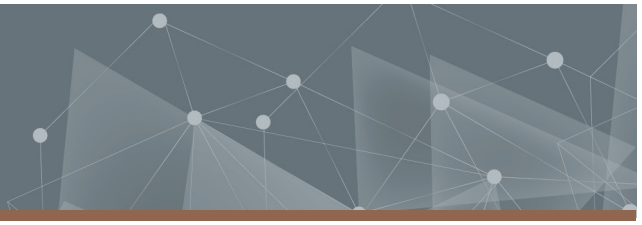
CHALMERS
UNIVERSITY OF TECHNOLOGY



Develop an anti-collision system for a fleet of ATRs in a distributed real time environment using ROS2/DDS

Using a greedy best-first search algorithm to coordinate a fleet of ATRs

Master's thesis in System Control and Mechatronics

David Carlsson & Emil Malmquist

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Develop an anti-collision system for a fleet of ATRs in a distributed real time environment using ROS2/DDS

Using a greedy best-first search algorithm to coordinate a fleet of ATRs

DAVID CARLSSON & EMIL MALMQUIST

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2021

Develop an anti-collision system for a fleet of ATRs in a distributed real time environment using ROS2/DDS
Using a greedy best-first search algorithm to coordinate a fleet of ATRs
DAVID CARLSSON & EMIL MALMQUIST

Supervisor: Per-Lage Götvall, Volvo GTO
Examiner: Emmanuel Dean, Department of Electrical Engineering Engineering

Master's Thesis 2021
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Develop an anti-collision system for a fleet of ATRs in a distributed real time environment using ROS2/DDS
Using a greedy best-first search algorithm to coordinate a fleet of ATRs
DAVID CARLSSON & EMIL MALMQUIST
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Volvo is currently investigating different solutions to make their production facilities able to produce electric, hybrid, diesel, and autonomous trucks on the same assembly line. The issue with several products on the same assembly line is the limited storage space available close to the assembly stations, when producing trucks with different drivetrains more parts are needed and therefore the close vicinity storage space is not enough. A solution to this is to keep the stored parts further away and then transport them to the assembly line stations. With the parts further away a solution to the transport problem is autonomous transport robots, ATRs, moving between the storage and a assembly station. This thesis studies the collision avoidance between ATRs in a large fleet delivering assembly kits. The task was divided into two problems, estimating and solving the collisions. The algorithm estimating these collisions must be fast so that new ATRs can start their journey on time. To find the collisions, the factory floor is stored as a 3D matrix with x, y, time. The trajectory points are then discretized and stored in this matrix, in this way, it is easy to detect nearby trajectories and only the vicinity of the added path needs to be analyzed. Solving a collision is done recursively with a greedy best-first search since it is fast if there is no collision and deviates minimally from the original trajectory otherwise. Another challenge is communication, in this thesis, ROS 2 was used since it can handle more ATRs than ROS 1. The results show that the estimation algorithm finds the possible collisions quickly and solves intersection collisions most of the time between 0.5 ms and 1.5 ms. Head-on collisions take a longer time to solve and are dependent on the distance between the ATRs moving on the same path. Solve time is also affected by maximum/minimum velocity/acceleration of the ATRs and the paths step length. Depending on the time before the collision it may not be able to solve it. The communication works very well and ROS 2 is capable of handling a large fleet of ATRs.

Keywords: Fleet control, Anti-collision, ATR, ROS 2, agile.

Acknowledgements

Firstly we want to thank our examiner Emmanuel Dean, he has helped us a lot through his support and guidance. We couldn't have done it without him since the work he has done to connect all the different parts of the whole system was fundamental to come this far. Another person that have stood by the project and helped us with user input, vision and practicalities is our supervisor Per-Lage Götvall on Volvo GTO, thanks for everything. We also want to acknowledge the help we have gotten the director of our master program Knut Åkesson whose support helped us find literature and aided us when choosing recursive algorithms. Lastly, we want to thank all the other student that contributed to the larger system.

David Carlsson, Gothenburg, june 2021
Emil Malmquist, Gothenburg, june 2021

Contents

List of Figures

List of Tables

1
Introduction

Volvo is starting to scale up the manufacturing of vehicles that have hybrid and electrical powertrains as well as preparing for the manufacturing of driverless vehicles. During this process, it was found that keeping existing factories and assembly lines where advantageous for several reasons. For example, integrating the infrastructure needed for these new products would improve the flexibility to match demand during the transition to more sustainable solutions. It also means that no new factories or assembly lines need to be built. One of the problems when integrating the infrastructure for these new products however is space in the old factories.

1.1 Background

In the current final assembly lines, all assembly details needed for a product at an assembly station are picked into assembly kits in close vicinity to the assembly stations. When integrating the infrastructure for different products on one assembly line the number of different part or assembly details needed increases. The space available close to the assembly stations is then not going to be enough to store the larger number of parts. To solve this problem Volvo started investigating the use of Autonomous Transport Robots (ATR). These robots will go to storage racks further away from the assembly line and load an assembly kit for an assembly station and deliver them to the station at a set time window. The ATRs will use a centralized computer and cameras in the roof of the factory to navigate and achieve the deliveries. This centralized computer will also do fleet control, in other words, make sure each robot can do just in-time deliveries and not collide. This centralized system will reduce costs since each ATR will need fewer sensors compared to decentralized solutions that utilize ATRs with lidars and cameras on-board.

Volvo has together with Chalmers and Örebro university developed an advanced prototype of centralized fleet control for ATRs. This prototype is currently able to control a limited number of ATRs between different goal positions in a lab environment. The prototype uses Robot Operating System 1 (ROS1) [1] for communication between the components, and the prototype ATRs are robotic lawnmowers with trays welded on top. In the current prototype, there exist subsystems that do fleet control. However, since the fleet control is closely connected to when and if parts are delivered just in time, the whole assembly line is dependent on safe and efficient fleet control. This is why further improvements and research within this area can yield important results for the system as a whole.

Volvo wants to develop a new prototype using the same idea as the last prototype, but build it around a communication framework that can handle up-scaling better. Therefore, a collaboration with Chalmers started and the new system will be developed by several other master thesis groups.

1.2 Problems and tasks

The current system works for a limited number of ATRs, however, to have any practical value for Volvo the fleet size of ATRs needs to be increased. The limiting factor of the current system is within the communication, more specifically ROS1. In ROS1, all the communication is managed by the ros core. The core knows all of its connected nodes and topics. When everything is managed by one node scalability will be a problem if all of the communication happens at once. This is one of the problems the thesis is set out to solve.

Furthermore, the just-in-time delivery of assembly kits to the assembly stations is crucial for keeping production up, any stops or delays cause large costs every second. The anti-collision system, therefore, needs to be efficient. Efficient in the sense that the overall cost for the system and its productivity is heavily affected by the efficiency of the anti-collision system. If the anti-collision system is poorly implemented more ATRs would need to be in the fleet to achieve the same productivity as a fleet with a better anti-collision system.

1.3 Aim

This master thesis aims to implement an anti-collision system with a new communication architecture, using ROS2, to increase the safety and scalability of the fleet.

1.4 Limitations

The project will be limited to a small number of ATRs in the physical demonstration. It is a large project with many teams working together, therefore there will be restrictions in what kind of input and output this project can utilize. There will also be restrictions in the refresh rate of the input as well as the output, that comes from hardware limitations.

