware route planning. *Delft University of Technology, The Netherlands*, 2011.

[3] J. Hvezda, T. Rybecky, M. Kulich, and L. Preucil. Context-aware route planning for automated warehouses. *21st IEEE International Conference on Intelligent Transportation Systems (ITSC 2018)*, 2018.

[4] Tomislav Petkovic, Jakub Hvezda, Tomas Rybecky, Ivan Markovic, Miroslav Kulich, Libor Preucil, and Ivan Petrovic. Human intention recognition for human aware planning in integrated warehouse systems. *28th Mediterranean Conference on Control and Automation (MED 2020)*, 2020.

[5] Roni Stern. *Multi-Agent Path Finding – An Overview*, pages 96–115. 10 2019.

[6] Sharon G., Stern R., Goldenberg M., and Felner A. *The increasing cost tree search for optimal multi-agent pathfinding.*, page 470–495. 2013.

[7] Sharon G., Stern R., Felner A., and Sturtevant N.R. *Conflict-based search for opti-mal multi-agent pathfinding.*, page 40–66. 2015.

[8] T.S. Standley. Finding optimal solutions to cooperative pathfinding problems. page 173–178, 2010.

[9] Wagner G. and Choset H. *Subdimensional expansion for multirobot path planning.*, page 1–24. 2015.

[10] M. Phillips and M. Likhachev. Sipp: Safe interval path planning for dynamic environments. *in Proceedings - IEEE International Conference on Robotics and Automation (2011)*, page 5628–5635, 2011.

[11] Yu J. and LaValle S. M. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. page 1163–1177, 2016.

[12] Adriaan W. ter Mors. Evaluating heuristics for prioritizing context-aware route planning agents. *Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, The Netherlands*, 4 2011.

[13] A. Andreychuk and K. Yakovlev. Two techniques that enhance the performance of multi-robotprioritized path planning. *AAMAS 2018, Stockholm, Sweden*, 2018.

[14] Dave Ferguson, Maxim Likhachev, and Anthony Stentz. A guide to heuristic-based path planning. *School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA*.

[15] Likhachev Maxim, Gordon Geoffrey, and Thrun Sebastian. Ara*: Anytime a* with provable bounds on sub-optimality. *in Proceedings - Advances in Neural Information Processing Systems (2003)*, 16, 01 2003.

[16] V. Narayanan, M. Phillips, and M. Likhachev. Anytime safe interval path planning for dynamic environments. *in Proceedings - IEEE/RSJ International Conference on Intelligent Robots and Systems (2012)*, pages 4708–4715, 2012.

[17] Koenig Sven and Likhachev Maxim. D*lite. pages 476–483, 01 2002.

[18] Ingerman Peter Z. *Algorithm 141: Path Matrix.* 1962.

[19] W. Matthews and S. Dawson. The shed of the future e-commerce: its impact on warehouses. *Online, available: http://www2.deloitte.com/uk/en/pages/real-estate/articles/shed-of-the-future.html*, 2014.

[20] Ruthie Bowles. Warehouse robotics: Everything you need to know in 2019. *Online, available: https://www.logiwa.com/blog/warehouse-robotics*, 2020.

[21] Custodio Larissa and Machado Ricardo Luiz. Flexible automated warehouse: a literature review and an innovative framework. *The International Journal of Advanced Manufacturing Technology*, 106:1–26, 01 2020.

[22] Wang J. and Olson E. Apriltag 2: Efficient and robust fiducial detection. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2016).*, 2016.

[23] Quigley M., Conley K., Gerkey B. P., Faust J., Foote T., Leibs J., Wheeler R., and Ng A. Y. Ros: an open-source robot operating system. *ICRA Workshop on Open Source Software*, 2009.

[24] Alsulaiman Mansour, Emaduddin Muhammad, Hedjar Ramdane, Mattar Ebrahim, and Al mutib Khalid. D* lite based real-time multi-agent path planning in dynamic environments. *International Journal of Engineering Research and Applications*, 2:1414–1419, 2012.

[25] Konstantin S. Yakovlev and Anton Andreychuk. Any-angle pathfinding for multiple agents based on SIPP algorithm. *Conference on Automated Planning and Scheduling (ICAPS 2017)*, page 586–593, 2017.

[26] A. Andreychuk T. Rybecky M. Kulich, K. Yakovlev. On the application of prioritized safe-interval path planning with kinematic constraints to the single-shot pickup and delivery problem. *17th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2020)*, 2020.

Appendix B

Enclosed CD contents

The root directory of the enclosed CD contains the following items

- thesis.pdf: This thesis
- figures: A directory containing all presented figures and images
- source: A C++ project containing the implementation of all mentioned algorithms
- tex: A LaTeXproject of this thesis
- stats: Python script for the statistics processing
- readme.txt: Instructions to running the C++ project and the statistics generation, list of CD contents

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Rybecký Tomáš** Personal ID number: **457220**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

Branch of study: **Cybernetics and Robotics**

II. Master's thesis details

Master's thesis title in English:

Trajectory planning for a heterogeneous team in an automated warehouse

Master's thesis title in Czech:

Plánování trajektorie pro heterogenní tým v automatizovaném skladu

Guidelines:

1. Get acquainted with current approaches to collision-free trajectory planning for a team of cooperating agents.
2. Design and implement a planning algorithm for a team of robots operating in an automated warehouse. Focus on the case when replanning of already generated trajectories is needed.
3. Design and implement a control system for a fleet of autonomous vehicles in an automated warehouse.
4. Realize an environment for testing and evaluation of the developed system.
5. Extend the control system with the possibility of planning for a human.
6. Evaluate experimentally properties of the implemented control system. Describe and discuss obtained results.

Bibliography / sources:

[1] S. Koenig and M. Likhachev, "D* Lite," in Proceedings of the National Conference on Artificial Intelligence, 2002, pp. 476–483.
[2] J. Hvězda, T. Rybecký, M. Kulich, L. Přeučil. Context-Aware Route Planning for Automated Warehouses. In Proceedings 21st International Conference on Intelligent Transportation Systems (ITSC). IEEE Intelligent Transportation, 2955–2960, Maui, USA, November 2018.
[3] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," The International Journal of Robotics Research, vol. 32, no. 12, pp. 1495–1512, Oct. 2013.
[4] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," Artificial Intelligence, vol. 172, no. 14, pp. 1613–1643, 2008.
[5] V. Narayanan, M. Phillips, and M. Likhachev, "Anytime Safe Interval Path Planning for dynamic environments," in IEEE International Conference on Intelligent Robots and Systems, 2012, pp. 4708–4715.
[6] M. Phillips and M. Likhachev, "Planning in domains with cost function dependent actions," in Proceedings of the 4th Annual Symposium on Combinatorial Search, SoCS 2011, 2011, pp. 203–204.
[7] M. Phillips and M. Likhachev, "SIPP: Safe interval path planning for dynamic environments," in Proceedings - IEEE International Conference on Robotics and Automation, 2011, pp. 5628–5635.
[8] D. Ferguson, M. Likhachev, and A. Stentz, "A guide to heuristic-based path planning," Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling (ICAPS), pp. 1–10, 2005.
[9] M. Likhachev, G. Gordon, and S. Thrun, "ARA *: Anytime A * with Provable Bounds on," Science, pp. 767--774, 2014.

Name and workplace of master's thesis supervisor:

RNDr. Miroslav Kulich, Ph.D., Intelligent and Mobile Robotics, CIIRC

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **12.02.2020** Deadline for master's thesis submission: **22.05.2020**

Assignment valid until:
by the end of summer semester 2020/2021

RNDr. Miroslav Kulich, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