2
Approach

The first step taken was to define what kind of system that was going to be developed. This was crucial since this definition can through research yield valuable insights into challenges and problems well documented by others for this project. A system of systems (SoS) is a collection of independent systems that work together to solve a task that needs multiple systems to be solved. [2]. However, the systems are still independent in the sense that they can perform a partial task without the other systems operating. Since each master group is developing their part of the larger system that they are supposed to handle a partial task, one could claim that this is indeed an SoS. Mark W. Maier talks about the lack of availability of the different systems, in other words, the challenge to develop a system that needs input from other places that may not be available during the development. Another aspect that is important according to Maier is the communication between the different systems. Maier says "In a collaborative system (a system-of-systems) the intersystem communications is the architecture (in the sense of the organizing structure)." to emphasize just how important the communication architecture is. Having a predefined communication standard that satisfies all sub-systems is key when developing an SoS. Furthermore, there are several challenges when solving the actual anti-collision function of the system. One of the challenges is that the system is operating in real-time. There will also be constraints in update frequency, a high update frequency is needed to handle sudden events. The algorithm developed was required to have around 1 Hz updating frequency according to Volvo. Furthermore how to find collisions, and how to solve them were also areas that need to be solved within the constraints given. To solve the challenge with the time constraint it was important to have an algorithm that takes little time in finding the collisions and then quickly solve the collision that was found. How to deploy the system was also an area that has been looked into since Volvo wants the system to be modular and able to work on different platforms. Lastly, methods on how to evaluate the project during and after development needed to be looked into to measure results.

2.1 Communication

The foundation of the whole system is the communication architecture according to Maier. Therefore our examiner Emanuel Dean together with all the project groups formulated the message structure between the different projects. To solve the communication problems in ROS1, ROS2 is used [3]. ROS2 nodes can discover each other independently [4]. This discovery removes the bottleneck that existed in

the ROS1 when all the messages moved through the core. The information is going both in and out of the fleet control system. The input is updated ATR paths that the fleet control adds to its list of active ATRs. Another input that is needed is a list of the current ATR formations. This information needs to be requested before the ATRs are added to the current active ATRs. There is one output to each ATR which contains a set of the future positions and timestamps for that ATR. It is important to keep track of the actual position and timestamp and check if the ATR is in the correct location. If The ATR is too far away from the assumed location it should stop and any further message will not be sent to that particular ATR. If the ATR is a little bit ahead of the assumed location messages with positions behind it should be removed from the message. All of these types of inputs and outputs can be messaged via ROS2 and in a well-structured way.

2.2 Simulation and modelling

One challenge when developing a system of systems is the independence of each system, or as in this thesis case the independence of each master thesis group. According to Maier, it can be challenging when several independent SoS are being developed simultaneously since inputs and outputs between them are not available at the time of development. Through research, a solution to overcome this challenge was found. According to [5] simulation and modelling can be a useful tool to replace the other systems that will generate or handle output/input. It was therefore decided that the closest independent systems to this project would be simulated during the development phase to verify this system. Testing and development with input/output with all the real systems would be done further into the development process.

2.3 Finding a collision

To fulfill the requirements of a fast update frequency, it was decided to divide the factory floor into smaller areas where a collision might happen. In other words, instead of looking at the entire factory and solve it as one big optimization problem, a faster search is made to find smaller areas where more than one ATR exists at the same time. These areas will be called "Collisions" and are what's going to be used by the solver.

New collisions will only appear when either a trajectory is altered or added. Comparing the new or altered path to all others is an inefficient way of doing it and will scale poorly with more trajectories. If instead a grid is used where all the trajectories are stored then only the vicinity of the added or altered trajectory needs to be checked. With this solution, the size of the grid will be the limiting factor. This method is chosen because of the calculation speed advantages when only the vicinity around the path is necessary to check. If there is a collision this search will give multiple collisions that have to be stitched together, the stitching decides how big the interval will be.

These collisions will be stored so that they can be used to know how long it is until the next collision. This way is chosen because then only the collisions will have to be looped over each update, this is something that is also done by [6]. The collisions are stored as an interval and will be solved on this interval with some guard in the beginning and end so that it is solvable. This means that one collision can be solved every update and if there are multiple collisions to solve within this guard it will be possible. Solving one collision every update means that the amount of times multiple collisions have to be solved in one update will be minimized and thus the computational load is distributed over several loops.

2.4 Solving a collision

To solve the problem the timestamps need to be shifted so that one of the ATRs involved will either slow down or speed up. With the requirement to run at 1 Hz a fast algorithm is needed. It is also faster to solve the problem in discrete-time since there will be fewer steps to reach a solution. One disadvantage of solving in discrete time is that there may be a solution to the problem that can not be found because the solver is not able to compute it. Since time is a stricter requirement it is better to have the solver in discrete time.

The solver may be of many different kinds, it can either be breadth-first or depth-first. Because of the time requirement, the breadth-first is not of interest because it will take too much time on average. The requirement is that the altered path should have the same end time as the path with the collision before it was altered. This constraint is soft, only if possible, and is there to not introduce any new collisions in the system after altering the path. With these requirements, the recursive best-first search algorithm looks to be the best according to our research. This algorithm searches in a depth-first style and decides which step to take by weighing it so that it keeps to the original path as much as possible. Because of this weighting, the solution is chosen as the first solution that reaches the position (x,y). This algorithm will take a little longer because of the weighing, if the solution chooses to slow down then in the next step it will be weighted to take the order of speed up, normal, slow down. Given it needs to slow down several times in a row the algorithm will have to go through several steps of speed up and normal that are not possible. This is still a preferred behavior to have since the solution will contain the smallest amount of change from the original path that still avoids the collision. The algorithm will also need a max solution time because of its time complexity.

2.5 Priority

When a potential collision is detected it is passed to the solver to modify the timestamps in the path and make it collision-free. To do this, the solver needs to know which ATR is supposed to move out of the way and which one can continue on its original path. To solve this, a priority system was implemented. The logistics

behind the priority system was not looked into, the only important part it had for the project was to make sure that one ATR has higher priority than the other. More advanced priority logistics could later be implemented in the coordinator or by the user. Early on in development, a problem with the priority system had become clear. One could call it the "catching up" problem. The problem comes when a lower priority ATR changes speed for some reason, to avoid colliding with a higher prioritized ATR from behind for example. In figure 2.1 one can see a graphical illustration of this. If a higher priority ATR then comes on the same path in front of the lowest priority ATR it will be given the right of way. This will introduce a collision or a unsolvable scenario as can be seen in figure 2.2. In other words the ordering of ATRs when solving "catching up problems" needs to be that the leading ATR in the train get collision free first and then the second ATR and so on, regardless of priority. To solve this, a way to detect if an ATR is "catching up" to another ATR, to then flip priorities. This method of solving the "catching up" problem was not the first that was tested, and it was recognized that it was probably not an optimal solution. Switching priorities is problematic but the cost for doing this is at worst case missed deadlines for higher priority ATRs instead of causing collisions. This is an area of further research.

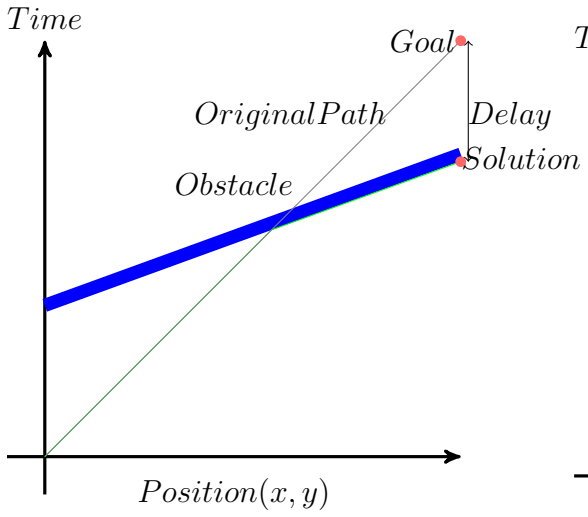


Figure 2.1: Visual interpretation of how the solve space looks like when a low priority ATR needs to speed up for an higher prioritized ATR, observe that this results in a negative delay.

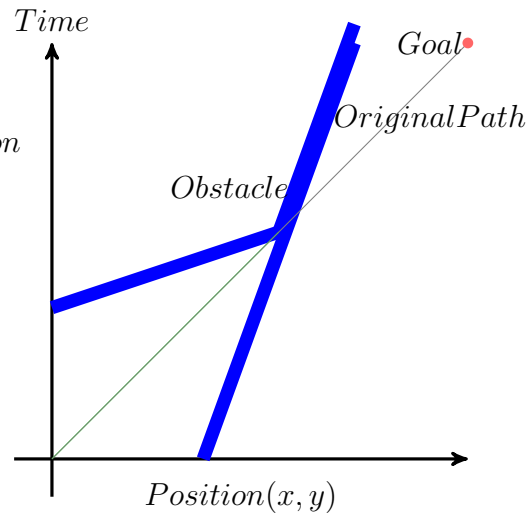


Figure 2.2: Visual interpretation of how the solve space looks like when a low priority ATR gets cut off from a viable solution by two higher prioritized ATRs.

2.6 Evaluation

Simulations will be used to evaluate and also develop the subsystems needed to solve the described problems. These simulations will be used not only to evaluate the final

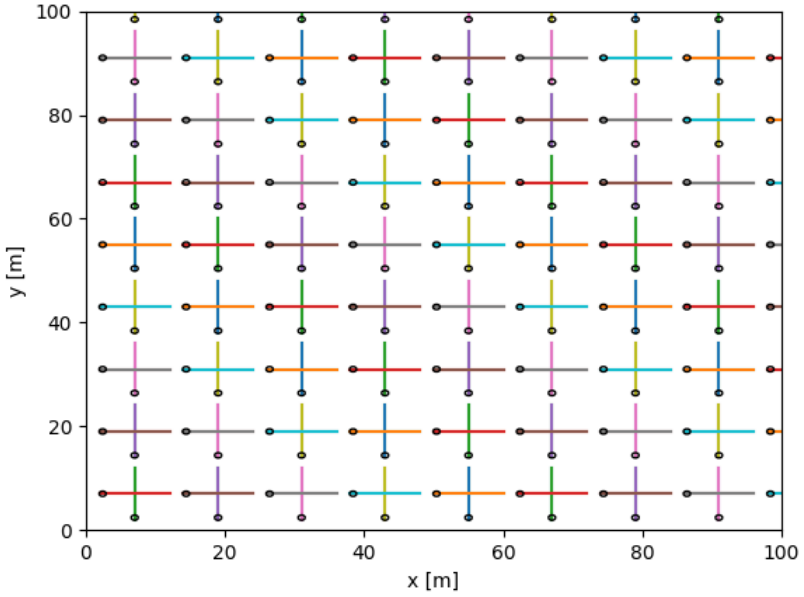


Figure 2.3: Worst case scenario where multiple ATRs collide at the same time and the solver has to solve multiple collisions at once.

project but also during development. Additionally, the system will be tested in a physical demonstration, with a small fleet of ATRs, to make sure the system also works in reality. One feature of the physical demonstration is the ability to compare the new system with the current system in a controlled environment. A key feature that can not be tested through the physical demonstration is how larger fleets of ATRs perform. The reason for this is that there will be a limited number of ATRs in the physical demonstration. The simulation then becomes an important tool when evaluating how the new communication architecture affects fleet size.

The new implementation will also be tested with various worst case scenarios to see how robust the system is. This could for example be to set the whole fleet to enter a critical section at the same time to test the anti-collision system, Figure 2.3. Another example is to make the fleet of ATRs receive their goals at the same time to test the communication architecture, Figure 2.4.

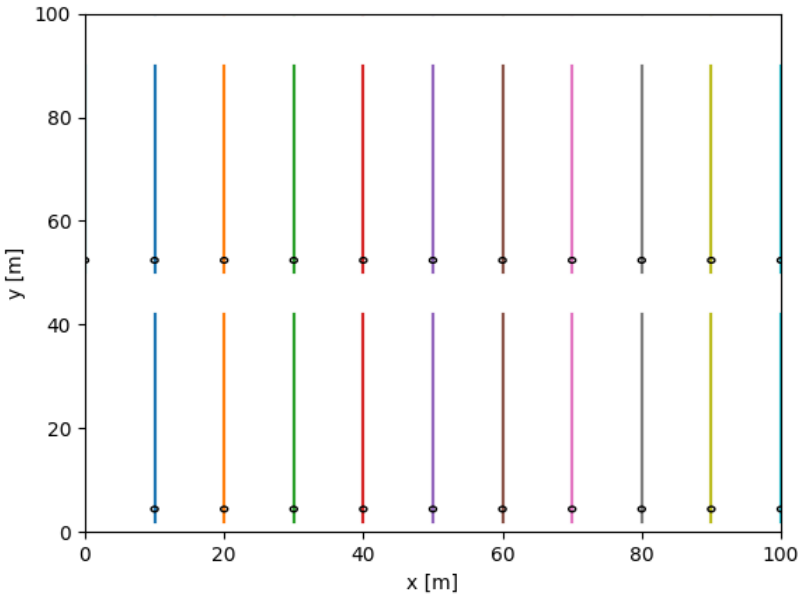


Figure 2.4: ATRs in a grid with no collisions to test how many ATRs can be sent messages simultaneously.

3
implementation

3.1 ROS 2

The ROS 2 connections as input and output from this project were created by Emmanuel Dean together with all of the projects. They are implemented as seen in the following figure 3.1, where the left side is the inputs and the right side is the outputs.

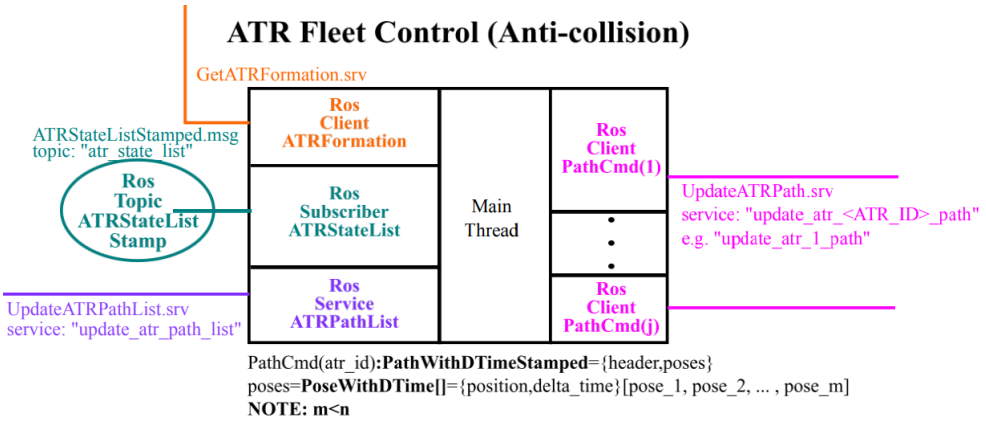


Figure 3.1: The inputs and outputs from the fleet control system.

In ROS2 there are different ways of communicating, the two used in this project are the client/server and the subscriber/publisher. In the client/server configuration the client requests the service from the server and then gets a response [7]. There can be multiple clients but but only one server. With this type of communication the client can be sure that the server got the message and it executed without error. Subscriber/publisher are connected to a specific topic, subscribers to this topic will listen to anything that is published here and publishers will publish to anyone who is listening to this topic [8]. This means that the publisher cant make sure that a certain subscriber got the message.

The input ATRFormation is of client type and returns the formations for the ATR id's it is sent. The input ATRPathList is of service type and takes a message which contains multiple ATRs and their respective path. This callback function uses the ATRFormation client, to get the formations, and adds the new paths with the formation information of the ATRs to the anti-collision algorithm. The input ATRStateList is a subscriber node, this node updates all of the ATRs latest recorded

positions so that the anti-collision algorithm then can flag if there is a discrepancy between the actual position and the position the algorithm thinks it is at. The output PathCmd(1),...,PathCmd(j) is a set of clients where each one goes out to one of the active ATRs. It outputs the next 30 points to the ATR, this topic is updated once every second.

3.2 Simulation and modelling

In order to validate results during and after development, the input and output to the anti-collision system need to be simulated. To do this the communication framework was used as a template to know what was needed to generate the input. To understand the output a visualization tool was made.

3.2.1 Input

The input is called ATRPathList and it contains all active ATRs paths with time stamps. To generate this input, a path generator was developed. The path generator can create lines, circles, or custom paths in the same format as the required input. The path generator can also produce randomized to find edge cases.

3.2.2 Output

When it comes to how to validate the output a tool to visualize the ATRs was made. This tool takes the output and draws the ATRs movements after the anti-collision has done its part.

3.3 Finding a collision

Finding a collision among all other ATRs and their time stamps is a computationally heavy task if each point on the trajectory is compared. Instead, a grid is used with the sides (x,y) step length of 2 m and a time step of 1 s (z). The grid size is chosen with respect to the maximum velocity of 1.5 m/s so that an ATR will never move over a step in the grid. This grid is of fixed size, a grid of 500,500,300 was used to run the simulations and the demonstration with. Each update the first plane in x,y needs to be removed and a new plane of zeros appended in the end.

When a path is altered or added it is updated to the grid. In order to update the grid, the path first needs to be discretized into the grid cells. Each grid cell contains a value of how many paths are passing through that cell, so all of the cells that the new/altered path is in is incremented by one. To check for collisions the adjacent cells of a discretized point on the trajectory are summed and if the value is greater than one there is a collision. Greater than one because the previous value from the trajectory will be in an adjacent cell. If the ATR moves in a diagonal with respect to the grid it may find a collision with itself but the solver solves this situation quickly, see solving a collision chapter 2.4. Once a path has been added to

the grid all of the potential collisions need to be checked for involved IDs and close collisions are stitched together. Because one actual collision may be part of several potential collisions if they are colliding over a large area and time. The collisions will be stored in a list where the (x,y,time) of the first collision in several stitched collisions happen, also the IDs of the involved ATRs, and the interval of the collision (x,y,time). These are the collision that will be looped over and solved.

3.4 Solving a collision

The collisions in the list are only potential collisions. The algorithm is a depth-first algorithm that has weights on the different branches which is a greedy best first search (Algorithm 1). The branches are either to speed up, slow down, or normal. Normal means following the original path timestamps, speed up increases the speed of the ATR and slow down decreases the speed of the ATR.

The solver is weighted so that it will follow the original path (normal action) until it runs into an obstacle, in this way if nothing is in the way it will go to the next point in the trajectory with the same weighting. If the normal branch is not possible then slow down is selected. This is because it is better to slow down due to safety reasons. If that doesn't work either the algorithm will try to speed up. If the potential collision isn't an actual collision, meaning it continues with normal action through the collision area (minimum safety distance is not breached), then the algorithm will only do the normal action at each trajectory point. If there is a collision the result time will depend on the complexity of the collision, in most cases, it is either an intersection or catch up/head on collision. If the algorithm steps one step with slow down at a point in the trajectory then the next point in the trajectory will be weighted differently. It will first try to increase the speed, then normal, and then slow down again. This weight shift is what keeps the original path as much as possible. If slow down or speed up is selected the new speed is calculated from equation (3.1),

$$V_{new} = V_{old} - (V_{old} - S(D/T_C))M, \text{ where } S(D/T_C) = \begin{cases} |D/T_C| < V_{min} V_{min} \\ |D/T_C| > V_{max} V_{max} \\ \text{else } D/T_C \end{cases} \tag{3.1}$$

where D is the distance between the two points the new velocity is calculated from, T_C is the calculated time it can take with maximum or minimum acceleration and M is a multiplier. If D/T_C is either larger than V_{max} or lower than V_{min} then it is replaced with the max and min respectively. The time is calculated differently depending on the situation, in the normal case, it is just the time between the two points with the current velocity. In the slowed-down case, it is set to V_{min} to get a constant delay, and in the speed up case, it is calculated from the equation (3.2),

$$ax^2 + V_{old}x - D = 0, \tag{3.2}$$

where a is the acceleration, V_{old} is the average velocity at the previous point in the trajectory and D is the distance.

Algorithm 1: Recursive Best First Search

Result: A recursive function that returns the path if a path is found or None if there is no path found.

```
RBFS;
if Reached goal then
    return x,y,time;
end
if Position occupied then
    return None;
end
Select weighting;
Calculate time;
RBFS(new time);
```

Before the algorithm starts to solve the collision, the priorities on the ATRs are defined. The first one from this list is allowed to move along its path without any modification to the timestamps. This path is added to a grid with zeros and has a radius of 1 m around it. This is to make sure that the other ATRs won't run into it with our safety distance of 2 m. This grid is fed into the solver that then finds a way around it. This can be seen in Figure 3.2. If there are multiple ATRs involved the new path for the second ATR in the list is then fed into the grid with a 1 m radius and then the next ATR is sent through the solver with this new grid with both the previous ones.

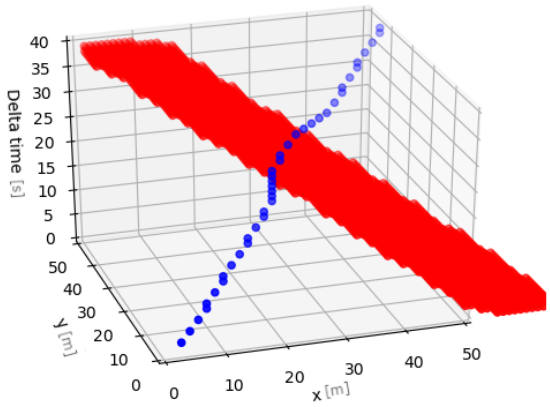


Figure 3.2: One ATR keeping to its original path, red, with a safety distance around it. The other ATR, blue, finds the safety distance and moves around the ATR in time.

3.5 Priority

The priority system was implemented with integers ranging from 1 to as large as the fleet is. Lower priority gives a higher right of way. As explained in the approach, the priority system is a heuristic approach without optimization. One should look at the priority system as a placeholder until further research on how to optimize it has been done. When the solver receives information of a collision, the involved ATRs will be sorted from the lowest to the highest priority where the lowest will have right of way. The solver will then take the first ATR in the list and add it to its "solve space", the second ATR in the list will have the timestamps of its path changed so it won't collide with the first ATR in the list. After the situation is solved, the ATRs path that has been altered will be added to the "solve space" and the next ATR in the list will be altered. In other words, the ATR that is first in the priority list will never be altered. To solve the "catching up" problem, a tool that can detect if an ATR is approaching another ATR from behind or not was developed. First is the direction of each ATR is taken out from a point where they are the closest to each other. If their heading is to close to each other it means they are going in the same direction during the collision. Then with the help of the relative vector between them and their first location in the collision, the ATR that is in front is found and is then set as the ATR with the right of way, regardless of its priority. If there are several ATRs on the same path the whole "queue" is extracted through a recursive function. To save computing resources when dealing with the "catching up" problem a way of blocking catching up collisions for the solver was introduced. The reason behind this is that when a catching up collision has been solved, the finding collision part of the system still sees it as a collision. This is because the finding collision part of the system triggers on a larger distance than the minimum safety distance. Through blocking a solved catching up collision, the system doesn't waste resources on a scenario that's already been solved. If a new ID is introduced into the collision or if any of the involved paths are altered the blocking flag is removed and it will solve the changed collision.

3.6 Evaluation

In order to evaluate the system robustness in terms of refresh rate, communication and solving time, several test cases were created. The first test was done with only running the main node with no active ATRs. This was done to get a baseline of the refresh-rate of the system.

The second test was to evaluate the scalability of the solution, to test how many ATRs can be handled at the same time to test the communication. To ensure no collisions could happen and interfere with the communication evaluation all ATR paths where placed outside of each others collision radius, seen in Figure 2.4. 100 ATRs were initially added and then increased until the refresh-rates were slower than 1 Hz. This has to be done with two computers where one will simulate the fleet of ATRs.

The third test case was similar to the second test. However each ATR has at least one collision in its path, seen in Figure 2.3. This test was done to evaluate the collision system as a whole, and to see how the refresh-rate is affected when collisions is introduced.

4

Results

4.1 Hardware

All of the following tests have run on two computers with the specifications seen in Table 4.1.

Table 4.1: The specifications of the two computers used as sender and receiver during the simulations.

	PC1	PC2
Processor	AMD ryzen 5 3600x	Intel i5 3210m
Graphics card	Nvidia gtx 1070	Nvidia gt 630m
RAM	16Gb	8Gb

All of the simulations were done with the PC1 running the anti-collision node and sending the information to the PC2.

4.2 Communication

Testing the latency between sending and receiving messages with a large amount of ATRs in the system has to be simulated computer to computer and not between one computer and multiple raspberry pi's. The raspberry pi's are the main computing system for the ATRs and part of that hardware. To accurately simulate the whole system one would need as many raspberry pi's as there is simulated ATRs. There weren't this large amount of ATRs, therefore it had to be simulated computer to computer. The results for this test can be seen in Table 4.2, one thing to notice is the increment in the delay at 300 ATRs for test 1. Test 2 handles many more ATRs with low latency. The reason for this increment is unknown but it could be the fact that the computer receiving all the data can't handle the input with other background processes running.

Table 4.2: Message delay between sent and received message with different amount of ATRs in the system.

ATRs	Test 1 message latency (ms)	Test 2 message latency (ms)	Lost messages (%)
100	50	51	0
200	101	95	0
300	1187	154	0
400	3762	321	0
500	3569	4813	0

4.3 Simulation and modelling

The overall result of the simulation and modelling of inputs/outputs to the system can be divided into two parts. One part is to evaluate how well the simulation mimics the real system it is supposed to replace. Another part is to show how the simulation aided the development while the other group's systems were in development. The results showing how well the simulations mimicked the other independent systems will be shown below, but only for the parts with enough data that were accessible. When it comes to how the simulation and modelling aided the development, this will be explored in the discussion since there are no quantitative results.

4.3.1 Real world demonstration

In figure 4.1 the distance between where the simulation said the ATRs should be and where they are in a real-world demonstration is shown. The mean is 0.1405 meters and there is a big gap at the start of the demonstration. The reason for this gap is because the actual position doesn't start exactly in the position of the planned trajectory. Furthermore, in figure 4.2 the simulated path and the actual path are shown zoomed in a corner. Here, it is clear that the real-world demonstration can not do perfect 90 degree turns and hence the spikes shown in the figure 4.1.

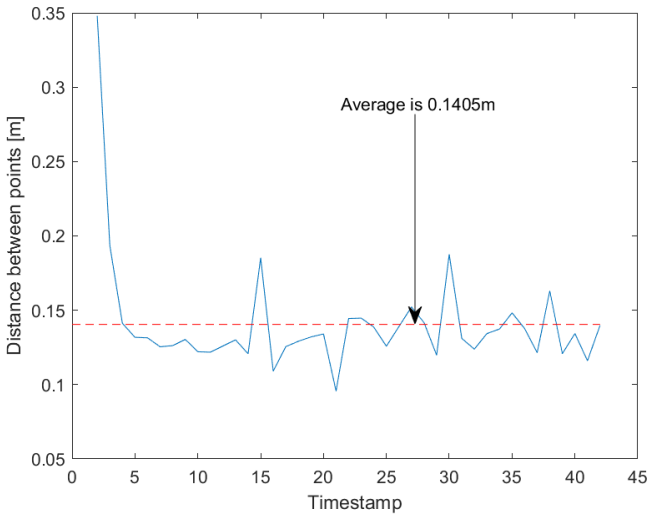


Figure 4.1: Plot showing the distance between simulated position and actual position.

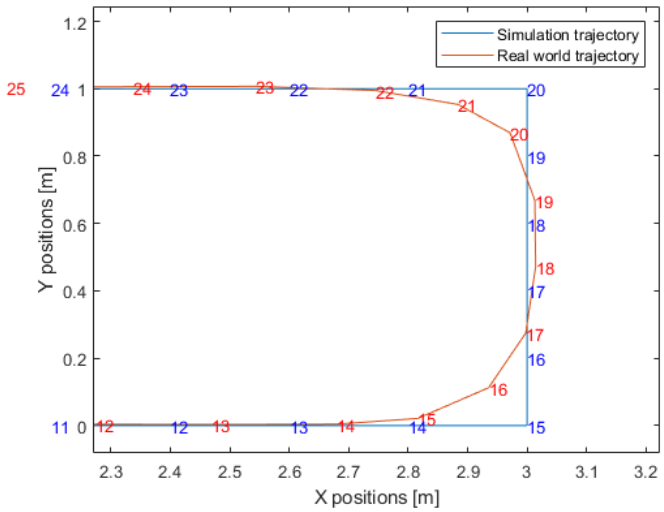


Figure 4.2: Zoomed in plot over a corner with simulated trajectory and actual trajectory, each point is highlighted with the corresponding timestamp.

4.4 Finding & solving a collision

In figure 4.3, the time it takes to find a collision is shown for an increasing number of ATRs. Each ATR has one collision in its path.
In figure 4.4, the sleep time is shown for 100 ATRs in the system with no collisions. Sleep time is what is left of the 1 s update frequency after having done the calculations. We can see that the first step takes a large amount of time, this is because all the services and subscribers need to be created in the first iteration. Furthermore, the last 30 iterations increases its sleep time with each iteration. The reason behind

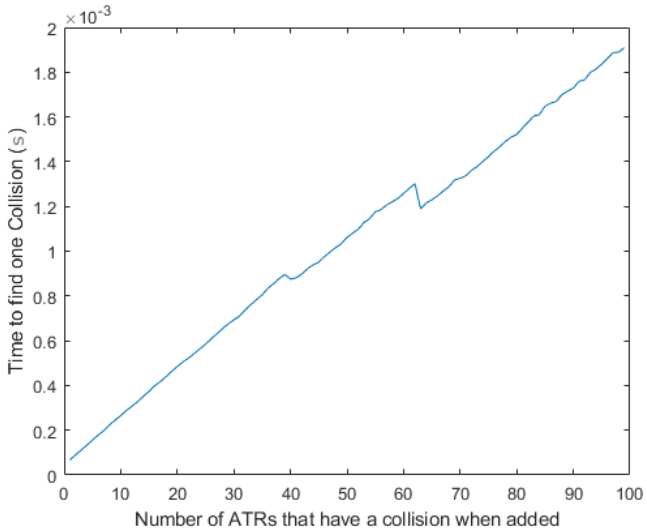


Figure 4.3: Time to find a collision versus the total amount of ATRs. As can be seen the amount of time needed increases with the number of ATRs.

this is that the remaining path length to send for each ATR is shorter than the message length then, the system has less data to send in each iteration. Initialization of the services and subscribers happens once in the beginning of an added ATR. Also, the last 30 iterations the message gets shorter with each iteration because the trajectory isn't longer. To avoid bias because of this and find results that are not affected by path lengths or if the system is starting up, it was decided to disregard the first and last 30 steps when looking at average sleep times in the following tests.

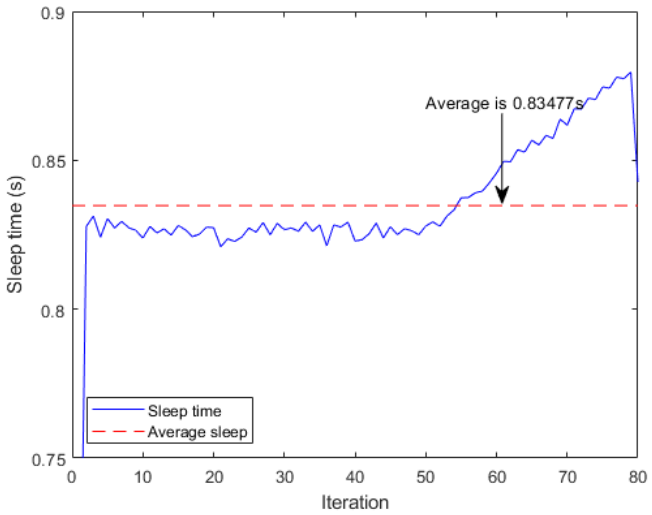


Figure 4.4: Sleep time each iteration with 100 ATRs and no collisions in the system. With sleep time closer to 1 s the calculations are closer to 0 s.

The main loop sleep times were recorded with no active ATRs to quantify how much time the main loop takes in each iteration with no active ATRs, and to have

a base line for further tests. The results from this test can be seen in Figure 4.5. The average sleep time is around 0.87583 s and is steady under 5 minutes of run time. This means that the main loop without any collisions or ATRs to solve/find collision takes 12.417% of the available time to keep the 1 Hz update frequency on the current hardware. One can also see that the sleep time is stable and does not increase with time.

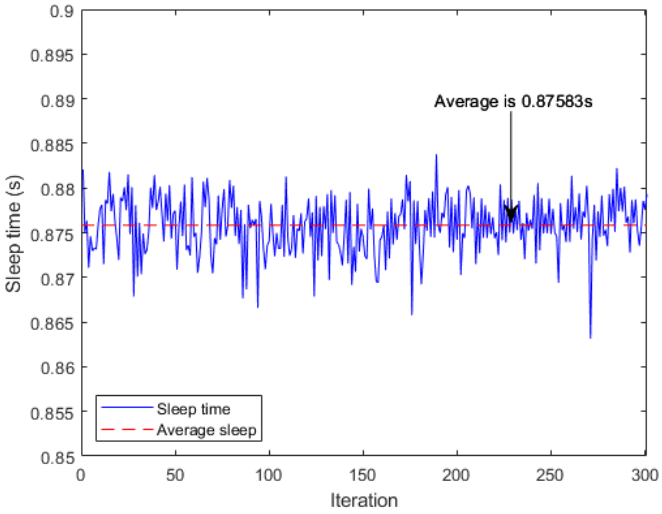


Figure 4.5: Sleep time each iteration with no ATRs in the system.

In the next test, all added ATRs had no collisions in their paths. In Figure 4.6, the sleep time in each iteration is shown with an increasing number of ATRs. The sleep time is still stable and one can also see the standard deviation on the error plot. Furthermore, the curve has a linear behavior.

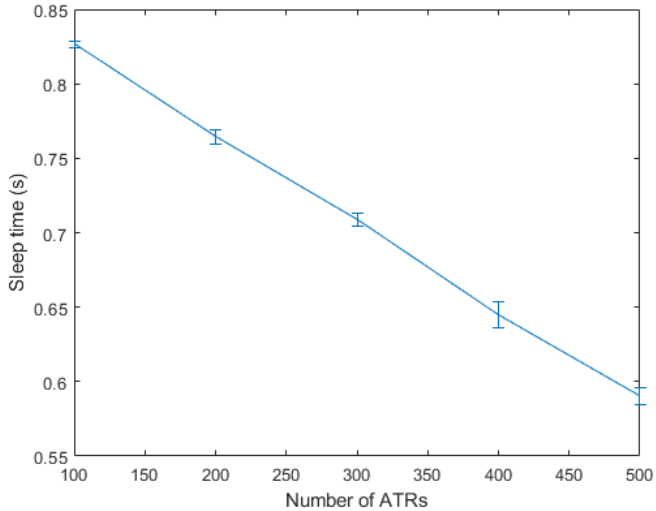


Figure 4.6: Sleep time plotted versus active ATRs with no collisions. With the total number of ATRs increasing the sleep time decreases.

In figure 4.7, the sleep time is plotted versus the number of active ATRs each with one collision in their path. This result is generated with all the ATRs colliding at the same time in order to generate a worst case scenario, seen in Figure 2.3. Once again one can see a linear behavior but with a much larger standard deviation. The standard deviation is also increasing with the number of active ATRs.

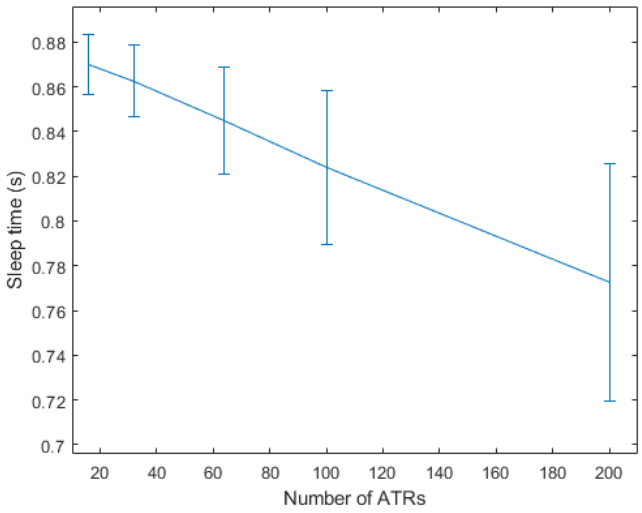


Figure 4.7: Sleep time plotted against active ATRs with collisions.

Furthermore, in Figure 4.8, the solving time when using the same collision scenario but with different step lengths can be seen. Notice how the solving time gets shorter when the step length gets longer. As well as when the step length is longer the delay increases.

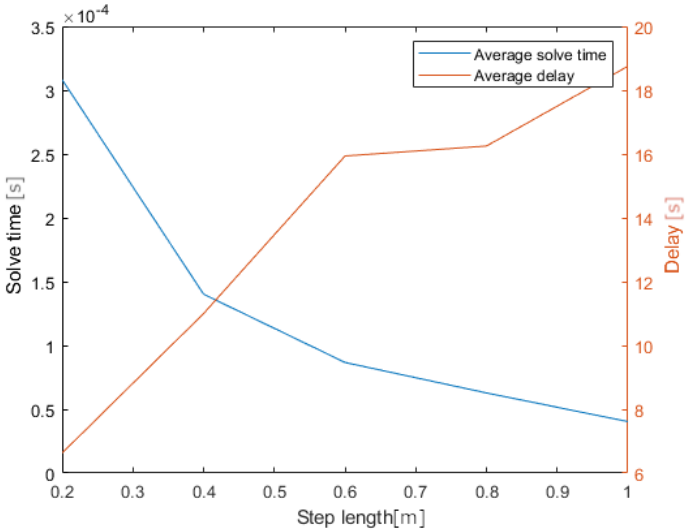


Figure 4.8: Average solve time and delay plotted against the step length with 32 collisions.

In Figure 4.9, 100 intersections (collisions) are created with two ATRs in each. These

intersections are identical but are placed at different locations in the grid. In the figure, the solving time can be seen to increase from 0.225 ms to 0.450 ms. The trajectory delays are mostly the same. This is expected since the intersections are the same.

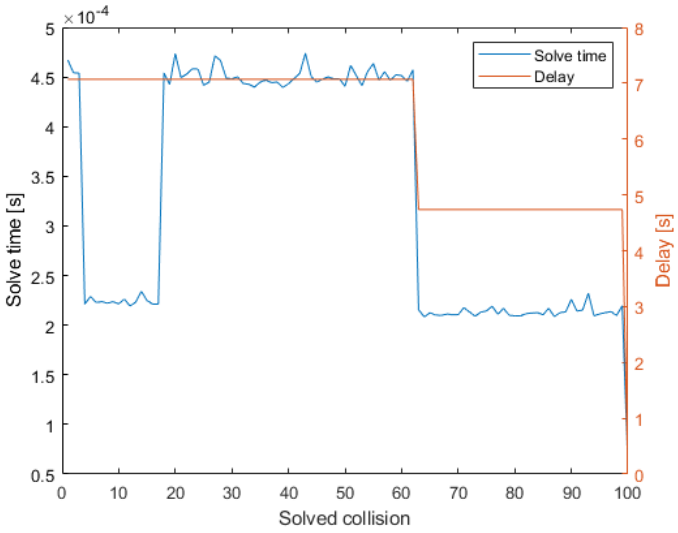


Figure 4.9: Time it takes to solve an intersection collision and the delay it creates.

4.5 Priority

After solving various collision situations through simulations it is clear that the solver can handle most "easy" scenarios. An "easy" scenario, is for example, a pure 90 degrees intersection. However, the current priority system has issues when sections of paths are overlapping with opposing directions for longer distances. The maximum length of the overlapping sections in order to solve them, is affected by how the path looks in terms of how far apart each point is located. Acceleration constraints play an important part in this. Furthermore, the priority system also had problems when several ATRs are following a common path in a train configuration, or in other words a large catching up scenario. The problem with this scenario was evident when an ATR is in collision with a train of ATRs, the first ATR in the train would slow down due to priority reasons to allow the ATR to cross. This in turn led to the second ATR in the train colliding with the leading ATR since the collision scenario between them was regarded as solved and therefore blocked as discussed in the implementation. However, the priority system worked well when solving catching up problems with several ATRs on the same path, it was only when other collisions were introduced the system started to fail.

5
Discussion & Conclusion

5.1 Communication

The communication interface provided by ROS2 presented good scalability under stress testing. This indicates that it could be used in real-world applications. One of the drawbacks seen in the results is the latency. This is not necessarily ROS2 problem but may be caused due to the fact that just one computer simulating several hundred raspberry pi's was used. With 100 ATRs, there are no lost messages and a small latency of 50 ms. Even though the latency is 50 ms, the fleet control node still manages to send all 100 messages within 1 Hz.

To handle the 50 ms latency, the clocks need to be synchronized and the messages time stamped according to the clock. With that extra information, it would be possible to estimate the latency and then calculate the path according to this latency. If there is a big difference in the latency between different ATRs getting their messages, there can be a chance of a collision in the current state of the system. This is because the system assumes there is no latency in sending messages. Since the messages can't be sent simultaneously, there will be an offset in time between the first and the last message that is not accounted for at the moment. With synchronized clocks this will not be a problem. By solving these issues, the solver could have an even smaller safety margin which would lead to faster solving times since there it need fewer solving steps.

The latency difference seen in Table 4.2 can either something that happens in the network at the same time that slows down these ROS2 messages or the fact that it is one computer that receives all of these messages and it can't handle it with other background processes running.

5.2 Simulation and modelling

Without using simulating and modelling tools to replace other systems close connected to this thesis system, the development would have been much harder. The reason for this is when evaluations and tests can be executed along with the development, critical bugs can be found and subsystems tweaked long before the first trials of the whole system. One could say that the time invested in simulations and models will yield a return later on. However, it is also important to note that even a perfect simulation never can replace the real system. All simulations and models

have their limitations and it is wise to balance the effort of making these tools versus the expected reward.

In Figure 4.1, the average distance between the simulated path and the actual path was around 0.1405 meters. This error is significant when setting the minimum distance between the ATRs. The circular size of an ATR is around 70 cm which means the minimum safety distance needs to be above 140 cm and when adding the position error then the safety distance needs to be above 170 cm. The reason behind the error is latency issues that have not been handled. To minimize this error, one could measure the latency and then account for it when following the given paths time stamps. Another plausible contributor to the error could be other latency issues with the controller of the robot and time miss matches due to longer computational times in the fleet control node.

5.3 Finding a collision

Finding the collisions works well and is fast, and so does the solving of the collisions if the ATRs collide in an intersection. The solver cannot guarantee a short solve time but is in the order of 0.225-0.450 ms most of the time for these type of intersections. The time is linearly increasing with the amount of ATRs in the system seen in Figure 4.3. The reason for the linear behavior is that the ATRs id are not saved in the 3d map over the factory, which means when a collision is found the involved ATR needs to be found. This could be solved by using more memory and save the ATRs in the grid map. Then the list of current active ATRs wouldn't have to be searched and it would find the collisions in constant time.

The program can find a collision of an ATR with that same ATR if it moves diagonally on the grid. The collisions involving only one ATR are easy to remove but to find the affected ATRs is in linear time with respect to the amount of ATRs in the system. With the above solution, this would also be solved since the only ATR in the vicinity would be itself and there would be no collision.

5.4 Solving a collision

As can be seen in Figure 4.8, the solving time is greatly impacted by the step length or "resolution" of the problem. A higher "resolution" problem has smaller steps in the input path and/or strict limits on acceleration and maximum speed. This forces the solver to take more recursive steps with a more accurate solution. Accurate in the sense that the corrected path is closer to the defined minimum safety distance with no overshoot and smother time stamps. In Figure 5.1, a visual demonstration shows how the algorithm solves a high-resolution path. The best first search needs to explore 47 branches before finding the solution. The same visual interpretation but with a lower resolution path can be seen in Figure 5.2. Here the only thing that has changed is the step distance which is doubled compared to the example in Figure 5.1. Now, the best first search only needs to explore 17 branches before the solution

is found. One can also see that the solution goes further away from the obstacle than needed, which also leads to deceleration earlier than in Figure 5.1. In other words, overshooting obstacles leads to the need for more space to solve the collision. Overshooting can also lead to delay as seen in Figure 4.8. The reason for this is that the solving window is too small for the solver to get the path back on the original time stamp since extra space is needed. Furthermore, the angle between the different branches in the interpretations is decided by the allowed acceleration/speeds of the solution. With higher allowed accelerations/speeds the solver can take larger steps in time, which means that the solving time is shorter with the same overshoot trade-off as the step length. With lower allowed accelerations/speeds the solve time increases since the algorithm is forced to explore more branches before finding the solution because the algorithm is restricted when moving along the time axis.

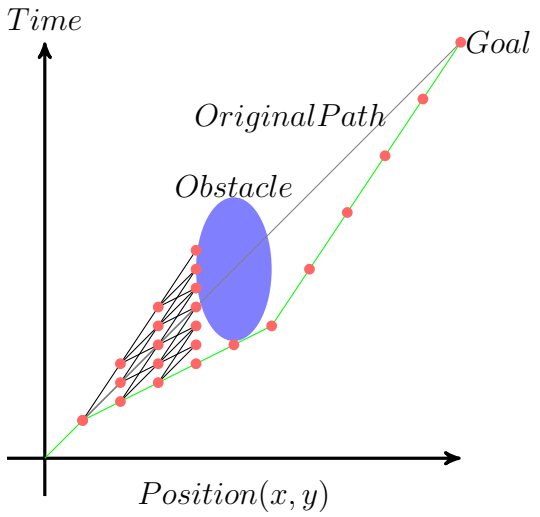


Figure 5.1: Visual interpretation of how the solve space looks like when a high "resolution" path is solved.

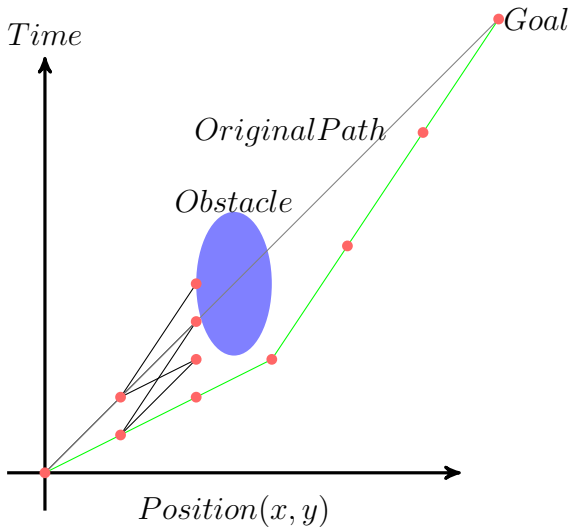


Figure 5.2: Visual interpretation of how the solve space looks like when a low "resolution" path is solved.

There has been no research done to explore how to optimize the acceleration, speed and step length to have a fast solve time with as little overshoot as possible. Hardware and how many ATRs the system should handle is also a factor to investigate when optimizing the solver since better hardware and fewer ATRs could open up for using a higher "resolution" path with few trade-offs. To summarize, the best first search algorithm is both fast and slow at solving collisions depending on hardware, step length, acceleration, and top speed restrictions. The weighting of the algorithm opens up the possibility to always find the solution with no or smallest delay, which is an important feature when dealing with just-in-time delivery. This solution needs further research since there are several parameters to be optimized, it did however show great potential for the use case presented in this thesis.

5.5 Priority

The priority system mentioned before is an heuristic solution and should not be considered an actual working part of the system. The reason for this is that implementing priority in the logistics makes more sense to get the most out of the delivery system. The implemented priority system works well when choosing which ATR should have the right of way. However, its complexity increased when introducing large train constellations and "catching up problems". The priority system could still solve most of these cases but larger flaws were also discovered in some specific cases. For example, using the direction of the ATRs to find which ATR is catching up with another is not optimal. If for example, if an ATR is standing in an intersection with a different heading than another ATR entering the intersection, collisions could still happen. If in further development of the system it is decided that priority still should be handled by the anti-collision system other solutions to this should be explored. One solution could be to try many different priorities for every collision and choose a successful solution with the smallest overall delay, this would be a brute force solution that will take more computational power.

5.6 Summary

Volvo requires a real-time collision avoidance that can handle a large fleet with an update frequency under 1 Hz by only changing time stamps on paths. All these requirements have been kept throughout the project and, even when there are obvious areas of improvement, the usage of a greedy best-first search for finding solutions is interesting and requires further research. Finding the collisions also needs more research for even larger fleets of ATRs. Searching for collisions in the vicinity of the new path is faster than searching all paths to see if they intersect but needs more memory. The priority system is an interesting part of the system since many optimization aspects have not been explored. The fact that the system could handle over 200 ATRs all colliding at once using the algorithm with no optimized parameter and using only 50% of the available time shows the potential of the system. The test is also done with a factory floor of 1000X1000 meters which also shows some scalability aspects. Although the system has flaws, these benchmarks show potential for further work.

6
Future Work

To further enhance this system, there are some areas of improvement. One is that the clocks of the computer running the fleet control and the raspberry pi in the ATRs needs to be synchronized. This will help create a more accurate trajectory following and any delays in the network will be accounted for by the ATR controller.

The finding of collisions could be improved by using more memory. By saving all of the ID's at the positions in the grid where they are moving through, it would be possible to find the collisions in constant time with respect to the amount of ATRs in the system. This would however need more research since increasing the memory may have bad scalability.

To optimize how fast the greedy best-first search algorithm can run and not run in to a dead end it is important to tune the distance between points and size of the solve window based on acceleration and top speed requirements as well as hardware. Further investigation needs to be done to find a suitable length that is good but still short enough to have details in the trajectory. Continuing on this track, one could improve the algorithm itself by removing already explored trees. This would help calculations by not calculating the same paths over and over again. The more complex cases would benefit a lot from this.

Another problem that can be improved is to save the delay over a longer time than just in the solver for that solution. This together with a speed up / slow down function would make sure that the ATRs reach their destination at the correct time more often. Because as seen in Figure 4.9, the delay is not 0 s at the intersections so the solver didn't have a large enough solve window to solve it with 0 s delay. This can be done outside of the solver since the collision has been avoided or with a larger solve window.

To improve the priority system one should look into the possibility of trying to solve all the different ways of ordering the ATRs in a collision and then choose the solution that has the best outcome in terms of overall delay. This will however need more computational power, but headroom could be created for this by tuning the best-first search algorithm. Furthermore, could weights be incorporated so that the coordinator can mark some ATRs as more important if they are in a hurry to deliver. If no changes are done to the priority system another solution to the "catching up" problem needs to be found.

Bibliography

[1] "Ros," https://www.ros.org/, accessed: 2021-02-08.

[2] M. W. Maier, "Architecting principles for systems-of-systems." *Systems Engineering*, vol. 1, no. 4, pp. 267 – 284, 1998. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edo&AN=ejs23842417&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[3] E. Erős, M. Dahl, K. Bengtsson, A. Hanna, and P. Falkman, "A ros2 based communication architecture for control in collaborative and intelligent automation systems," *Procedia Manufacturing*, vol. 38, pp. 349 – 357, 2019, 29th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM 2019), June 24-28, 2019, Limerick, Ireland, Beyond Industry 4.0: Industrial Advances, Engineering Education and Intelligent Manufacturing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2351978920300469

[4] "Ros 2," https://design.ros2.org/articles/ros_on_dds.html, accessed: 2021-02-08.

[5] P. Acheson, C. Dagli, and N. Kilicay-Ergin, "Model based systems engineering for system of systems using agent-based modeling," *Procedia Computer Science*, vol. 16, pp. 11–19, 2013, 2013 Conference on Systems Engineering Research. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050913000033

[6] F. Pecora, H. Andreasson, M. Mansouri, and V. Petkov, "A loosely-coupled approach for multi-robot coordination, motion planning and control," 2018. [Online]. Available: https://www.aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17746

[7] "Ros service," https://docs.ros.org/en/foxy/Tutorials/Services/Understanding-ROS2-Services.html, accessed: 2021-05-17.

[8] "Ros topic," https://docs.ros.org/en/foxy/Tutorials/Topics/Understanding-ROS2-Topics.html, accessed: 2021-05-17.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY