Simultaneous Multi-Robot Task Scheduling and Path Planning

An integrated approach to task scheduling and path planning for mobile robots in production environments

Jeremy Aarts

Master of Science Thesis

Delft University of Technology

Delft Center for Systems and Control

Simultaneous Multi-Robot Task Scheduling and Path Planning

An integrated approach to task scheduling and path planning for mobile robots in production environments

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Jeremy Aarts

September 5, 2018

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was supported by Prodrive Technologies. Their cooperation is hereby gratefully acknowledged.

Abstract

Manufacturing processes often run on a schedule, because each production step takes a certain amount of time, after which a component is transported to the next machine. Mobile robots can be used to automate transportation of components between production steps. To do so, a task scheduler assigns transportation tasks to robots and a path planner calculates trajectories for robots to follow to complete their tasks.

In scientific literature, path planning and task scheduling are treated as separate problems. Typically, path planners use initial and goal positions as a given and task schedulers consider tasks of fixed duration. In this thesis, the integration of task scheduling and path planning is treated, resulting in three contributions. First, a novel cost function to minimise the duration of paths is proposed for an existing Mixed-Integer Linear Programming (MILP) formulation of the multi-robot path planning problem. Second, a new MILP formulation is developed for the scheduling problem of tasks with deadlines to be completed by mobile robots. Third, two methods for integrating the path planner and task scheduler are proposed. Through simulations these methods are compared with an integration method that uses rule-based scheduling, which assigns tasks to robots in a predefined order.

Simulation results show that the novel path planning objective function improves upon that from the literature by striking a balance between duration, distance, energy use and computation time. Furthermore, integration methods using the MILP scheduler have trouble outperforming a rule-based approach, caused by a practical limit on the computation time. Moreover, compared with single-robot path planning the use of multi-robot path planning is shown to improve performance at the cost of higher computation time. However, this improvement is negligible for environments with a lot of free space, where robots can easily avoid each other without making detours.

These results lead to the conclusion that the MILP path planner with the new path planning objective function performs well when integrated with a task scheduler. For restrictive physical environments, multi-robot path planning can further improve results. On the other hand, for practical applications a scheduling method other than the MILP scheduler should be employed to integrate task scheduling and path planning. The scheduling method should at least outperform rule-based scheduling with a practical limit on computation time, even if an optimal solution is not guaranteed.

Table of Contents

Acknowledgements

I would like to thank my supervisors prof.dr.ir. Bart De Schutter, dr. Alfredo Nuñez Vicencio and Bas van der Oest (MSc) for our regular meetings and discussions and their assistance during the writing of this thesis. Their comments and advice were always helpful and steered me in the right direction. I would also like to thank Victor Dieben for our interesting discussions and for his periodically reviewing of my work. I have found that the deeper you dive into a topic, the more you discover how little you know and how much there is to learn from others.

Furthermore I thank my family and friends for their support during the past six years of study and in particular during the past year of working on this thesis.

Delft, University of Technology — Jeremy Aarts
September 5, 2018

“I love deadlines. I love the whooshing noise they make as they go by.”
— Douglas Adams, *The Salmon of Doubt*

Chapter 1

Introduction

1-1 Background

Manufacturing processes for products are often divided into stages. During each stage a manufacturing operation is applied to a component, after which it is passed on to the next stage. Each stage takes one or more components as inputs, and produces an output after performing its operation. By physically separating the production stages it becomes possible to modify individual stages or expand them without having to redesign the entire production line. In separating production stages the need arises for transporting components between stages.

One method for transporting components between stages is to use conveyor belts or robotic arms, but they need to be modified when the production line changes. It would also be necessary to place one between every production stage. Another option is to let humans transport components between stages, but human labour is costly.

Mobile robots as devices for transportation of components between stages do not have these disadvantages. Using them allows production lines to become more flexible, because stages can be modified, expanded or rearranged without having to make any changes to the transportation system, apart from some software changes so the robots can find the new locations for pick-up and delivery. Another advantage is that one robot can handle transportation between multiple stages, because it can move freely through (part of) the production environment. Furthermore, running costs are much lower than for humans, although robots come with an investment cost.

Production processes often run on a schedule, with each production stage requiring a certain amount of time to complete its operation and having a certain throughput. To make sure mobile robots pick up and deliver components on time, an automated task scheduler is used to assign pick-up and delivery tasks to robots available in a production facility. Timely completion of tasks also depends on the paths taken by robots, since these paths influence on the duration of completing a task. A path planner computes a path for a robot to follow from its initial position to its destination.

When it comes to task scheduling, many challenges exist. One is how to take varying priorities of tasks into account in the scheduling process. Another is to determine how many tasks to schedule for at once. For example, it could be efficient to schedule tasks up to a finite time horizon that shifts into the future over time. Another challenge is that of determining where robots go after completing a task. Options include defining buffer zones for robots to go to and moving to unoccupied pick-up or delivery stations when an idling robot is blocking the way at such a station. Finally, computation time is also factor to be taken into account to determine the practicality of a task scheduling method.

The path planning problem also has many challenges. Planning and following paths requires information about the locations of robots. Keeping track of these locations is called localisation, for which many methods exist. Localisation involves the use of sensors and a prediction model to accurately determine robot locations. Besides keeping track of robot locations, a path planner also uses information about obstacles to find paths that are collision free. Gathering and updating this information is a procedure called mapping, i.e. representing the physical environment as constraints for the path planner to deal with. A challenge that arises when path planning for multiple robots is deciding between a centralised or distributed path planning approach. With a distributed approach robots calculate their own paths individually, while a centralised approach plans paths for all robots all at once. While centralised approaches are able to produce better global solutions because they can use more information about the problem, they often come at a higher computational cost compared to distributed approaches. As with task scheduling, when using mobile robots in practice, computation time is an important factor to take into account.

This thesis is conducted in cooperation with Prodrive Technologies, a company in the Netherlands that develops and produces electronic and mechatronic products and systems. To assist their production processes where flexibility is of major importance, Prodrive develops mobile robots that can transport components. The long-term goal for Prodrive is to develop a 'lights-out factory', which is a fully automated factory without any humans present, which would theoretically allow the lights to be turned off. This objective is chosen because it is believed that a fully automated factory would result in high efficiency and quality of manufacturing processes, to which human error is detrimental.

1-2 Safety implications

The most important aspect of any technological solution involving humans is its safety. Since humans can be present in production environments at Prodrive and in other places where mobile robots are used, safety implications need to be considered.

When it comes to path planning for mobile robots, several measures can be taken to improve safety of the system. First of all a robot needs to be able to halt at any time, if such a command is given by a human through a stop-button for example. This is to give control to humans in a situation where they do not trust the robot to act safely. Another measure would be to have the robot show its intent to humans around him, with a display showing its planned direction for example. This can help humans to navigate a space with mobile robots safely and prevent unexpected robot behaviour to cause harm. Mobile robots should also be equipped with a collision detection system, so that obstacles or humans of which the location is not know to the robot do not collide with it.

1-3 Problem statement

Task scheduling and path planning both have an effect on the performance of a production line. After all, a particular schedule can unnecessarily cause pick-up and delivery tasks to be started or completed too late, which can cause production to be delayed because a machine spends time waiting for a new component. Likewise, planning paths that are longer than necessary can also cause delays if it causes robots to arrive late. Rather than considering the effects of task scheduling and path planning on the performance of a production line separately, the problem that is tackled in this thesis is the evaluation of the effects of the way task scheduling and path planning are integrated, on the performance of a production line.

Therefore, the objective of this thesis is to **develop a methodology for integration of task scheduling and path planning for multiple robots to optimise performance of a production line**. Since the mobile robots are used to assist a production line, the methodology should be feasible within practical computational constraints for use in a production environment.

Given a set of mobile robots and a set of tasks, the objective of the integration methodology is to assign tasks to robots, determine pick-up and delivery times based on constraints imposed by the manufacturing process and plan paths for robots to perform pick-up and delivery tasks.

1-4 Research questions

The main research question follows from the problem statement and is formulated as follows:

What integrated task scheduling and path planning methodology for mobile robots assisting a production line can be developed, considering computational constraints, to optimise for the performance of the production line?

Three sub-questions treating aspects of the main research question are posed:

1. How can the problems of task scheduling and path planning of mobile robots be formulated to allow for an integrated approach?
2. What integrated task scheduling and path planning approach for mobile robots can be developed to optimise performance of a production line?
3. How do integrated approaches to path planning and task scheduling of mobile robots assisting a production line compare, as measured by the performance of the production line?

Answers to these questions are provided in Chapter 6.

1-5 Reading guide

This thesis is structured as follows. Chapter 1 provides background information to this thesis, defines the problem statement and poses research questions. In Chapter 2 an overview is given on the literature concerning path planning and task scheduling. Chapter 3 discusses a Mixed-Integer Linear Programming (MILP) formulation of the path planning problem and a novel objective function is proposed. In Chapter 4 a MILP formulation of a task scheduler is introduced. Simulation results are reported and discussed in Chapter 5. Finally, in Chapter 6 conclusions are drawn, recommendations are made to Prodrive Technologies and suggestions are made for future work.

All chapters can be read independently, except for Chapter 5 which assumes the reader is familiar with the path planning and task scheduling formulations that are discussed in Chapters 3 and 4 respectively (Figure 1-1).

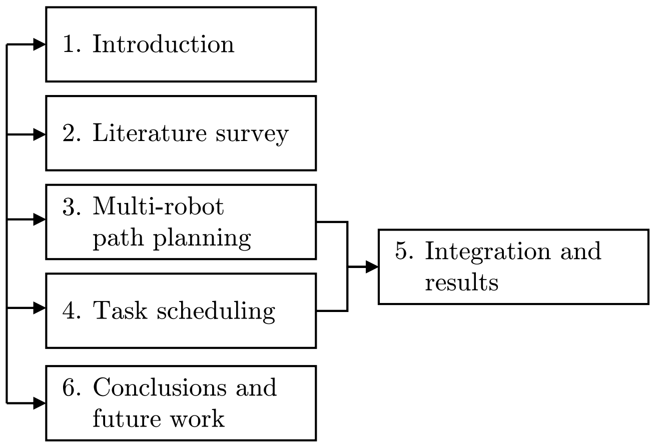

1. Introduction
2. Literature survey
3. Multi-robot path planning
4. Task scheduling
5. Integration and results
6. Conclusions and future work

Figure 1-1: Reading guide showing the relations between chapters

Chapter 2

Literature survey

This literature survey is structured as follows. Section 2-1 discusses aspects of path planning that are relevant to simple problems, with a single robot and no moving obstacles. In Section 2-2, the scope is further expanded to planning in changing environments with moving obstacles. Section 2-3 introduces methods for planning with multiple robots in the same environment. Section 2-5 discusses conclusions from the literature survey.

2-1 Static environments

For a computer to plan a path in a continuous world, the physical environment first needs to be represented in such a way that the planner can search for solutions. Path planning approaches for static environments that are discussed can be divided into three main categories: combinatorial, sampling based and optimisation methods. Furthermore, kinodynamic planning are discussed that use sampling based methods and take vehicle dynamics into account.

2-1-1 Combinatorial planning

The configuration space $\mathcal{C}$ is the set containing the rigid-body transformations that can be applied to the robot [36]. It contains all configurations q of the robot $\mathcal{A}(q)$. The obstacle configuration space $\mathcal{C}_{\text{obs}}$ contains all configurations at which the robot is in collision with obstacles. What remains of $\mathcal{C}$ outside of $\mathcal{C}_{\text{obs}}$ is the free configuration space $\mathcal{C}_{\text{free}}$. Combinatorial methods entail first creating a roadmap, which is a graph of nodes and edges. A search algorithm searches the graph for a path connecting the initial and goal configurations using.

To create a roadmap cell decompositions divide $\mathcal{C}_{\text{free}}$ into cells and place nodes in those cells [9]. For non-circular robots that can rotate, collisions depend on the orientation of the robot, adding a dimension to the configuration space. Paths can be very suboptimal, because the roadmap is heavily dependent on the size and shapes of obstacles and open spaces. A *visibility roadmap* is created by connecting vertices of obstacles and the edges of a *maximum-clearance roadmap* are equidistant to obstacles [32]. Graph search is used to find a path on a roadmap from the initial configuration of the robot to its goal configuration. Uninformed search methods such as Dijkstra's algorithm do not consider prior knowledge about the location of the goal [12]. Informed search methods such as A* use a heuristic, such as a lower bound estimate of the cost to the goal, to guide the search [19]. Dijkstra's algorithm and A* both find the path with the lowest cost from the start to the goal if one exists. For large graphs or with multiple robots computation time can become very long.

2-1-2 Sampling-based planning

Sampling based planning builds a roadmap by randomly sampling points in the workspace and then checking if they are in the free configuration space [13]. For combinatorial methods the complexity of the roadmap depends on the complexity of obstacles, while the complexity of a roadmap generated by sampling based methods depends on the method of sampling and the number of sampling points. Combinatorial methods are complete while sampling based methods are not. An algorithm is complete if it is guaranteed to find a solution or report that none exists. Alternative notions of resolution completeness and probabilistic completeness are used for sampling based methods. A resolution complete planner guaranteed to find a solution up to a certain sampling discretisation resolution of $\mathcal{C}$. Probabilistic completeness means the probability of finding a solution approaches one as the number of samples tends to infinity [11].

The two most prevalent sampling based methods are the probabilistic roadmaps and rapidly-exploring random trees. Probabilistic roadmaps are built during a preprocessing phase by randomly sampling nodes in the free space [25]. The roadmap can be reused to plan multiple paths. Rapidly-exploring random trees samples new nodes every time a new path is planned until a path connecting the initial and goal configuration of the robot emerges [31]. The advantage of probabilistic roadmaps is that after time is invested into generating a roadmap, paths can quickly be planned using the roadmap [44]. This advantage ceases to hold when a new roadmap needs to be generated regularly. Rapidly-exploring random trees have the advantage of finding a path to the goal faster, which makes them more suitable when the environment is too large to generate a complete roadmap or when changes occur in the environment, necessitating a new roadmap. Both methods have been shown to be probabilistically complete [4, 21]. The approximate cell decomposition is a sampling based variant of regular cell decomposition and that is faster but less accurate at defining obstacle boundaries. The method is resolution complete [7].

2-1-3 Optimisation approaches

The potential field method constructs an artificial potential field that guides the robot using gradient descent [26]. Attractive potential attracts the robot towards the goal and repulsive potential keeps the robot clear of obstacles. Another disadvantage is that the velocity of the robot needs to be optimised separately. Furthermore, paths achieved with this method are not necessarily optimal. The potential field method suffers from local minima causing the robot to get stuck. To prevent this, measures need to be taken that result in suboptimal and unnatural paths. Non-circular robots that can rotate require an additional dimension to the potential field, making it harder to find a solution and more likely that local minima are encountered [44]. Another issue with the method is its handling of narrow corridors [28]. Because both obstacles forming the corridor exert repulsive force on the robot, the entrance to the corridor can become inaccessible to the robot.

Mixed-Integer Linear Programming (MILP) is an optimisation method for systems that are described by linear dynamic equations and constraints with continuous and integer variables [6]. Efficient solvers exist to optimise MILP problems, linear programming solvers are particularly optimised for performance. Disadvantageous is the fact that using many integer variables drastically increases computation time [49]. A MILP formulation of the path planning problem is proposed in [40]. The trajectory of the robot is discretised into a finite number of sampling instances with a finite time horizon. Obstacles are modelled using integer constraints. Local minima can occur when the time horizon is too short, which can be remedied by extending the horizon. The method find an optimal path that minimises a cost function calculating the distance between the robot state at sampling instances and its goal state, whereas the potential field method follows the path of gradient descent without calculating an optimal path. Furthermore, MILP is able to find globally optimal paths for multiple robots. Another advantage is that it takes vehicle dynamics into account, like kinodynamic planning methods that are discussed next.

2-1-4 Kinodynamic planning

Kinodynamic planning uses sampling based methods and considers kinematics and dynamic constraints on the robot, such as a maximum turning radius or velocity.

One approach is to ignore differential constraints in the path planning phase, and to adjust the path afterwards to satisfy these constraints [29]. However, this could cause paths to become very suboptimal or even infeasible. In [34], sampling is done of permissible control actions instead of robot states. The control action that moves the robot nearest to the desired state is used. The problem with this method is that it causes unnatural paths with varying control actions, because they are randomly sampled. Another approach is to randomly sample a state and a control action, however this has the same disadvantage [20]. In [16] the free state space is divided into regular states and inevitable collision states. These are states that do not immediately cause a collision but inevitably lead to a state of collision. By avoiding these states future collisions can be prevented [16]. Motion primitives are precomputed motions that satisfy kinodynamic constraints and that can be stitched together to form a path. An advantage of using motion primitives is that it can generate natural paths within relatively good computation time [17]. However, no systematic methods exist for designing motion primitives that optimise planning performance, since they are very specific to a robot model [33]. They are primarily useful for over-actuated robots, because they simplify the problem by reducing the number of control variables. However, this also reduces the size of the solution space.

2-1-5 Discussion

While combinatorial methods have the advantage of being complete because they capture all information about the configuration space, they are harder to implement than sampling-based methods. Additionally, they can become computationally infeasible for increasing dimensions and tightening constraints. The completeness of sampling based methods is limited to a certain resolution. Paths are suboptimal and can seem unnatural, roadmap nodes are randomly generated. Information about obstacles is not used until after sampling so the placement of nodes is inherently less efficient. The potential field method has a problem with getting stuck local minima its paths follow the route of the gradient descent optimisation method. Methods to escape such minima are limited and resulting paths can be very suboptimal. MILP can produce paths that are optimise an objective function. It is also able to take vehicle dynamics into account and to be used for multi-robot path planning. However, a sufficiently long time horizon is necessary so that the goal can be reached, because too short a horizon can guide te robot to local minima. Kinodynamic planning occurs under narrower constraints than methods that do not take vehicle dynamics into account. Combinatorial methods are incompatible with kinodynamic planning, because it is inherent to these methods that the roadmap is unique and depends exclusively on the geometry of the configuration space and not on robot dynamics.

2-2 Dynamic environments

Dynamic environments contain moving or unknown obstacles and need other methods to be dealt with. The timing method and direct method use methods for static environments for dynamic planning. Reactive methods change their representation of the free space and their plan when a change occurs in the environment. Corridor methods reserve an area rather than a path.

2-2-1 Static methods

The timing method assumes that trajectories of obstacles are known. Robots follow paths calculated using a method for static environments while adjusting their speed to avoid collisions. This can make paths very suboptimal since in many cases it is much faster to move around a dynamic obstacle than to wait for it to move. The direct method adds the time dimension to search space. Because computation time for this method increases exponentially with the dimensions of the search space, it is slower than the timing method [45].

2-2-2 Reactive planning

With replanning a new plan is made when the environment changes. D* is an example of a replanning method that uses the A* graph search algorithm that allows for the costs of traversing edges to change during the search [41]. An approach for dealing with dynamic obstacles using probabilistic roadmaps is to locally update the roadmap [22]. However, both these methods do not take the predicted movement of obstacles into account. Reconfigurable random forests are an implementation of rapidly exploring random trees that discard nodes and edges of the tree as they are found to lead to collisions [35]. The result is that the path gets separated into parts that have to be reconnected, which is a complicated procedure because for a dynamically constrained robot it is hard to exactly connect two points that are each other. Reactive planning uses local information about obstacles instead of a global map, which is a useful property for situations where only local information is available. A reactive planning method that uses velocity obstacles, which are first-order approximations of the robot velocities resulting in collision, is proposed in [14]. However, this method has the problem of getting stuck in local minima like the potential field method. Model predictive control is a methodology to optimise control actions over a time horizon by predicting system behaviour with a model. Every time step the optimal control law is determined that minimises a cost function over a defined time horizon. Only the first control action is executed, after which the procedure is repeated to determine the next control action. The Dynamic Window Approach proposed in [15] reduces overall computation time for long paths by effectively splitting it into parts. However, as a result no global optimum is guaranteed.

2-2-3 Corridors

Corridor planning entails planning a path that is wider than strictly necessary when only considering the size of the robot [8]. In this way, the robot follows the general direction of the initial plan and can solve local conflicts with obstacles without the need for major replanning [23, 44]. However, replanning may be necessary anyway when a corridor is blocked. Furthermore, corridors can cause other robots to take suboptimal paths when they are too wide, while narrow corridors make the problem more constrained.

2-2-4 Discussion

Using static methods for dynamic environments leads to very suboptimal paths, because an avoidance manoeuvre can save a lot of time compared to waiting for an obstacle to disappear. Reactive methods can get stuck in local minima because only local information about obstacles is used. Using corridors is a way of separating global path planning from local obstacle avoidance, but their applicability depends heavily on the structure of obstacles in the environment.

2-3 Multiple robots

Multiple robots increase of dimensions of the solution space, assuming robots have equivalent configuration spaces. This makes many of the methods described before computationally expensive, because worst-case computation time scales at least exponentially with the dimension of the search space [44]. Therefore, other approaches may be required for multi-robot path planning. Planning techniques can be divided into decentralised methods that plan for different robots independently and centralised methods that plan for multiple robots at the same time.

2-3-1 Decentralised planning

In order to prevent collisions between robots paths can be planned for individual robots in a fixed order [47]. Each subsequent robot treats previous robots as obstacles. The main advantage of this approach is its simplicity. This method is neither optimal nor complete, as the problem can become infeasible depending on the way robots are ordered. Reciprocal velocity obstacles are used to share the responsibility of collision avoidance between the two robots involved, which is a decentralised planning method [46]. However, for an increasing number of robots this methodology can result in oscillations of robot movements due to robots using the same decision rules. Furthermore, it is a linear method that can only account for linear robot dynamics. Optimal reciprocal collision avoidance works to eliminate these oscillations by allowing robots to choose from a set of possible relative velocities, selecting the one that matches its target velocity the closest [48]. However, this method is limited to linear dynamics and to homogeneous systems, where all robots have the same equations of motion.

2-3-2 Centralised planning

Subdimensional expansion is a method proposed to counter the exponential growth of the configuration space with the number of robots that occurs when using regular methods, designed for single robots, for multi-robot path planning [50]. The algorithm M* generates a path for each robot separately using the A* graph search algorithm. Next, paths are checked for collisions between robots. A new higher-dimensional search space is created to generate a collision-free path for the two robots. This method combines centralised and decentralised approaches, only using the centralised approach when a collision would occur. This has the advantage of reduced computation time that comes with decentralised approaches, and of being able to capture better solutions that come with centralised approaches. However, in the worst case when each robots would collide with another the higher-dimensional search space becomes the same size as the joint configuration space of all robots, which destroys the benefit if using the method.

2-3-3 Discussion

Centralised approaches for multi-robot path planning quickly become infeasible as the number of robots increases, because worst-case computation time scales exponentially with the number of robots. A decentralised approach on the other hand covers only a small part of the solution space, so that a globally optimal solution cannot be guaranteed. A hybrid method such as subdimensional expansion can reduce computation time by applying both approaches, but depending on the physical environment it can become equivalent to fully centralised approach with long computation time.

2-4 Task scheduling

Task scheduling is used for different kinds of problems, from vehicle routing to project management. Prioritisation methods assign tasks based on their priorities. Time window methods start with a set of deadlines and compute possible solutions to fit those deadlines. Optimisation methods minimise an objective function to generate an optimal schedule.

2-4-1 Prioritisation

In [43], an algorithm is developed that uses prioritisation to assign tasks to robots and plan their respective paths, so that the outcome minimises a global cost function. All robots are interchangeable, so they are not considered to carry a specific payload. The algorithm proceeds in four steps, starting with a set of goal configurations to be reached by the robots. First, individual paths are planned from all robots to all goals. Second, goal configurations are assigned to robots by minimising a cost function, such as the length of the longest path. Third, robots are ordered by priority, using the following two rules, where R is the radius of the circular robot. Given a set of robots, for every two robots $\mathcal{A}^i$ and $\mathcal{A}^j$ in the set:

1. $\mathcal{A}^i$ has priority of $\mathcal{A}^j$ if the path of $\mathcal{A}^i$ comes within $2R$ of the goal configuration of $\mathcal{A}^j$.
2. $\mathcal{A}^j$ has priority of $\mathcal{A}^i$ if the path of $\mathcal{A}^i$ comes within $2R$ of the initial configuration of robot $\mathcal{A}^j$.

Fourth, using the previously determined priorities, speed profiles are calculated for the robots along their trajectories, so that collisions are avoided. For prioritisation methods, the order of the priority queue has large consequences for the algorithms performance. Several heuristics have been proposed to order the queue effectively. One method is to order the queue by *query distance*, which is the estimated time duration of following the path, if other robots would not need to be avoided [47]. This heuristic aims to minimise the latest arrival time, given a set of tasks, by allowing robots travelling longer distances and hence requiring more time, to go first.

2-4-2 Time windows

In [10], algorithms are proposed for path planning with time windows consisting of departure and arrival deadlines. A directed graph called a *time window network* is defined. Several properties of nodes and edges are calculated for the graph, such as earliest arrival time and latest departure. The resulting data on the nodes can be used to plan valid paths for a robot under time window constraints. An algorithm is proposed that gives the most *flexible* path, i.e. the path that is most robust to delays while still meeting the time window requirements. The algorithm could be generalised to compute flexible paths for multiple robots. An auction-based method for routing multiple robots under time window constraints is discussed in [37]. Robots bid on tasks, offering their cost of fulfilling it, based on factors such as the distance to travel. The lowest bid wins and the task is assigned to the robot. During execution when more information becomes available, auctions are re-run which can cause changes to the schedule. The disadvantage of such a decentralised method is that the optimality of the schedule depends on factors such as the order in which tasks are assigned. To prevent robots never completing their task, a weighted cost on its duration can be used.

2-4-3 Optimisation

The travelling salesman problem is an optimisation problem, where the goal is for a salesman to visit a set of cities at least once, and return to his city of origin, all the while minimising travelling costs. The problem can of course be generalised for other problems, such as robots visiting warehouse storage racks for example. The multiple travelling salesmen problem is an extension that allows for multiple agents to travel between cities at the same time [5]. Exact solution methods exist that work for limited numbers of agents and nodes, using approaches like integer linear programming [24], or the cutting-plane method [30]. In addition, heuristic methods that do not guarantee optimal solutions but are more computationally efficient have been proposed, based on genetic algorithms [42] or neural networks [38] for example. The resource-constrained project scheduling problem involves activities and resources, which are available in limited quantities. Processing an activity requires time and resources. In [27, 2], an event-based MILP formulation is proposed to assign activities to resources. Instead of sampling time into fixed time steps, the event-based model defines a fixed number of events, one for the start and finish of each task. The goal of the optimiser is to assign these events to tasks and determine the time at which they occur. The resource-constrained project scheduling problem is related to multi-robot scheduling, since robots can be considered resources. Task scheduling and path planning are used together in [1]. A MILP formulation based on the Travelling Salesman Problem assigns tasks to agents. The potential field method is then used check if a robot can feasibly fulfil a task without collisions. However, collision avoidance between robots is not taken into account. Furthermore, the potential fields can have local minima so that paths could be rejected even though they are feasible.

2-4-4 Discussion

With prioritisation, optimising the order of priority queue can have major consequences on the overall performance. Time windows that are used in some situations give rise to the possibility of optimising for flexibility and robustness to delays, rather than time or distance for example. Various solution methods for optimisation approaches exist, such as exact solutions of a linearisation of the problem, heuristics that are not optimal but fast to compute and integer programming. A MILP formulation uses integer and binary variables to model the on/off behaviour characteristic to scheduling. Though linear solvers are very efficient, it can involve a lot of additional variables and constraints to transform a problem into MILP form.

2-5 Conclusion

The literature on path planning is vast and many methods have been developed over the years. Task scheduling is a problem that arises in various forms, such as vehicle routing or project management. However, from the literature survey it is concluded that not enough research is done to investigate the integration of path planning and task scheduling, treating them as a problem with a single objective to be optimised. In this thesis, new methods are developed with that objective in mind.

Chapter 3

Multi-robot path planning

In this chapter the general path planning problem is formulated and a Mixed-Integer Linear Programming (MILP) formulation from the literature is discussed [40]. Additionally, implementation challenges are addressed and a new cost function is proposed to optimise the duration of paths.

A linear formulation is used to reduce computation time. Hence, nonlinearities in the system model need to be linearised. The main causes of nonlinearity are the following.

- In a two-dimensional plane, Euclidean distances are nonlinear functions of x and y coordinates.
- Similarly, a maximum velocity or acceleration is a nonlinear function of x and y coordinates.
- Mapping the turning radius and velocity of a vehicle to a two-dimensional Euclidean coordinate system requires nonlinear geometric functions.
- Non-polygonal obstacles are described by nonlinear functions, therefore obstacles need to be represented as polygonals.

Solutions to remedy these causes of nonlinearity are covered in this chapter.

3-1 Problem description

Multi-robot path planning aims to find paths for a set of robots from their initial to their goal configurations, while avoiding obstacles. Let $\mathcal{A}$ be the set of P robots, where the pth robot is denoted by $\mathcal{A}_p$. The set $\mathcal{O}$ contains individual obstacles $\mathcal{O}_m$, with $m = 1, \ldots, M$. Time is discretised into time instances $k = 1, \ldots, K$. The variables that characterise the problem are divided into three categories, namely (i) given, (ii) state and (iii) decision variables.

Given variables are conditions on the problem describing its initial state and desired outcome. For the path planning problem, they are the following:

- For each robot $\mathcal{A}_p$, an initial state $\boldsymbol{s}_p^{\mathrm{i}}$ and a goal state $\boldsymbol{s}_p^{\mathrm{g}}$.
- Position and geometry of obstacles in $\mathcal{O}$. Assuming nonrotating rectangular obstacles (See Section 3-4-1 for motivation), each obstacle $\mathcal{O}_m$ can be described by its lower-left and upper-right coordinates, $\{x_m^{\min}, y_m^{\min}, x_m^{\max}, y_m^{\max}\}$. For dynamic obstacles, these variables depend on time, so they become $\{x_{mk}^{\min}, y_{mk}^{\min}, x_{mk}^{\max}, y_{mk}^{\max}\}$.

State variables contain information about the state of the system throughout time. Their values are subject to the solution of the optimisation problem. They are:

- Position of each robot $\mathcal{A}_p$, expressed by its coordinates $\{x_{pk}, y_{pk}\}$.
- Velocity of each robot, $\{\dot{x}_{pk}, \dot{y}_{pk}\}$.

Decision variables form the solution to the optimisation problem. The decision variables for the path planning problem are:

- Control actions applied to robot p at time instance k, producing a path for each robot: $\boldsymbol{u}_{pk}$.

3-2 Trajectory optimisation for a single robot

A linear-quadratic regulator is a classic approach to optimal control of dynamic systems. Control actions are calculated by minimising a quadratic cost function J. The optimisation is constrained by the linear state space equations of the dynamic system. Its general formulation in continuous time is [18]:

$$\begin{aligned} \min_{\boldsymbol{s},\boldsymbol{u}} \quad J = \min_{\boldsymbol{s},\boldsymbol{u}} \quad & \int_0^\infty (\boldsymbol{s}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{s} + \boldsymbol{u}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{u})dt \\ \text{s.t.} \quad & \dot{\boldsymbol{s}} = \boldsymbol{A}_{\mathrm{c}}\boldsymbol{s} + \boldsymbol{B}_{\mathrm{c}}\boldsymbol{u} \end{aligned} \tag{3-1}$$

where $\boldsymbol{s}$ is the state vector and $\boldsymbol{u}$ is the control vector. For the example of a single robot, the state vector could contain the position and velocity of the robot and the control vector could contain the current to an electric motor. $\boldsymbol{A}_c$ and $\boldsymbol{B}_c$ are continuous time state space matrices. $\boldsymbol{Q}$ and $\boldsymbol{R}$ are semi-positive definite weighting matrices for penalising the state and control actions. An initial condition determines the state at time $t = 0$.

It is possible to use a 1-norm in the cost function instead of a 2-norm, resulting in a linear optimisation. This means that only linear combinations of optimisation variables can be used in the objective function, prohibiting the use of performance measures that are nonlinear. However, in combination with integer variables that are used to handle collision avoidance with obstacles, linear problems are generally faster to solve than mixed-integer quadratic programs. The cost function using a 1-norm takes the following form:

$$J = \int_0^\infty (\boldsymbol{q}^{\mathrm{T}}|\boldsymbol{s}| + \boldsymbol{r}^{\mathrm{T}}|\boldsymbol{u}|)dt \tag{3-2}$$

where $\boldsymbol{q}$ and $\boldsymbol{r}$ are weighting vectors.

The next step is to discretise the system so the problem can be solved numerically. Time is divided uniformly into time steps $k = 0, \ldots, K-1$ over a finite time horizon of $K\Delta t$ seconds, where Δt is the sampling time. To account for cost accumulated beyond the time horizon, a terminal cost $\boldsymbol{p}^{\mathrm{T}}\boldsymbol{s}_K$ is added to the objective function. A large terminal cost encourages the solver to find a path so that the goal is reached. Rather than minimising the state vector itself, the difference between the state vector $\boldsymbol{s}_k$ and the goal state $\boldsymbol{s}_{\mathrm{g}}$ is minimised. The cost is minimal when the state of the robot equals the goal state, which can be a certain goal location at zero velocity for example. This results in the following formulation [40]:

$$\begin{aligned} \min_{\boldsymbol{s},\boldsymbol{u}} \quad J = \min_{\boldsymbol{s},\boldsymbol{u}} \quad & \sum_{k=0}^{K-1} (\boldsymbol{q}^{\mathrm{T}}|\boldsymbol{s}_k - \boldsymbol{s}_{\mathrm{g}}| + \boldsymbol{r}^{\mathrm{T}}|\boldsymbol{u}_k|)\Delta t + \boldsymbol{p}^{\mathrm{T}}\boldsymbol{s}_K \\ \text{s.t.} \quad & \boldsymbol{s}_{k+1} = \boldsymbol{A}\boldsymbol{s}_k + \boldsymbol{B}\boldsymbol{u}_k \end{aligned} \tag{3-3}$$

where $\boldsymbol{s}_k$ denotes the state vector at time instance k and $\boldsymbol{u}_k$ is the control vector at time instance k. Matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are the discretised state space matrices of the linear system.

To solve a MILP problem using a solver such as MATLAB or Gurobi, the problem needs to be brought into the standard form:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{A}_{\mathrm{ineq}}\boldsymbol{x} \leq \boldsymbol{b}_{\mathrm{ineq}} \\ & \boldsymbol{A}_{\mathrm{eq}}\boldsymbol{x} = \boldsymbol{b}_{\mathrm{eq}} \\ & x_i \in \mathbb{Z} \quad \forall \quad i \in X_{\mathrm{int}} \end{aligned} \tag{3-4}$$

where $\boldsymbol{c}$ is the weighting vector of the solution vector $\boldsymbol{x}$ in the objective function, $\boldsymbol{A}_{\mathrm{ineq}}$, $\boldsymbol{b}_{\mathrm{ineq}}$, $\boldsymbol{A}_{\mathrm{eq}}$ and $\boldsymbol{b}_{\mathrm{eq}}$ are constraint matrices and X_{int} is the set of indices of integer elements of $\boldsymbol{x}$.

To reach this standard form, the absolute values in the cost function are eliminated using slack variables $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$, which have the same dimensions as $\boldsymbol{x}_k$ and $\boldsymbol{u}_k$ respectively. The initial condition $\boldsymbol{s}_0$ and the sampling time Δt can be omitted from the objective function of equation (3-3), since they are constant. Finally, the weighting vectors $\boldsymbol{q}$ and $\boldsymbol{r}$ are indexed with time. This allows to increase the penalty as time goes on, which prevents the solver from spending a disproportionate amount of its efforts on the part of the path farthest from the goal. The MILP formulation for trajectory optimisation of a single vehicle becomes:

$$\begin{aligned} \min_{\boldsymbol{w}_k,\boldsymbol{v}_k} \quad & \sum_{k=1}^{K-1} \boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{w}_k + \sum_{k=0}^{K-1} \boldsymbol{r}_k^{\mathrm{T}}\boldsymbol{v}_k + \boldsymbol{p}^{\mathrm{T}}\boldsymbol{w}_K \\ \text{s.t.} \quad & \boldsymbol{s}_k \leq \boldsymbol{w}_k + \boldsymbol{s}_{\mathrm{g}} \\ & -\boldsymbol{s}_k \leq \boldsymbol{w}_k - \boldsymbol{s}_{\mathrm{g}} \\ & \boldsymbol{u}_k \leq \boldsymbol{v}_k \\ & -\boldsymbol{u}_k \leq \boldsymbol{v}_k \\ & \boldsymbol{s}_{k+1} = \boldsymbol{A}\boldsymbol{s}_k + \boldsymbol{B}\boldsymbol{u}_k \end{aligned} \tag{3-5}$$

3-3 Linear vehicle models

The formulation of (3-5) leaves the choice for the vehicle model open, as long as it is linear. First, a simple linear vehicle model is introduced. Next, it is expanded to become more realistic for use in simulations.

3-3-1 Simple vehicle model

An example of a simple linear discrete-time vehicle model is one that produces grid-like paths. The state consists of the x and y coordinates of the robot. Control actions are constrained by $-1 \leq \boldsymbol{u}_k \leq 1$, and simply move the robot in the x or y direction by their amounts. The discrete-time state space equation is:

$$\underbrace{\begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix}}_{\boldsymbol{s}_{k+1}} = \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}}_{\boldsymbol{A}} \underbrace{\begin{pmatrix} x_k \\ y_k \end{pmatrix}}_{\boldsymbol{s}_k} + \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}}_{\boldsymbol{B}} \underbrace{\begin{pmatrix} u_{\mathrm{x},k} \\ u_{\mathrm{y},k} \end{pmatrix}}_{\boldsymbol{u}_k} \tag{3-6}$$

Unit penalty is assigned to the state, and no penalty is assigned to the control actions, both independent of time. The initial and goal position are set to $\boldsymbol{s}_0^{\mathrm{T}} = \begin{pmatrix} 4 & 7 \end{pmatrix}$ and $\boldsymbol{s}_{\mathrm{g}}^{\mathrm{T}} = \begin{pmatrix} 0 & 0 \end{pmatrix}$.

Figure 3-1 shows a solution from running the optimisation with Gurobi. Note that though the solver returns a single solution, for this example there exist multiple solutions with the same cost.

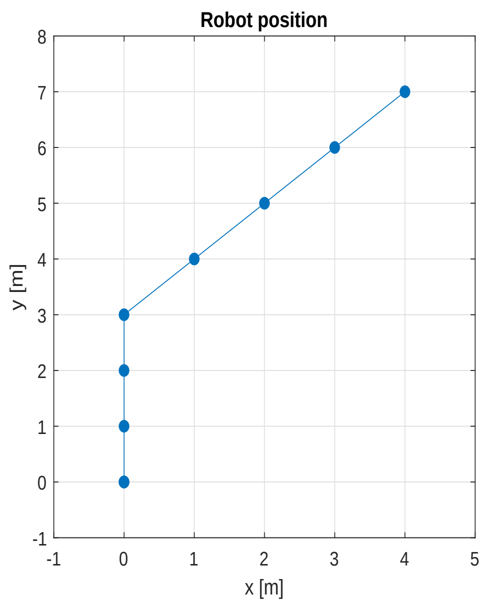


Figure 3-1: An optimal trajectory for the simple linear model, with unit weights $\boldsymbol{q}$ and $\boldsymbol{r}$, initial and goal states $\boldsymbol{s}_0^{\mathrm{T}} = (4 \quad 7)$ and $\boldsymbol{s}_{\mathrm{g}}^{\mathrm{T}} = (0 \quad 0)$. The blue line represents the trajectory and the dots represent the position of the robot at sampling instances.

3-3-2 Force control robot model

A more realistic linear vehicle model is the following continuous-time second-order model that uses force control:

$$\begin{aligned} m\ddot{x} + F_x &= 0 \\ m\ddot{y} + F_y &= 0 \end{aligned} \tag{3-7}$$

where m [kg] is the vehicle mass, x and y [m] are the coordinates of the robot, and F_x and F_y [N] are the input forces exerted on the robot.

The state space equations for this model are:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ -\frac{1}{m} & 0 \\ 0 & -\frac{1}{m} \end{pmatrix} \begin{pmatrix} u_x \\ u_y \end{pmatrix} \tag{3-8}$$

where u_x and u_y are the input forces F_x and F_y respectively. For numerical optimisation the state matrices are transformed into their discrete-time equivalents.

The vehicle is treated as a point mass that can be accelerated in all directions. Constraints are used to limit the force actuation and velocity of the robot. This model applies quite well to the Prodrive robot, which can rotate in place. This means it can follow any path in two-dimensional space.

In reality the permitted acceleration might depend on the velocity, but this is a nonlinearity that cannot be captured by a linear model. Safe bounds on the force actuation and velocity of the planner can be used to make sure the real robot can follow the simulated path. The alternative of using a turning radius or angular orientation results in a nonlinear vehicle model.

Another shortcoming of this linear model is that the constraints on the force actuation and velocity allow the robot to move and accelerate faster in the x and y direction at the same time, than in just the x or y direction. This is because a velocity or acceleration bound that is equal in all directions is nonlinear:

$$\sqrt{\dot{x}^2 + \dot{y}^2} \leq v_{\max} \qquad \sqrt{\ddot{x}^2 + \ddot{y}^2} \leq a_{\max} \tag{3-9}$$

where $V_{\max}$ is the velocity bound and $A_{\max}$ is the acceleration bound.

To mitigate this shortcoming, the nonlinear 'circular' bounds can be approximated using multiple linear constraints, as shown in Figure 3-2. More constraints can be added to approximate the circle more accurately. The same goes for constraints on the acceleration.

Since adding constraints increases complexity and computation time, the simplest form involving four constraints is used from here on, with safe bounds so no infeasible solutions are produced. The disadvantage of this approach is that the solution space is reduced, i.e. better solutions with velocities that fall outside of the simple bounds but are actually feasible are not found.

An example using this vehicle model would simply result in a straight line from the starting point of the robot to its goal. Therefore, an example is shown after the modelling of obstacles has been introduced.

3-4 Obstacles

3-4-1 Obstacle constraints

It is intuitive to regard obstacles as constraints on the path planning problem, since they constrain the area in which the robot is able to move. To simplify the problem formulation,

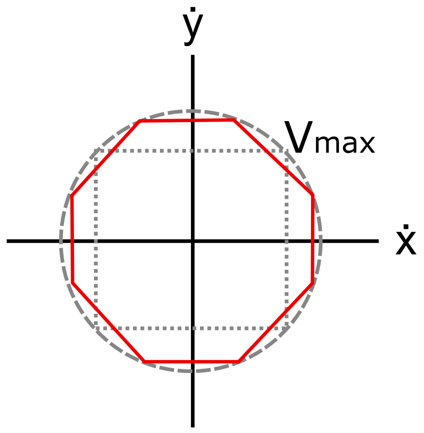


Figure 3-2: The dashed circle represents the most realistic, but nonlinear, maximum velocity v_{max} which is equal in all directions. The red hexagon shows a linear approximation of this circle. The dotted square shows a rectangular approximation requiring only four constraints.

the robot is treated as a point mass. To prevent collisions, obstacles are enlarged in all directions by the radius of the robot. Doing so with the maximum radius of a noncircular robot reduces the solution space because some robot positions are feasible depending on the orientation of the robot. Since the base of the Prodrive robot is circular, treating it as a point poses no problems. The base is designed to be circular for safety reasons, namely to prevent collisions while rotating in place, but also to simplify path planning, because then collisions due to rotation need not be considered.

Consider a rectangular obstacle $\mathcal{O}_m$ at time instance k, with its lower left corner at $(x_{mk}^{\min}, y_{mk}^{\min})$ and its upper right corner at $(x_{mk}^{\max}, y_{mk}^{\max})$. For a static obstacle these coordinates are the same for all k. At every point in time k, the position of the robot must not lie within the rectangle. This can be formulated using four constraints, of which only one needs to be met. The constraints are as follows:

$$\begin{aligned} \forall k \in \{1, \ldots, K\}, \quad \forall m \in \{1, \ldots, M\}: \qquad & x_k \leq x_{mk}^{\min} \\ \text{or} \quad & x_k \geq x_{mk}^{\max} \\ \text{or} \quad & y_k \leq y_{mk}^{\min} \\ \text{or} \quad & y_k \geq y_{mk}^{\max} \end{aligned} \tag{3-10}$$

These *or*-constraints need to be transformed into regular *and*-constraints for them to adhere to the MILP standard form of (3-4). The *big M* method can be used to do so [6]. Because M already denotes the number of obstacles, C will be used instead. Consider two binary variables $\delta_1 \in \{0, 1\}$ an $\delta_2 \in \{0, 1\}$. One can ensure that at most one of them is 1 by requiring $\delta_1 + \delta_2 \leq 1$.

Now consider the inequality $f(x) \leq 0$ and a binary variable $\delta \in \{0, 1\}$. It can be verified that $[f(x) \leq 0 \;\text{ or }\; \delta = 1]$ if and only if $f(x) \leq C\delta$, where $C \geq \max(f(x))$. An estimate of C closer to the actual value of $\max(f(x))$ is computationally more efficient than a larger value [51].

Combining these two notions, two *or*-constraints can be transformed into *and*-constraints. Consider the following two constraints, of which only one needs to be met:

$$f(x) \leq 0 \qquad \text{or} \qquad g(x) \leq 0 \tag{3-11}$$

These are readily transformed into standard form using the two notions inferred above:

$$\begin{aligned} f(x) \leq&\quad C_f\delta_1 \\ g(x) \leq&\quad C_g\delta_2 \\ \delta_1 + \delta_2 \leq&\quad 1 \end{aligned} \tag{3-12}$$

where $C_f \geq \max(f(x))$, $C_g \geq \max(g(x))$, $\delta_1 \in \{0,1\}$ and $\delta_2 \in \{0,1\}$.

Applying this method to (3-10), binary variables $\delta_{mk1}^{\text{obs}}, \ldots, \delta_{mk4}^{\text{obs}} \in \{0,1\}$ are introduced for all obstacles and time instances. Binary variables are implemented for a MILP solver by designating them as integer variables and constraining them between 0 and 1. Transformed into standard form, (3-10) becomes:

$$\begin{aligned} &\forall k \in \{1,\ldots,K\}, \quad \forall m \in \{1,\ldots,M\}: \\ x_k &\leq \quad x_{mk}^{\min} \quad + C\delta_{mk1}^{\text{obs}} \\ -x_k &\leq -x_{mk}^{\max} \quad + C\delta_{mk2}^{\text{obs}} \\ y_k &\leq \quad y_{mk}^{\min} \quad + C\delta_{mk3}^{\text{obs}} \\ -y_k &\leq -y_{mk}^{\max} \quad + C\delta_{mk4}^{\text{obs}} \\ \delta_{mk1}^{\text{obs}} + \delta_{mk2}^{\text{obs}} + \delta_{mk3}^{\text{obs}} + \delta_{mk4}^{\text{obs}} &\leq \quad 3 \end{aligned} \tag{3-13}$$

where C is at least two times greater than maximum values for x and y coordinates. Note that x_k and y_k are variables in the state vector $\boldsymbol{s}_k$, while $x_{mk}^{\min}$, $x_{mk}^{\max}$, $y_{mk}^{\min}$ and $y_{mk}^{\max}$ are constants deriving from the obstacles.

The obstacle constraints of (3-13) only apply to rectangular obstacles that are not rotated with respect to the x and y axes. Any polygonal obstacle, i.e. with straight edges, can similarly be captured by linear constraints [40]. However, more constraints are needed depending on the shape of the obstacle. Furthermore, a binary variable is needed for every added constraint. To reduce complexity, only unrotated rectangular obstacles are used from here on. Obstacles of other shapes can be approximated up to a certain resolution using smaller rectangles. Additionally, the layout of a typical production environment at Prodrive is adequately captured with rectangles. In theory, regarding robots as points and enlarging obstacles would result in round corners where the original obstacle has a square corner. Instead, obstacles are enlarged more than strictly necessary, by the same amount in the x and y direction so that obstacles remain rectangles.

3-4-2 Unknown obstacle trajectories

The constraints of (3-13) require the position of obstacles over the entire time horizon. For some (moving) obstacles, this information may not be available. One solution to this problem is to use Model Predictive Control (MPC). With MPC, a path is planned over a prediction horizon that is shorter than the full time horizon necessary to reach the goal. The obstacle positions then only need to be known or predicted for the duration of the prediction horizon. After a path is calculated, the robot starts following it and in the meantime a path is planned with a new prediction horizon. This step is repeated until the goal is reached. Another solution that can be used in addition to MPC, is to use the constraints from (3-13) to block a region that is larger than the actual obstacle. The size of this region can be increased over time to account for increasing uncertainty about the location of the obstacle [44]. Disadvantageous about this solution is that it may render the problem infeasible.

3-4-3 Corner cutting

The obstacle constraints of (3-13) ensure that robots do not collide with obstacles at every time instance k. However, this is not guaranteed for the line connecting these points, because the time of the system is discrete. As a result, the robot may cut corners of obstacles (Figure 3-4a). To prevent this, obstacles are enlarged in all directions by d_{grow}.

The maximum distance travelled between two states is $d_{\max} = v_{\max} t_{\text{s}}$, where $v_{\max}$ is the maximum speed of the vehicle and t_{s} is the sampling time. To prevent corner cutting of 90° corners, obstacles are grown in all directions by $d_{\text{grow}} = \frac{1}{4}\sqrt{2} \cdot d_{\max}$, as shown in Figure 3-3.

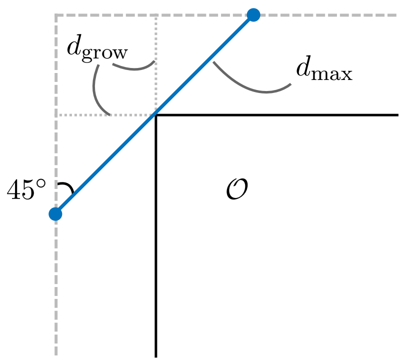


Figure 3-3: An obstacle $\mathcal{O}$ with 90°corners is grown by d_{grow} to prevent corner cutting. The blue line indicates the maximum distance $d_{\max}$ that can be travelled by the robot during one sampling period.

Figure 3-4a shows a path using the vehicle model with force control from Section 3-3-2. Although at all discrete time instances, denoted by points, the robot is not in collision with obstacles, on the connecting lines it clearly is. The dashed lines in Figure 3-4b show the edges of the virtual obstacles that have been enlarged by d_{grow}. The resulting path cuts no corners of the true obstacles.

The disadvantage of this method is that it reduces the solution space, since the robot will not approach obstacles as close as is possible in reality. In some cases this could even render the problem infeasible, even though a solution exists in the original solution space. Furthermore, a large sampling time can make the problem infeasible, or much harder to solve, since the virtual enlargement of obstacles is proportional to the sampling time.

An alternative method is to use additional constraints to ensure the line connecting sampling instances does not overlap with an obstacle [39]. However, these constraints increase complexity and computation time because they introduce new binary variables for every time step and obstacle. To save computation time, the simpler method of enlarging obstacles is used for the remainder of this research.

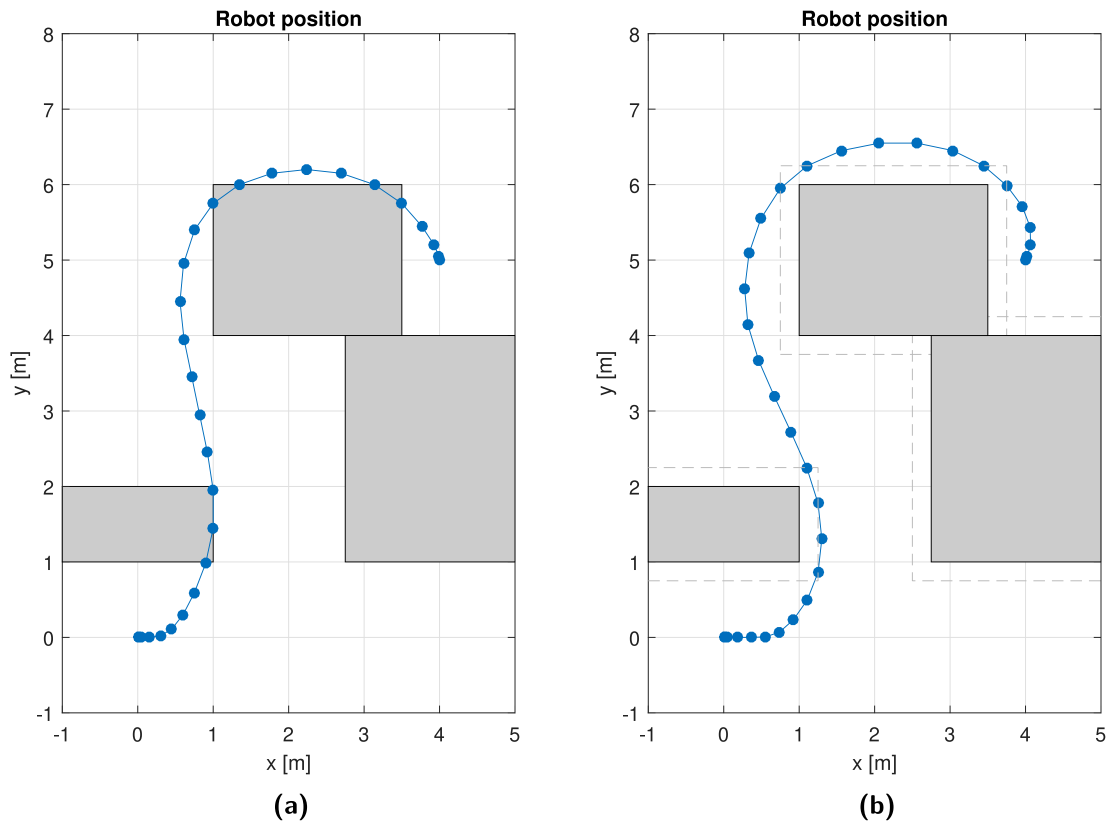


Figure 3-4: The blue lines represent paths and the dots represent the position of the robot at sampling instances. (a) A path cutting obstacle corners, from $(4, 5)$ to $(0, 0)$. (b) By growing the obstacles in all directions with d_{grow}, the corners of the virtual obstacles (dashed lines) are cut, but not those of the actual obstacles.

3-5 Multiple robots

Previous sections accounted for only one robot. This section treats the modifications to the objective function and constraints that are necessary when path planning for multiple robots.

3-5-1 Objective function

To account for multiple robots, the cost function (3-5) needs to be amended. Instead of K states $\boldsymbol{s}_k$ for all time steps, there are $K \times P$ states $\boldsymbol{s}_{pk}$ for all robots $p = 1, \ldots, P$ and time steps $k = 1, \ldots, K$. The same goes for the inputs $\boldsymbol{u}_{pk}$. There are P goal states $\boldsymbol{s}_{pg}$. Weighting vectors $\boldsymbol{q}_{pk}$ and $\boldsymbol{r}_{pk}$ can take different values depending on the robot, for example to give priority to certain robots. The cost function becomes:

$$J = \sum_{p=1}^{P} \left(\sum_{k=1}^{K-1} \boldsymbol{q}_{pk}^{\mathrm{T}} \boldsymbol{w}_{pk} + \sum_{k=0}^{K-1} \boldsymbol{r}_{pk}^{\mathrm{T}} \boldsymbol{v}_{pk} + \boldsymbol{p}_{p}^{\mathrm{T}} \boldsymbol{w}_{pK} \right) \tag{3-14}$$

3-5-2 Constraints

The constraints governing the movement of robots according to the vehicle model can be made dependent on the robot as well. Different state matrices $\boldsymbol{A}_p$ and $\boldsymbol{B}_p$ can be used for robots, if they are not homogeneous. However, the remainder of this report will only consider homogeneous robots.

$$\begin{aligned} \forall p, \forall k: \quad \boldsymbol{s}_{pk} \leq &\ \boldsymbol{w}_{pk} + \boldsymbol{s}_{pg} \\ -\boldsymbol{s}_{pk} \leq &\ \boldsymbol{w}_{pk} - \boldsymbol{s}_{pg} \\ \boldsymbol{u}_{pk} \leq &\ \boldsymbol{v}_{pk} \\ -\boldsymbol{u}_{pk} \leq &\ \boldsymbol{v}_{pk} \\ \boldsymbol{s}_{p(k+1)} = &\ \boldsymbol{A}_p \boldsymbol{s}_{pk} + \boldsymbol{B}_p \boldsymbol{u}_{pk} \end{aligned} \tag{3-15}$$

Obstacle constraints apply to all robots in the same way, so they are simply repeated for each robot. However, simultaneously optimising for multiple robots can also result in collisions between robots. Therefore additional constraints are needed for collision avoidance. This can be achieved by requiring a minimum distance between robots, using the following constraint:

$$\forall k, \forall p, q \quad | \quad q > p: \quad \begin{aligned} &|x_{pk} - x_{qk}| \geq d_x \\ \text{or} \quad &|y_{pk} - y_{qk}| \geq d_y \end{aligned} \tag{3-16}$$

where d_x and d_y are the minimum distance between robots in the x and y direction. The condition $q > p$ ensures constraints are not duplicated.

Recall from the force control vehicle model (Section 3-3-2) that a constraint on the velocity that is equal in all directions is nonlinear. Similarly, the absolute distance $|d|$ between two robots A and B is a nonlinear function of their positions:

$$|d| = \sqrt{(x_\text{A} - x_\text{B})^2 + (y_\text{A} - y_\text{B})^2} \tag{3-17}$$

This nonlinear relation can be approximated using linear constraints. The simplest approximation involving the fewest number of constraints is used, namely that of (3-16). This approximation is like the dotted square on the velocity constraint of Figure 3-2.

To bring (3-16) into standard form, absolute values are eliminated, and the Big M method (using the letter C as in Section 3-4-1) is applied to transform *or*-constraints into regular constraints. It is assumed that minimum distances between robots are the same in the x and y direction, i.e. $d = d_x = d_y$:

$$\forall k, \forall p, q \quad | \quad q > p: \quad \begin{aligned} x_{qk} - x_{pk} - C\delta^\text{col}_{pqk2} &\leq -d \\ x_{pk} - x_{qk} - C\delta^\text{col}_{pqk1} &\leq -d \\ y_{qk} - y_{pk} - C\delta^\text{col}_{pqk4} &\leq -d \\ y_{pk} - y_{qk} - C\delta^\text{col}_{pqk3} &\leq -d \\ \delta^\text{col}_{pqk1} + \delta^\text{col}_{pqk2} + \delta^\text{col}_{pqk3} + \delta^\text{col}_{pqk4} &\leq 3 \end{aligned} \tag{3-18}$$

where $\delta^\text{col}_{pqk1}, \ldots, \delta^\text{col}_{pqk4}$ are binary variables. C must be a large number, at least two times the largest value for x or y, plus d.

3-5-3 Minimum distance

For circular robots with radii r, the minimum distance between robots to prevent collisions should be:

$$d = 2r + d_{\text{safety}} \tag{3-19}$$

where d_{safety} is an extra safety distance to have between robots.

However, this minimum distance prevents collisions at discrete sampling moments, but not in between time steps. To ensure no collision occurs between time steps, the minimum distance at sampling moments should be:

$$d = 2\sqrt{2}r + d_{\text{safety}} \tag{3-20}$$

The $2\sqrt{2}$ makes sure the most extreme case, where two robots move alongside each other at a 45° angle does not result in a collision in between sampling instances.

When the sampling time is large enough, robots could seem to jump over one another if their distance is large enough at time steps before and after the jump, similar to corner cutting with obstacles. The simplest case of this happening is when two robots swap places between time steps. For the Prodrive case, this behaviour is of course not possible in reality (for the current version of the robots at least). To prevent this from happening, the minimum distance between robots must take into account the robots maximum velocity $v_{\max}$ and the sampling time t_{s}:

$$d = \max(2\sqrt{2}r, v_{\max}t_{\text{s}}) + d_{\text{safety}} \tag{3-21}$$

3-6 Duration optimisation

The objective function (3-14) proposed in the literature uses the distance between robots and their goal states as a cost metric. Another objective could be to minimise the time it takes for robots to reach their goal states. The authors of [40] propose running the optimisation multiple times with different time horizons, until the shortest time horizon resulting in a feasible path is found. However, this method itself is very time-consuming.

A new cost function is proposed to minimise path durations with a single optimisation. To that end, a binary variable $\delta_{pk}^{\text{goal}} \in \{0, 1\}$ is defined which is 1 if and only if robot p has reached its goal at time instance k, which is considered true when $\boldsymbol{w}_{pk}$ equals zero with a tolerance of $\boldsymbol{\epsilon}$:

$$\forall p, \forall k : \quad \boldsymbol{w}_{pk} \leq \boldsymbol{\epsilon} \quad \leftrightarrow \quad \delta_{pk}^{\text{goal}} = 1 \tag{3-22}$$

This relation can be realised in MILP form using two additional constants, $C_{\min} \triangleq \min(\boldsymbol{w}_{pk} - \boldsymbol{\epsilon})$ and $C_{\max} \triangleq \max(\boldsymbol{w}_{pk} - \boldsymbol{\epsilon})$, of which the definitions are upper and lower bounds respectively [6]. Two constraints per robot and time instance are needed:

$$\forall p, \forall k : \qquad \begin{aligned} \boldsymbol{w}_{pk} + C_{\max}\delta_{pk}^{\text{goal}} &\leq \quad C_{\max} + \boldsymbol{\epsilon} \\ -\boldsymbol{w}_{pk} + (C_{\min} - \boldsymbol{\epsilon})\delta_{pk}^{\text{goal}} &\leq \quad -\boldsymbol{\epsilon} \end{aligned} \tag{3-23}$$

Once a robot has reached its goal, all subsequent $\delta_{pk}^{\text{goal}}$ are 1, so the arrival time instance $k_{p,\text{arrival}}$ of robot p can be inferred as follows:

$$k_{p,\text{arrival}} = \begin{pmatrix} 2 & 3 & \cdots & K \end{pmatrix} \begin{pmatrix} \delta_{p2}^{\text{goal}} - \delta_{p1}^{\text{goal}} \\ \delta_{p3}^{\text{goal}} - \delta_{p2}^{\text{goal}} \\ \vdots \\ \delta_{pK}^{\text{goal}} - \delta_{p(K-1)}^{\text{goal}} \end{pmatrix} \tag{3-24}$$

The objective is to minimise the sum of all arrival times, with an optional weighting factor per robot W_p to be able to prioritise certain robots. A large negative weight on the last binary variable with a large number C_g rewards reaching the goal, since otherwise the cost would be the lowest when robots remain stationary at their initial positions. A term $C_\text{g}P$ is added so that the objective value gives calculates the sum of durations and can be read more intuitively.

$$J = \sum_{p \in \mathcal{A}} \left(W_p k_{p,\text{arrival}} - C_\text{g} \delta_{pK}^{\text{goal}} + C_\text{g} P \right) \tag{3-25}$$

A disadvantage of this time-penalising cost function compared to the distance-penalising cost function (3-14) is that if the time horizon is too short to reach the goal this will result in a very bad solution. The distance-penalising cost function on the other hand produces a path that brings the robot as close as possible to the goal. If partial paths that do not reach the goal are deemed useful, a new cost function combining the two cost functions could be used, penalising both distance and time.

3-6-1 Variables and constraints

The total number of variables and constraints depends on the state dimension D_s, the control action dimension D_u, the number of time steps K, the number of obstacles M and the number of robots P. The state and input dimension for the force control vehicle model are 4 and 2 respectively. Including auxiliary variables for the duration objective function from (3-25), the total number of variables is:

$$2D_sKP + 2D_uKP + 4KMP + 4K\frac{1}{2}P(P-1) + 2KP \tag{3-26}$$

The number of inequality constraints is:

$$2D_sKP + 2D_uKP + 5KMP + 5K\frac{1}{2}P(P-1) + 2D_sKP \tag{3-27}$$

The number of equality constraints is:

$$D_sKP + P \tag{3-28}$$

3-7 Conclusion

The goal of the multi-robot path planning problem is to find paths for multiple robots while avoiding obstacles and collisions. An approach based on a discrete-time linear regulator is used, where the linear state equations form constraints of the optimisation. The objective function minimises the cumulative distances between robots at sampling instances and their goal positions.

A linear vehicle model based on force control is proposed. The state consists of two spatial dimensions and their first order derivatives. This model is chosen because it is relatively simple, but still captures the behaviour of the Prodrive robot well. Bounds on the velocity and acceleration are linearised and a method for approximating more realistic nonlinear bounds is suggested.

Linear constraints are used to model obstacles. To limit the number of additional variables and constraints, obstacles are considered to be rectangular. To prevent robots from cutting corners of obstacles, a minimum clearance depending on the maximum speed of robots and the sampling time is proposed. Collision avoidance between robots is realised by requiring a minimum distance between robots.

A new objective function is introduced to optimise paths for the arrival time of the robot, without having to run the optimisation multiple times. This is realised using integer variables and linear constraints to fit the MILP formulation.

In Chapter 5 this path planner formulation is employed to simultaneously schedule tasks and plan the paths for the robots fulfilling them.

Chapter 4

Task scheduling

The task scheduling problem comprises of assigning tasks to robots and determining the time at which they are to be completed. To allow for a homogeneous combined approach with path planning, the task scheduler is also formulated as a Mixed-Integer Linear Programming (MILP) problem.

Given are a set of tasks, each of which has a departure time window and an arrival time window. The departure time window consist of an earliest departure time and a latest departure time, and the arrival time window consists of an earliest arrival time and a latest arrival time. Figure 4-1 shows a graphical representation of a set of five tasks, where each task is denoted by its own colour. Departure must occur during the departure time window and arrival during the arrival time window. The horizontal axis represents time and tasks are shown vertically. Notice how multiple robots are necessary to fulfil all these tasks on time.

4-1 Scheduling approach

The MILP path planner from the previous chapter uses variables indexed by time. This means that for an increasing time horizon the number of variables increases linearly. An event-based approach on the other hand has the same number of variables regardless of the scheduling horizon, but the number of variables does increase linearly with the number of tasks. It is called an event-based approach because it does not use a fixed sampling time but effectively samples only at the start and finish times of tasks, called events. In [27] and [2], an event-based MILP formulation is proposed for solving the *resource-constrained project scheduling problem.* Binary variables are used for marking events as the start or finish of an activity to be performed, where dependencies between activities are constraints on the problem. In this chapter, the same concept is used to develop a MILP formulation for task scheduling with multiple robots.

Assuming we have a fixed number of tasks N, a fixed number of *events* $\mathcal{E}_e$ can be defined, with e the event number. An event occurs at the start and finish time of each task. Two extra events are added to represent the first event and the final event and are not connected

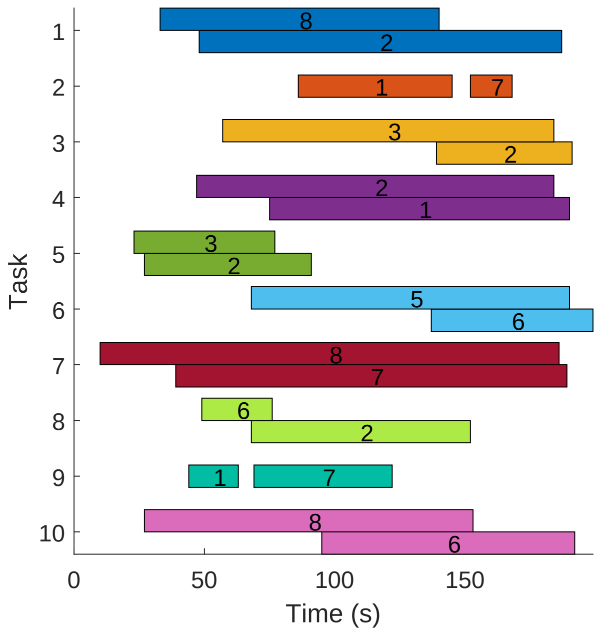


Figure 4-1: Tasks are shown on the vertical axis, each with its own colour and a time windows within which departure and arrival must occur. The departure time window is shown above or to the left of the arrival time window, with station numbers shown on the time windows.

to a specific task, so a reference time can be set and the completion time of the final task can be determined. Therefore a total of $L = 2N + 2$ events are defined, with event numbers $e = 0, \dots, L - 1$.

All events occur in sequential order, i.e. $\mathcal{E}_0$ occurs the earliest and $\mathcal{E}_{L-1}$ the latest. Initially, events are not associated to tasks. It is the goal of the optimisation to determine in which order tasks are fulfilled and therefore also to which events they are to be associated. The time at which events occur is a continuous optimisation variable.

Consider three tasks T_1, T_2 and T_3. Two events per task and two additional reference events are defined, making eight events in total. Figure 4-2 shows an example of a schedule where the start and finish of tasks are assigned to events. The time axis is displayed horizontally. Note that all events occur in sequential order, even though some events occur at the same time. When multiple robots are involved, tasks can be fulfilled at the same time, such as T_1 and T_2. Figure 4-2 shows how the start event of T_2 occurs before the finish event of T_1, which is only possible if multiple robots are available to complete tasks in parallel. The start event of a task need not directly be followed by its finish event, because other events may occur in between. Furthermore, notice how the events are not evenly spaced, the time at which they occur is part of the optimisation solution.

4-2 Problem description

Given are N tasks T_n with $n = 1, \dots, N$ and P robots $\mathcal{A}_p$ with $p = 1, \dots, P$. A set $\mathcal{E}$ is defined containing L events $\mathcal{E}_e$, with $e = 0, \dots, 2N + 1$. In addition to a reference event $\mathcal{E}_0$ and a final event $\mathcal{E}_{L-1}$, an event occurs for the start and finish time of each task.

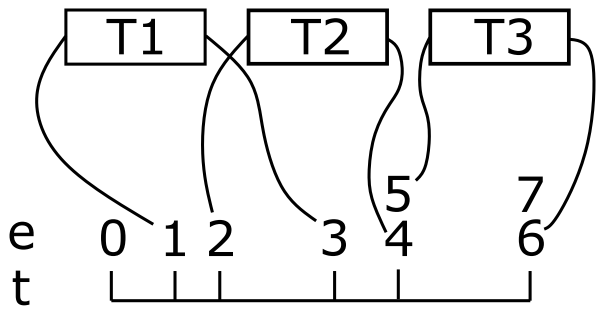


Figure 4-2: Events occur for the start and finish of each task. Events are in sequential order, but can function as a start of finish event of any task. The horizontal axis shows how events (e) can be distributed nonuniformly over time (t), or even occur at the same time, since different tasks can be started and finished at the same time (e.g. events 4 and 5, or 6 and 7).

The decision variables are:

- $s_{nep} \in \{0, 1\}$ is 1 if and only if task n starts at event e and is fulfilled by robot p
- $f_{nep} \in \{0, 1\}$ is 1 if and only if task n finishes at event e and is fulfilled by robot p
- $t_e \in [t_0, \infty)$ is the time at event e, where t_0 is the time at the earliest reference event

4-3 Constraints

The constraints on the MILP optimisation problem are essentially the mathematical description of the scheduling problem. Recall the decision variables s_{nep}, f_{nep} and t_e. The first constraint sets the reference time, which by default is zero. This is done by setting the time of the first event, which is a special event not associated to any task, to zero:

$$t_0 = 0 \tag{4-1}$$

The next constraint ensures that events occur in sequential order:

$$t_{e+1} \geq t_e \quad e = 0, \ldots, L-2 \tag{4-2}$$

Every task is completed only once, therefore the number of times it starts or finishes must be limited to one also:

$$\sum_{e \in \mathcal{E}} \sum_{p \in \mathcal{A}} s_{nep} \leq 1 \quad \forall i \in T \tag{4-3}$$

$$\sum_{e \in \mathcal{E}} \sum_{p \in \mathcal{A}} f_{nep} \leq 1 \quad \forall i \in T \tag{4-4}$$

A task that is started by a robot p, must be completed by the same robot:

$$\sum_{e \in \mathcal{E}} s_{nep} = \sum_{e \in \mathcal{E}} f_{nep} \quad \forall i \in T, \quad \forall p \in \mathcal{A} \tag{4-5}$$

The following constraint is to make sure that tasks are finished after they are started. In other words, the finish event of a task must occur later event than its start event:

$$\sum_{v=0}^{e}\sum_{p\in\mathcal{A}} f_{nvp} + \sum_{v=e}^{L-1}\sum_{p\in\mathcal{A}} s_{nvp} \leq 1 \quad \forall n \in T, \quad \forall e \in \mathcal{E} \tag{4-6}$$

The next constraint keeps track of time. The time at which a finish event for a task occurs must be greater than the sum of the start time of that task and its duration. It is an inequality constraint because it may be beneficial to the schedule to take longer than the minimum duration of the task.

$$t_r \geq t_q + \sum_{p\in\mathcal{A}} (s_{nqp} + f_{nrp}) d_n - d_n \quad \forall q \in \mathcal{E}, r \in \mathcal{E}, r > q, \quad \forall n \in T \tag{4-7}$$

where d_n is the duration of completing task n, including loading and unloading of the delivery.

If two tasks are fulfilled by the same robot, time for reaching the starting point of the second task from the finishing point of the first task must be accounted for:

$$t_r \geq t_q + \sum_{p\in\mathcal{A}} (s_{jrp} + f_{iqp}) d_{ij} - d_{ij} \quad \forall q \in \mathcal{E}, r \in \mathcal{E}, r > q, \quad \forall i \in T, j \in T, i \neq j \tag{4-8}$$

where d_{ij} is the minimum duration of driving from the finishing location of task i to the start location of task j. Note that this duration does not involve loading and unloading time.

If a task is the first one to be completed by a robot, the time to go to the start location of the task from the initial position of the robot must be taken into account:

$$t_e \geq s_{nep}(d_{pn} + t_p^{\text{ready}}) - \sum_{v=1}^{e-1}\sum_{n\in T} s_{nvp}(d_{pn} + t_p^{\text{ready}}) \quad \forall e \in \mathcal{E}, e \neq 0, \quad \forall n \in T, \quad \forall p \in \mathcal{A} \tag{4-9}$$

where d_{pn} is the duration of robot p going tot the starting point of task n and t_p^{ready} is the time at which robot p is available for starting a task, to account for robots that are occupied until after the reference time t_0.

At this point it is useful to know which events correspond to the start and end of a task. Therefore auxiliary variables u_{np}^{s} and u_{np}^{f} are defined, which are the start and finish event number for task n done by robot p:

$$u_{np}^{\text{s}} = \begin{pmatrix} s_{n1p} & \dots & s_{n(L-2)p} \end{pmatrix} \begin{pmatrix} 1 \\ \vdots \\ L-2 \end{pmatrix} \quad \forall n \in T \quad \forall p \in \mathcal{A} \tag{4-10}$$

$$u_{np}^{\text{f}} = \begin{pmatrix} f_{n1p} & \dots & f_{n(L-2)p} \end{pmatrix} \begin{pmatrix} 1 \\ \vdots \\ L-2 \end{pmatrix} \quad \forall n \in T \quad \forall p \in \mathcal{A} \tag{4-11}$$

Using these auxiliary variables, a constraint that ensures a robot only fulfils a single task at a time can be applied. This is done by forcing that one out of a pair of tasks fulfilled by the same robot starts only after the other has finished:

$$\begin{aligned} \forall (i,j) \in T, i < j, \quad \forall p \in \mathcal{A}: \qquad & u^{\mathrm{s}}_{ip} \geq u^{\mathrm{f}}_{jp} + 1 \\ \text{or} \quad & u^{\mathrm{s}}_{jp} \geq u^{\mathrm{f}}_{ip} + 1 \\ \text{or} \quad & u^{\mathrm{s}}_{ip} + u^{\mathrm{s}}_{jp} \leq 0 \end{aligned} \tag{4-12}$$

If the last *or*-condition is met, robot p starts nor finishes neither task n or j, so the others need not apply.

Transforming these inequalities into MILP standard form, using the big M method as described in Section 3-4-1:

$$\begin{aligned} \forall (i,j) \in T, i < j, \quad \forall p \in \mathcal{A}: \qquad -u^{\mathrm{s}}_{ip} + u^{\mathrm{f}}_{jp} - Cb_{ijp1} &\leq -1 \\ -u^{\mathrm{s}}_{jp} + u^{\mathrm{f}}_{ip} - Cb_{ijp2} &\leq -1 \\ u^{\mathrm{s}}_{ip} + u^{\mathrm{s}}_{jp} - Cb_{ijp3} &\leq 0 \\ b_{ijp1} + b_{ijp2} + b_{ijp3} &\leq 2 \end{aligned} \tag{4-13}$$

where $b_{ijp1} \in \{0,1\}$, $b_{ijp2} \in \{0,1\}$, $b_{ijp3} \in \{0,1\}$ and $M \geq u^{\mathrm{s}}_{ip} + u^{\mathrm{s}}_{jp} + u^{\mathrm{f}}_{ip} + u^{\mathrm{f}}_{jp} + 1$ is a large number.

To prevent pick-up before the earliest departure time and delivery before the earliest arrival time of a task the following constraints are applied:

$$\begin{aligned} \forall n \in T: \quad t^{\mathrm{s}}_n \geq {} & t^{\mathrm{ED}}_n \\ t^{\mathrm{f}}_n \geq {} & t^{\mathrm{EA}}_n + t_{\mathrm{unload}} \end{aligned} \tag{4-14}$$

where the constant t_{unload} is the time it takes to unload a delivery. This constraint ensures unloading cannot begin before the earliest arrival deadline. The variables t^{s}_n and t^{f}_n are the start and finish time of task n. They must be written in terms of the decision variables:

$$t^{\mathrm{s}}_n = \sum_{e \in \mathcal{E}} \sum_{p \in \mathcal{A}} s_{nep} t_e = \sum_{e \in \mathcal{E}} \sum_{p \in \mathcal{A}} z^{\mathrm{s}}_{nep} \tag{4-15}$$

The multiplication of s_{nep} with t_e is nonlinear, so the auxiliary variable z^{s}_{nep} is realised using the following constraints:

$$\begin{aligned} \forall n \in T, \quad \forall e \in \mathcal{E}, \quad \forall p \in \mathcal{A}: \quad z^{\mathrm{s}}_{nep} & \leq Cs_{nep} \\ z^{\mathrm{s}}_{nep} & \geq cs_{nep} \\ z^{\mathrm{s}}_{nep} & \leq t_e - c(1 - s_{nep}) \\ z^{\mathrm{s}}_{nep} & \geq t_e - C(1 - s_{nep}) \end{aligned} \tag{4-16}$$

where C is a large number and c a small number, once again applying the big M method (Section 3-4-1). The auxiliary variable z^{f}_{nep} is defined in the same way, except for that it applies to finish events.

4-4 Objective function

The goal of the scheduler is to make sure tasks are completed on time and as soon as possible. Several objective functions are proposed.

4-4-1 Makespan

The makespan is the duration of completing all tasks. To minimise the makespan, the objective function is the arrival time of the task that is completed the latest. Recall that events are in sequential order, therefore the last event is the completion time of the final task.

$$J_1 = t_{L-1} \tag{4-17}$$

The disadvantage of this approach is that multiple solutions can result in the same cost. If one task has time windows later than all the others, it will nearly always be the latest scheduled task and so the assignments of all other tasks bears no effect on the objective function.

4-4-2 Start and finish times

This objective function minimises aggregate start and finish times of tasks. A weighting factor can be used to give certain tasks higher priority.

$$J_2 = \sum_{n=1}^{N} \left(W_n^{\mathrm{s}} t_n^{\mathrm{s}} + W_n^{\mathrm{f}} t_n^{\mathrm{f}} \right) \tag{4-18}$$

where t_n^{f} is the finish time of task n and W_n is a weighting factor for task n.

4-4-3 Lateness

This cost function minimises the total time that tasks arrive or depart after their departure or arrival deadlines. Earliest departure and earliest arrival are enforced by constraints rather than with the objective function, so that a delivery cannot be picked up before it is ready (Section 4-3). Contrary to J_2, this function accumulates cost only for the time tasks miss their deadlines, being early has no effect.

$$J_3 = \sum_{n=1}^{N} \left(W_n^{\mathrm{s}} \max(t_n^{\mathrm{s}} - t_n^{\mathrm{LD}}, 0) + W_n^{\mathrm{f}} \max(t_n^{\mathrm{f}} - t_n^{\mathrm{LA}}, 0) \right) \tag{4-19}$$

where t_n^{LA} and t_n^{LD} are the latest arrival and departure deadlines for task n.

Auxiliary variables h_n^{s} and h_n^{f} are used to bring the nonlinear max-operation into MILP form:

$$\begin{array}{rrcl} \forall n \in T: & h_n^{\mathrm{s}} & \geq & t_n^{\mathrm{s}} - t_n^{\mathrm{LD}} \\ & h_n^{\mathrm{s}} & \geq & 0 \\ & h_n^{\mathrm{f}} & \geq & t_n^{\mathrm{f}} - t_n^{\mathrm{LA}} \\ & h_n^{\mathrm{f}} & \geq & 0 \end{array} \tag{4-20}$$

Note that this method of representing the max-function assumes that the optimiser will seek to minimise h_n^{s} and h_n^{f}, otherwise their values are not guaranteed to produce the same outcome as the max-function. It is possible to guarantee this, but that would require defining additional variables which is not necessary for the purpose of minimising lateness.

The objective function finally becomes:

$$J_3 = \sum_{n=1}^{N} (W_n^{\mathrm{s}} h_n^{\mathrm{s}} + W_n^{\mathrm{f}} h_n^{\mathrm{f}}) \tag{4-21}$$

4-4-4 Time windows missed

The next objective function aims to minimise the percentage of departure and arrival time windows that were missed by the schedule. Introducing binary variables $\delta_{\mathrm{s}n}^{\mathrm{late}} \in \{0,1\}$ to denote whether task n was started after its latest departure deadline and $\delta_{\mathrm{f}n}^{\mathrm{late}} \in \{0,1\}$ to denote whether it was finished after its latest arrival deadline, the function becomes:

$$J_4 = \frac{100}{2N} \sum_{n=1}^{N} \left(\delta_{\mathrm{s}n}^{\mathrm{late}} + \delta_{\mathrm{f}n}^{\mathrm{late}}\right) \tag{4-22}$$

The relations $\delta_{\mathrm{s}n}^{\mathrm{late}} = 1 \leftrightarrow h_n^{\mathrm{s}} > 0$ and $\delta_{\mathrm{f}n}^{\mathrm{late}} = 1 \leftrightarrow h_n^{\mathrm{f}} > 0$ are obtained using the following constraints:

$$\begin{aligned} \forall n \in T: \qquad -h_n^{\mathrm{s}} + \epsilon &\leq C(1 - \delta_{\mathrm{s}n}^{\mathrm{late}}) \\ h_n^{\mathrm{s}} - \epsilon &\leq -c\delta_{\mathrm{s}n}^{\mathrm{late}} \end{aligned} \tag{4-23}$$

where $\epsilon > 0$ is a small number to denote the tolerance of the inequality constraint, C is a large number and c a small number. Similar to J_3, this function is not guaranteed to compute the correct percentage of time windows missed if the solver does not optimise for h_n^{s} and h_n^{f}, or $\delta_{\mathrm{s}n}^{\mathrm{late}}$ and $\delta_{\mathrm{f}n}^{\mathrm{late}}$.

4-4-5 Variables and constraints

The total number of variables and constraints depends on the number of robots P, the number of tasks N and the number of events $L = 2N + 2$. The number of variables is:

$$2NLP + L + 2NP + 3N^2P + 2NLP + 4N \tag{4-24}$$

The number of inequality constraints is:

$$\begin{aligned} (L-1) + NL + N\left(\frac{L(L+1)}{2} - 1\right) + N(N-1)\left(\frac{L(L+1)}{2} - 1\right) + NLP + \cdots \\ \cdots 4P\left(\frac{N(N+1)}{2} - 1\right) + 2N + 6NLP + 6N \end{aligned} \tag{4-25}$$

The number of equality constraints is:

$$1 + 2N + 5NP \tag{4-26}$$

4-5 Conclusion

This chapter introduces a scheduler for pick-up and delivery tasks to be done by multiple robots. Like the path planner from Chapter 3, a MILP formulation is used. MILP is suitable to the scheduling problem because integer variables allow to model the decision tree. The decision tree arises from the fact that a finite number of solutions exists, due to the finite number of robots and tasks.

The behaviour of the scheduler is defined by the constraints that ensure robots fulfil only a single task at a time, that tasks are done by only one robot, that the time for moving between tasks is taken into account etc. Four objective functions are proposed. The makespan, i.e. the completion time of the last task, is a common optimisation goal, but it is not very suitable when the schedule must accord to time windows. If the time windows prescribe a certain task to be completed later than all others, the solver will only optimise for this task and not for the rest. The second objective function aims to minimise start and finish times of tasks, i.e. start and finish tasks as early as possible. The third cost function penalises the total amount of time that tasks are started after their pick-up deadline or finished after their delivery deadline. This objective is used when lateness must be avoided but the amount of time tasks are completed before their deadline is not important. The fourth cost function minimises the percentage of tasks missed. This objective function does not penalise the time robots are late, only the fact that a deadline was missed. This function can be used when there is no point completing a task at all if it is not on time.

The scheduler takes pick-up and delivery time windows for tasks, and assigns them to the available robots in such a way that an cost function is minimised. Together with the path planner from Chapter 3, the scheduler is used in Chapter 5 to schedule tasks as well as plan paths for the robots.

Chapter 5

Integration and results

In this chapter three methods for integration of path planning and task scheduling are proposed: rule-based scheduling with single-robot path planning (RB-SR), MILP scheduling with single-robot path planning (MILP-SR) and MILP scheduling with multi-robot path planning (MILP-MR). These methods are compared using simulations and the results are reported and discussed.

5-1 Performance criteria

The Mixed-Integer Linear Programming (MILP) path planner and MILP scheduler are distinct optimisations aiming to minimise their respective objective functions.

5-1-1 Path planner objective function

In Chapter 3 two objective functions were discussed. The objective function (3-14) minimises control actions and distance between robots and their goal states at every discrete time step and was introduced in [40]. A new objective function (3-25) that minimises the sum of durations for robots reaching their goals was proposed in Section 3-6. The objective function used by the path planner in this chapter is the sum of these two functions:

$$J_{\mathrm{pp}} = \sum_{p\in\mathcal{A}} \left(W_p k_{p,\mathrm{arrival}} - C\delta_{pK}^{\mathrm{goal}} + CP + \sum_{k=1}^{K-1} \boldsymbol{q}_{pk}^{\mathrm{T}} \boldsymbol{w}_{pk} + \sum_{k=0}^{K-1} \boldsymbol{r}_{pk}^{\mathrm{T}} \boldsymbol{v}_{pk} \right) \tag{5-1}$$

where $\mathcal{A}$ is the set of robots, $k_{p,\mathrm{arrival}}$ is the arrival time instance of robot p, $\delta_{pK}^{\mathrm{goal}}$ is a binary variable denoting whether robot p has arrived at its goal at the final time instance K, $\boldsymbol{w}_{pk}$ is the absolute value of the state error of robot p at time instance k and $\boldsymbol{v}_{pk}$ is the absolute value of the control action of robot p at time instance k. The term CP is added so that the objective value gives an intuitive value for the approximate sum of durations. Parameters are chosen as follows: $W_p = 1$, $C = 1000$, $\boldsymbol{q}_{p1} = 0.001 \cdot (1\ 1\ 0\ 0)^{\mathrm{T}}$ increasing linearly to

$\boldsymbol{q}_{pK} = 0.001 \cdot (100\ 100\ 00)^{\mathrm{T}}$ and $\boldsymbol{r}_{pk} = 0.0001 \cdot (1\ 1)^{\mathrm{T}}$. The variable ϵ from (3-23) that determines the tolerance of $\delta_{pk}^{\mathrm{goal}}$ is set to $0.01 \cdot (1\ 1\ 1\ 1)^{\mathrm{T}}$. This means that when the robot approaches its goal within 1 cm in the x and y direction and has a speed lower than 0.01 m/s in both directions it is assumed to have reached the goal.

The main objective is to minimise duration of paths. However, the duration is a discrete variable and can only take values that are a multiple of the sampling time. To further optimise paths and to prevent excessive or oscillating control actions the distance objective function is used with small weights $\boldsymbol{q}_{pk}$ and $\boldsymbol{r}_{pk}$ on the difference between the robot state and the goal state and on the control actions respectively. A comparison of objective functions for the path planner is made in Section 5-6-1.

5-1-2 Scheduler objective functions

Two objective functions are used for the MILP scheduler. The first is used to complete a set of tasks without deadlines and the second to complete a set of tasks with deadlines. The main objective for task sets without deadlines is to minimise the makespan from (4-17), which is the completion time of the last task. However, minimising the makespan only optimises the assignment of tasks to robots and the start and finish times of tasks that influence the makespan. To start and finish all tasks as early as possible given a particular assignment that optimises the makespan, (4-17) is combined with (4-18) with weights on the latter set really small, because the makespan is the primary optimisation objective. The objective function becomes:

$$J_{\mathrm{makespan}} = t_{L-1} + \sum_{n=1}^{N} \left(W_n^{\mathrm{s}} t_n^{\mathrm{s}} + W_n^{\mathrm{f}} t_n^{\mathrm{f}} \right) \tag{5-2}$$

where t_{L-1} is the time of the last event, i.e. the finish time of the last task, N is the number of tasks, t_n^{s} is the start time of task n and t_n^{f} the finish time of task n. The weights are set as $W_n^{\mathrm{s}} = W_n^{\mathrm{f}} = 0.01$.

The second objective function is used for task sets that do have deadlines, and aims to minimise the percentage of latest departure and latest arrival deadlines missed as is done with (4-22). Using this function produces rather discrete outcomes and penalises being 1 second late equally to being 10 seconds late. The reason for this is that being late will require some kind of human intervention anyway to not disturb the production line. However, without any weights on start and finish times the scheduler might schedule a task very far into the future if it is late anyway. To remedy this, the same terms from (4-18) with small weights $W_n^{\mathrm{s}} = W_n^{\mathrm{f}} = 0.01$ are added as was done with J_{makespan}:

$$J_{\mathrm{deadlines}} = \frac{100}{2N} \sum_{n=1}^{N} \left(\delta_{\mathrm{s}n}^{\mathrm{late}} + \delta_{\mathrm{f}n}^{\mathrm{late}} \right) + \sum_{n=1}^{N} \left(W_n^{\mathrm{s}} t_n^{\mathrm{s}} + W_n^{\mathrm{f}} t_n^{\mathrm{f}} \right) \tag{5-3}$$

where $\delta_{\mathrm{s}n}^{\mathrm{late}}$ is a binary variable denoting whether task n started after its departure deadline and $\delta_{\mathrm{f}n}^{\mathrm{late}}$ is a binary variable denoting whether task n finished after its arrival deadline.

Start and finish times are minimised in addition to the makespan or percentage of deadlines missed so that delays caused by collision avoidance have less effect on the main objective than they would if start times were maximised for example. Maximising start times could be done in an effort to make robots spend less time doing tasks and more time unoccupied. This can be advantageous in case new tasks are added that need an unoccupied robot to complete them.

5-1-3 Performance indicators

Performance of the integration of path planning and task scheduling is measured by the indicators described in Table 5-1.

Performance indicator	**Description**
Computation time (s)	Total time spent by the MILP solver on path planning for a given schedule. Scheduling time is not included in this time, because a maximum computation time of 120 s was used as a stopping criterion. The scheduling time is constant for simulations that use the MILP scheduler, since it does not converge within 10 hours, let alone 120 s.
Total distance (m)	Sum of distances driven by all robots.
Energy (10^3 J)	Total amount of energy spent by all robots, calculated using control actions of the force control model. Note that the model does not account for friction so this is also not included in the energy calculation.
Deadlines missed (%)	Percentage of latest departure and latest arrival deadlines that have been missed over the entire task set.
Total delays (s)	Cumulative time that all robots depart or arrive after their respective deadlines.
Later than earliest (s)	Cumulative time robots depart or arrive after the *earliest* departure or arrival time of a task. In other words, they were not necessarily late, but it is an indicator of how much earlier they could have been.
Makespan (s)	Completion time of the last task.

Table 5-1: Performance indicators for the integration of path planning and task scheduling

5-2 Settings and parameters

All simulations are run on a HP ZBook Studio G4 with an Intel Core i7-7700HQ processor and 8GB of RAM. Constraint matrices are generated using MATLAB R2017b on a 64-bit Windows 10 operating system, from which Gurobi Optimizer 7.5.1 is called for running optimisations. Results are processed and visualised using MATLAB.

The robot is modelled as a solid disk and the force control vehicle model is used (Section 3-3-2). Default parameters for simulations in this chapter are shown in Table 5-2.

The maximum control action of 200 N amounts to a maximum acceleration of 1 m/s^2. Loading and unloading times are minimum durations for a robot to remain stationary at the departure and arrival station of a task, to allow for the load to be transferred to or from the robot.

Parameter	**Value**
Robot mass (kg)	200
Robot radius (m)	0.5
Maximum robot speed	2
Maximum control action (N)	200
Loading time (s)	5
Unloading time (s)	5
Maximum relative gap G_{rel} (%)	0.01

Table 5-2: Default simulation parameters

For MILP problems with many variables it is often computationally not feasible to evaluate all integer solutions. Stopping criteria are used to terminate the optimisation and return the best solution found until then. A common stopping criterion is computation time. Another criterion is the relative MILP optimality gap G_{rel}. This is the difference between the lower bound J_{lower} on the objective value and the best feasible objective value that has been found so far, called the *incumbent* objective value $J_{\text{incumbent}}$, relative to $J_{\text{incumbent}}$.

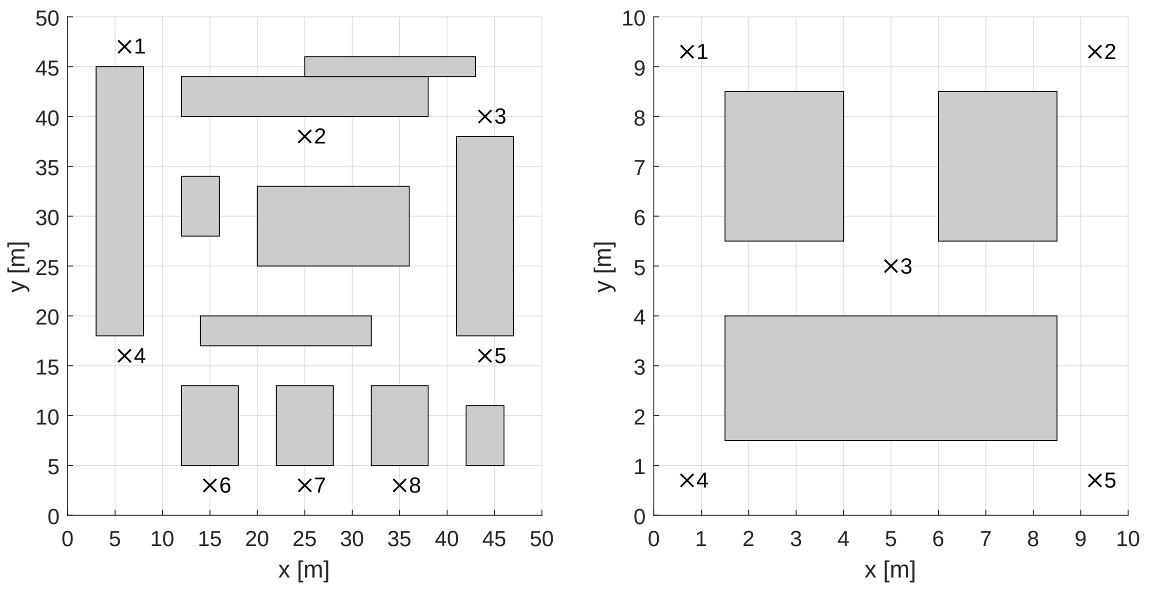
$$G_{\text{rel}} = \frac{|J_{\text{lower}} - J_{\text{incumbent}}|}{|J_{\text{incumbent}}|} \tag{5-4}$$

The lower bound is determined by *relaxing* (i.e. disregarding) some integer constraints and calculating the objective value for this likely infeasible solution. The relative gap can become lower over time either if a feasible solution with a lower objective value is found or if the lower bound is increased by relaxing fewer integer constraints. By using a maximum relative gap tolerance as a stopping criterion a certain solution quality is guaranteed, but depending on the problem it can take a long time to reach that criterion. That is why it is often used in combination with a maximum computation time.

The MILP path planner uses a finite time horizon with a number of time steps K. The default time horizon for single-robot path planning is set to twice the expected duration of a task according to a pre-calculated table of durations between all stations. In case no path to the goal is found within the default time horizon, the path planner reruns with a time horizon increased in steps of 10 seconds until a path to the goal is found. The time horizon is increased up to a duration that would allow to traverse the physical space from one end to the other and back. If still no path is found, the problem is assumed infeasible and human intervention is required, but this has not occurred during experiments in this chapter.

5-3 Maps

Two physical environments are used, of which maps are shown in Figure 5-1. The first is a **factory floor map** and is shown in Figure 5-1a. The map is based on a 2D scan of a factory floor at Prodrive Technologies (Figure B-1 in Appendix B). Obstacles from the original scan have been simplified into 11 rectangular obstacles to reduce the number of obstacle constraints necessary. The factory floor is 50-by-50 m area where the edges of the map are assumed to be walls. Because robots are 1 m in diameter, they can pass each other almost everywhere. This is not the case for the second map, which is denoted as the **corridor map** and is shown in Figure 5-1b. Robots are not able to pass each other so they must drive round obstacles to avoid one another. Simulations with the factory floor map use a sampling time $T_s = 0.5$ s and those with the corridor map use $T_s = 0.25$ s because of the smaller distances involved. Table A-1 and Table A-2 are pre-calculated duration tables for the two maps containing the durations for a single robot to move between stations.



(a) Factory floor map **(b)** Corridor map

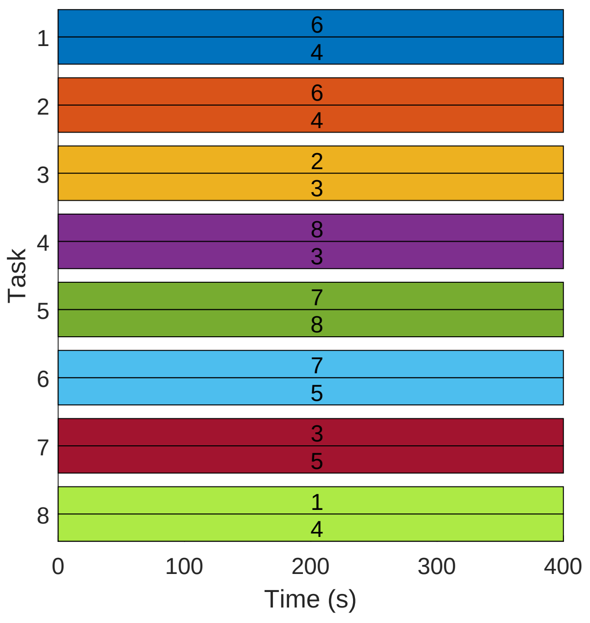
Figure 5-1: Maps of physical environments, where obstacles are shown as grey rectangles and pick-up and delivery stations are denoted by crosses with their respective station numbers

5-4 Task sets

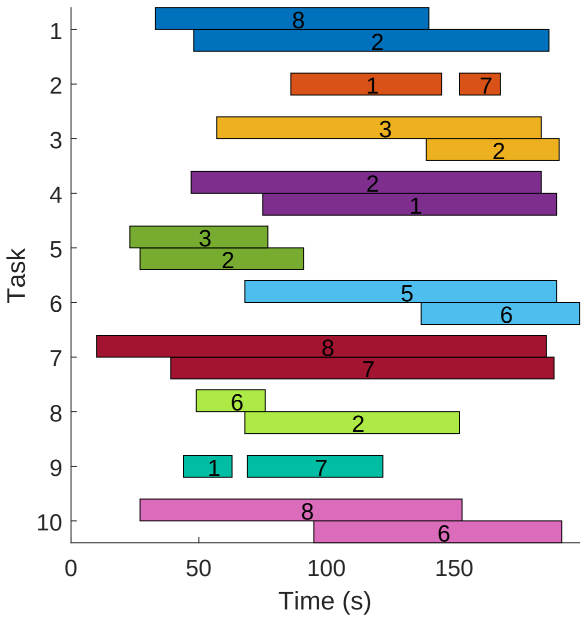
Four cases are presented for comparison, each containing a set of tasks with a pick-up time window and a delivery time window. The cases are shown in Figure 5-2 and Figure 5-3. All station numbers, robot starting positions and deadlines are randomly generated. The goal of the scheduler is to schedule tasks so that their start and finish times fall within the respective time windows.

5-5 Integration methods

In previous chapters MILP formulations for path planning and task scheduling have been introduced. Next, three methods for integrating path planning and scheduling are introduced. The goal of these integration methods is to generate a schedule and the corresponding paths of the robots, in order to complete a set of tasks.

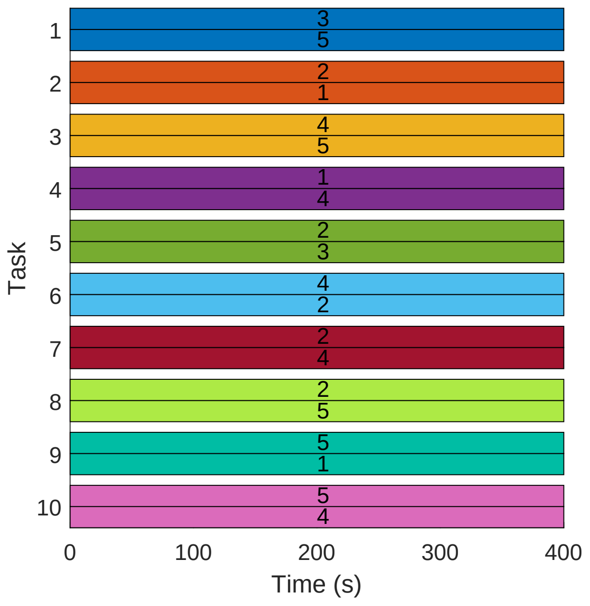


(a) Factory floor map, no deadlines

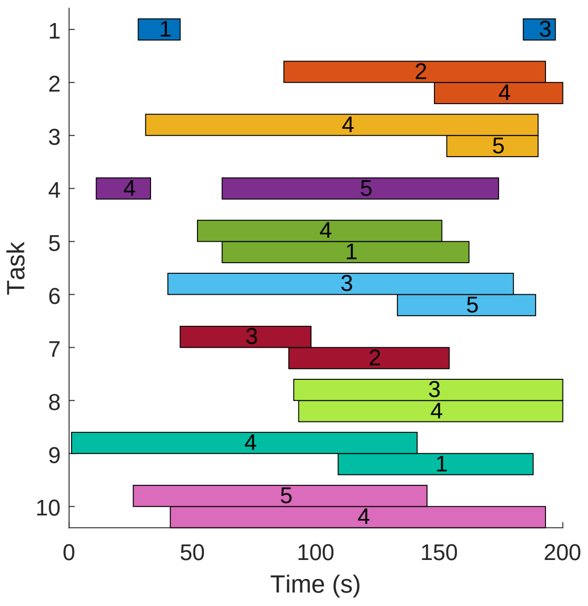


(b) Factory floor map, with deadlines

Figure 5-2: Task numbers are shown on the vertical axis, each task number corresponds to a colour. Station numbers are shown on the time windows. Departure time windows are displayed above or to the left of arrival time windows.



(a) Corridor map, no deadlines



(b) Corridor map, with deadlines

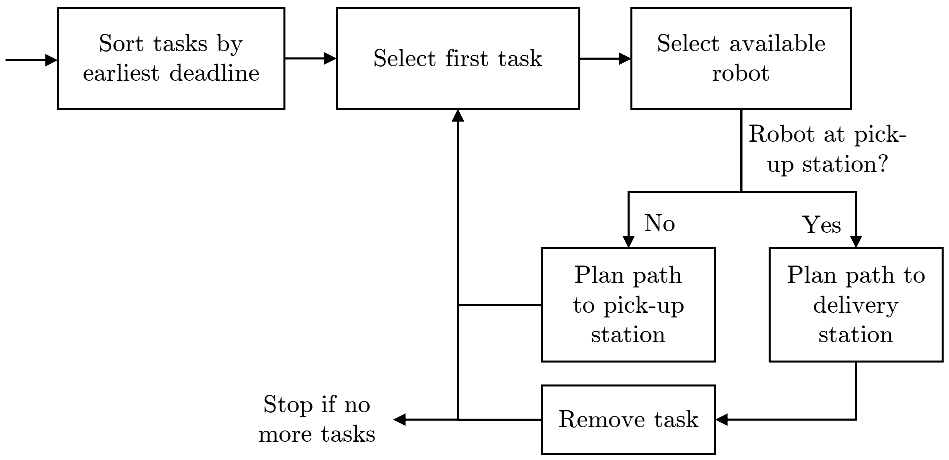
Figure 5-3: Task numbers are shown on the vertical axis, each task number corresponds to a colour. Station numbers are shown on the time windows. Departure time windows are displayed above or to the left of arrival time windows.

5-5-1 Method 1: Rule-based scheduling, single-robot path planning (RB-SR)

The first method uses rule-based scheduling. This means that tasks are ordered using a heuristic, and assigned to robots in that order. The Earliest Deadline First (EDF) scheduling rule is used. All tasks are sorted in ascending order by their latest arrival time [3]. This is a useful heuristic when aiming to minimise late arrivals, since priority is given to tasks with the nearest deadline.

Path planning is done using the MILP path planner from Chapter 3. However, paths are planned for only one robot at a time. After a path is calculated, it is saved and the robot is treated as a dynamic obstacle by other robots that have their paths planned subsequently.

Figure 5-4 shows the process the rule-based single-robot (RB-SR) method for scheduling and path planning. After the tasks have been ordered by their latest arrival time, the first task on the list with the earliest deadline is selected. Next, a calculation is made for each robot when it would be able to start the task, using state information about the robots position and availability and a pre-calculated table of estimated travel durations between stations. The robot that is expected to arrive at the pick-up location of the task the earliest is assigned to the task.



Sort tasks by earliest deadline
Select first task
Select available robot
Robot at pick-up station?
No
Yes
Plan path to pick-up station
Plan path to delivery station
Stop if no more tasks
Remove task

Figure 5-4: Process diagram for the rule-based scheduling and single-robot path planning method (RB-SR)

If the selected robot is already at the pick-up station, the MILP path planner plans the path from the pick-up station to the delivery station, regarding any previously planned paths of other robots as moving obstacles. The resulting path is saved, including the robot being stationary during loading and unloading of its delivery. Finally the task is removed from the list and the next task on the list is selected.

If the selected robot is not yet at the pick-up station for its assigned task, a path is planned to the starting position of the task. Afterwards, the first task on the list is selected, which will still be the same task. But now the robot is at the pick-up station of the task, so the process will go as described above. Once all tasks have be completed the list of tasks will be empty and the process stops.

To facilitate the use of a durations table, robots are assumed to start at a station and remain there after making their delivery. After a robot has unloaded, it is removed from the map to allow other robots to use the station. The robot is assumed to go to a buffer, the exact manoeuvre of which is not computed because this is a complex planning problem by itself and out of scope for this project. Once a robot is needed again, it is able to reappear at the station it arrived last, provided no other robots are loading or unloading at the station.

5-5-2 Method 2: MILP scheduling, single-robot path planning (MILP-SR)

The second method for scheduling and path planning uses the MILP scheduler from Chapter 4 and the same single-robot path planning method as the first method. It is therefore called MILP-SR for short.

The process for the MILP-SR method is shown in Figure 5-5. First, the MILP scheduler generates a schedule using the set of tasks while optimising for its objective function. In doing so it uses the same table of durations for its duration estimates as is used by the RB-SR method. The result is a schedule with a set of tasks per robot in a particular order, with estimated start and finish times of tasks.

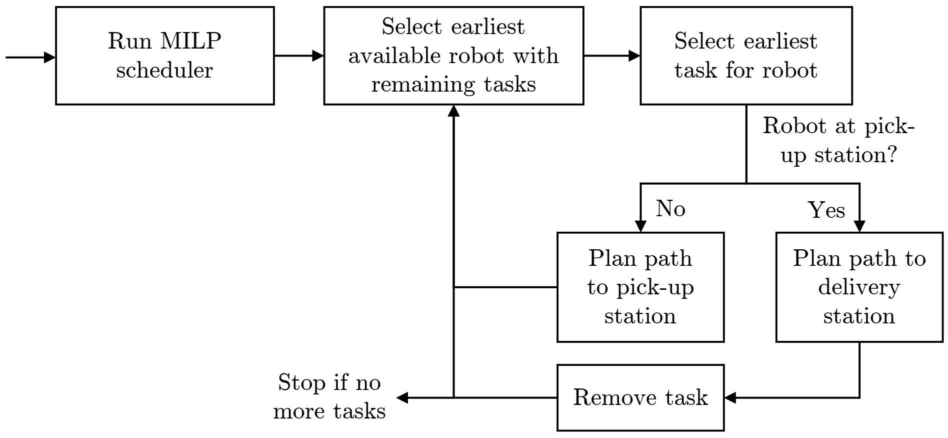

Run MILP scheduler
Select earliest available robot with remaining tasks
Select earliest task for robot
Robot at pick-up station?
No
Yes
Plan path to pick-up station
Plan path to delivery station
Stop if no more tasks
Remove task

Figure 5-5: Process diagram for the MILP scheduling and single-robot path planning method (MILP-SR)

The robot is selected that has remaining tasks to complete and that has the earliest availability after completing its previous tasks. For this robot, its next task as was determined by the MILP scheduler is selected. Depending on whether the robot is already present at the pick-up location, a path is planned to move to the start of the task or to complete the task using the MILP path planner. If the task is completed it is removed from the set of tasks and the process repeats by selecting the earliest available robot again.

5-5-3 Method 3: MILP scheduling, multi-robot path planning (MILP-MR)

The third method, called MILP-MR, uses the MILP scheduler and the MILP path planner, planning paths for multiple robots at the same time. The process for this method is shown in Figure 5-6. As with the second method, the MILP scheduler first generates a schedule. However, instead of selecting one task corresponding to the earliest available robot, the first task for each robot is selected.

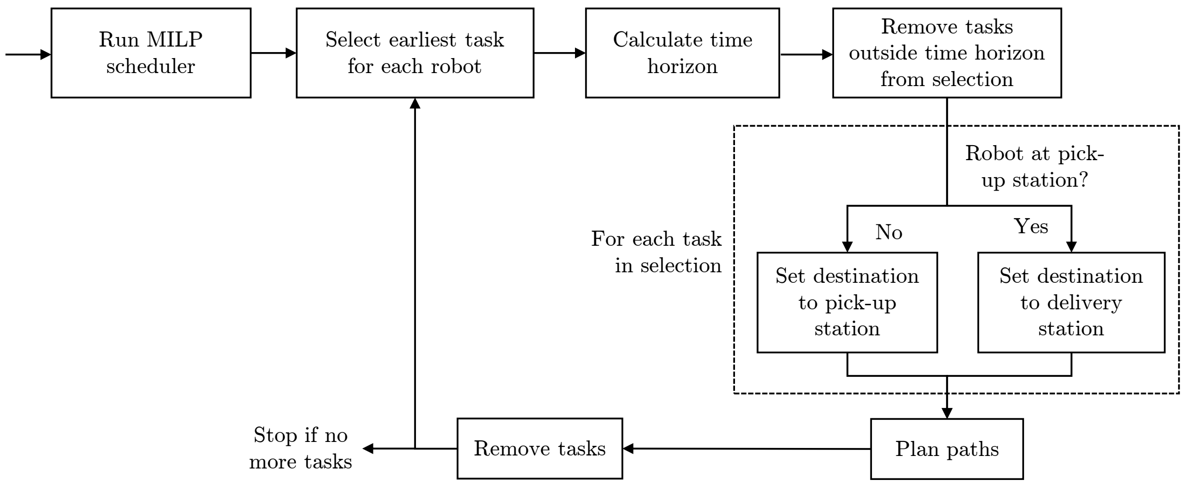

Run MILP scheduler
Select earliest task for each robot
Calculate time horizon
Remove tasks outside time horizon from selection
Robot at pick-up station?
For each task in selection
No
Yes
Set destination to pick-up station
Set destination to delivery station
Stop if no more tasks
Remove tasks
Plan paths

Figure 5-6: Process diagram for the MILP scheduling and multi-robot path planning method (MILP-MR)

Next, the time horizon for the path planner is determined. If tasks for different robots do not overlap in time, planning them with a single optimisation would require a large time horizon without robots interacting, resulting in an unnecessarily large number of constraints and variables, and long computation time. Therefore, starting with the robot that is available the earliest, its time horizon is determined by doubling its estimated completion time according to a pre-calculated duration table between stations, to account for detours. The same is done for the robot with the next earliest availability, if its departure time falls within the time horizon of the previous robot. If the departure time of a robot does not fall within the time horizon of previous robots, its task is removed from the selection, to be planned for at a later iteration.

Next, for each robot it is determined to which goal location it should move, depending on whether it must go to the pick-up location of a task or do the task itself. Finally, paths are planned for multiple robots simultaneously, and the completed tasks are removed from the taskset. Once again the earliest task for each robot is selected and the process repeats itself until all tasks are completed.

5-6 Results

Experiments are done on path planner objective functions, scheduling methods and integration methods.

5-6-1 Path planner comparison

Four objective functions for the path planner are compared. The first cost function J_1 uses (3-14) that calculates cumulative distances to the goal state and control actions at every time instance, with the weights weights $\boldsymbol{q} = (1\ 1\ 0\ 0)^{\mathrm{T}}$ and $\boldsymbol{r} = (0.01\ 0.01)^{\mathrm{T}}$. The terminal weight is $\boldsymbol{p} = (1000\ 1000\ 0\ 0)^{\mathrm{T}}$. All weights are the same for each robot and for every time instance.

The second objective function J_2 also uses (3-14) but with weights $\boldsymbol{q}_{p1} = (1\ 1\ 0\ 0)^T$ to $\boldsymbol{q}_{pK} = (100\ 100\ 00)^T$ that increase linearly over the time horizon. The weights on the control action and the terminal cost are the same as those of J_1.

The third objective function J_3 minimises the cumulative durations for all robots to reach their goal with (3-25). Robots are weighted equally with $W_p = 1$ and the constant C is set to 1000 to reward robots for reaching their goal.

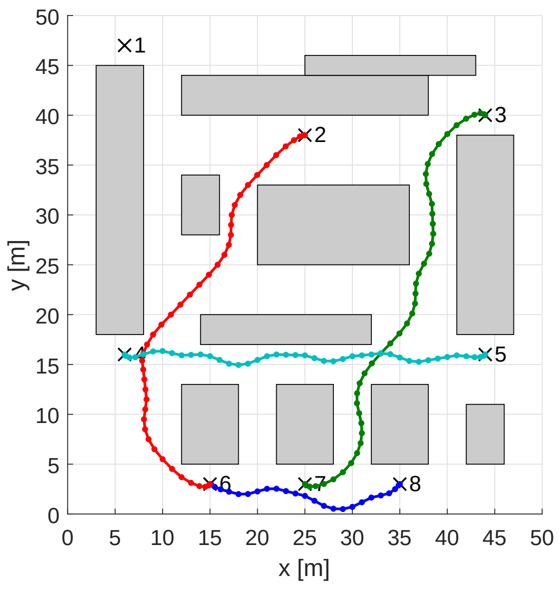
The fourth objective function is (5-1) with parameters $W_p = 1$, $C = 1000$, $\boldsymbol{q}_{p1} = 0.001 \cdot (1\ 1\ 0\ 0)^{\mathrm{T}}$ increasing linearly to $\boldsymbol{q}_{pK} = 0.001 \cdot (100\ 100\ 00)^{\mathrm{T}}$ and $\boldsymbol{r}_{pk} = 0.0001 \cdot (1\ 1)^{\mathrm{T}}$.

In addition to the default relative gap tolerance a maximum computation time of 120 seconds is used as a stopping criterion. The factory floor map is used. Three cases are defined, with 2, 3 and 4 robots respectively. The cases involve varying path lengths and interactions between robots. All robots start at $t = 0$ and the time horizon is fixed to 80 time steps, or 40 seconds, for all simulations. All cases are optimised with all four objective functions. Table 5-3 contains the objective values for each objective function for all of the experiments. Performance indicators are shown in Figure 5-8. Full results are found in Table A-3.

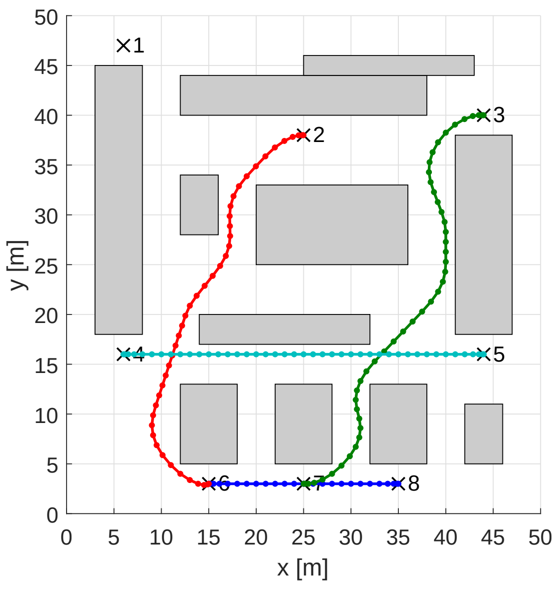
Case	**Objective**	J_1	J_2	J_3	J_4
1	J_1	3556	34035	119	154.7
	J_2	3545	31491	112	145.5
	J_3	10910	150347	107	259.6
	J_4	3488	31205	107	139.6
2	J_1	4576	48650	168	218.2
	J_2	4610	44800	155	202.2
	J_3	15387	231315	151	385.6
	J_4	4444	43207	151	196.3
3	J_1	3540	35028	159	195.3
	J_2	3566	34535	159	195.3
	J_3	15729	227716	155	386.1
	J_4	3563	34915	155	191.4

Table 5-3: Objective values for all path planner cost functions while optimising paths for each cost function for three cases

The objective function J_3 stands out in Table 5-3 because of the large values for the other objective values when optimising for J_3. The first reason for this is that J_3 is an integer. This means that variables that do not have a large enough effect on the duration to change J_3 can take very large values, resulting in high cost for objective functions that do penalise these variables. The second reason is that due to a lack of penalty the control actions are very high and oscillate between time steps. This oscillation can be seen in Figure 5-7a. Figure 5-7b shows the paths for J_4 which does have penalties on the control actions, eliminating oscillations.

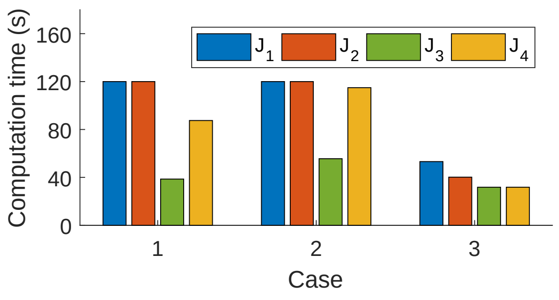


(a) J_3 as objective function

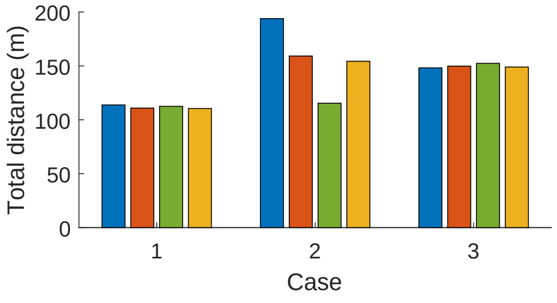


(b) J_4 as objective function

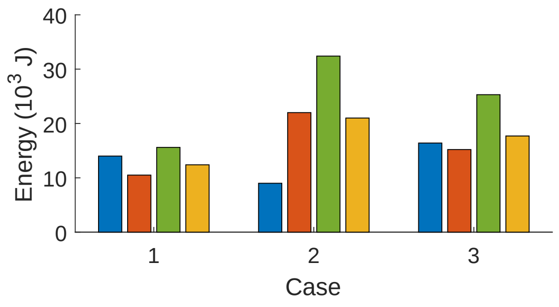
Figure 5-7: Each coloured line represents a path of a different robot, with dots on the lines representing sampling instances



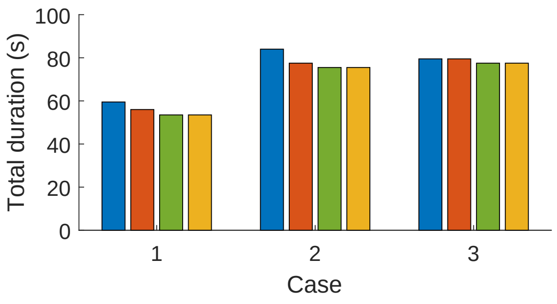
(a)



(b)



(c)



(d)

Figure 5-8: Performance indicators four objective functions for the path planner

Furthermore, notice how optimising for J_4 obtains the same value for J_3 as optimising for J_3 itself. Figure 5-8 shows that J_4 performs better than J_3 on all performance indices, except for the computation time. This shows that the extra time used by J_4 is time well spent on improving paths without coming at the cost of longer path durations. Both J_3 and J_4

perform better than J_1 and J_2 on computation time. The distance cost functions J_1 and J_2 only perform better on energy than the J_3 and J_4, which is explained by the fact that J_1 and J_3 have relatively larger penalties on the control action. Note that J_1 and J_2 do not minimise the total distance, instead they accumulate cost for the distance to the goal at every time step. Figure 5-8b shows that the duration cost functions produce paths with similar or even lower total distance than the distance cost functions. All in all, J_4 is a good choice when the aim is to minimise path durations while taking computation time, total distance and energy into consideration.

5-6-2 Scheduler comparison

Two experiments are done to compare the MILP scheduler that is used by the methods MILP-SR and MILP-MR with rule-based scheduling that is used by the method RB-SR. Both use the factory floor map. The first uses the task set without deadlines from Figure 5-2a and the second uses the task set with deadlines from Figure 5-2b. A maximum computation time of 120 s is used as stopping criterion for the MILP scheduler. This maximum time chosen as an acceptable time for on-demand scheduling in production environment. Results for the experiments are reported in Table A-4 and Table A-5.

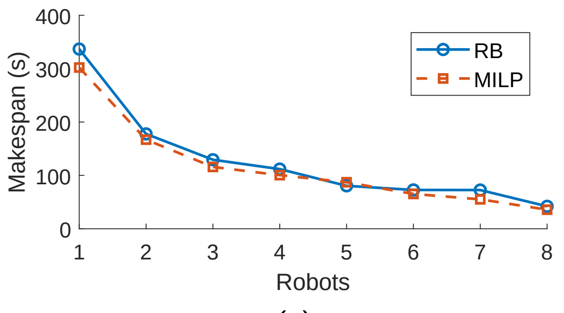
The first experiment uses the scheduler objective function J_{makespan} from (5-2) and the second experiment uses $J_{\text{deadlines}}$ from (5-3). Table 5-4 lists the objective values obtained using rule-based scheduling (RB) and using the MILP scheduler.

	J_{makespan}		$J_{\text{deadlines}}$	
Robots	**RB**	**MILP**	**RB**	**MILP**
1	336.75	302.1	127.91	110.16
2	177.72	166.98	58.55	73.13
3	129.33	115.85	26.61	31.57
4	111.78	100.44	16.94	41.4
5	80.46	87.15	13.98	24.98
6	72.71	65.23	12.21	18.1
7	72.52	55.12		
8	42.04	35.7		

Table 5-4: Objective values for two experiments, one for a task set without deadlines using the J_{makespan} and one for a task set with deadlines using $J_{\text{deadlines}}$

For the first experiment the MILP scheduler results in a better objective value and makespan for all runs except with 5 robots, as can be seen in Table 5-4 and Figure 5-9. This is unexpected, because the MILP scheduler optimises for the cost function, while the rule-based scheduler does not. The reason for this result is a lack of computation time. A test with unlimited computation time shows the MILP scheduler outperforms the rule-based scheduler after 706 s. Another test ran for 10 hours after which it was manually stopped, at a relative gap of 67%. Reaching a relative gap of 0.01% is therefore practically unattainable.

For the second experiment the results show worse performance of the MILP scheduler. In all cases except with a single robot the rule-based scheduler outperforms the MILP scheduler as becomes clear from Table 5-4 and Figure 5-10. For the test with 6 robots there are a total



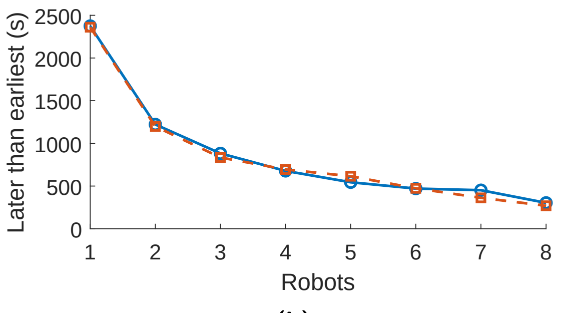
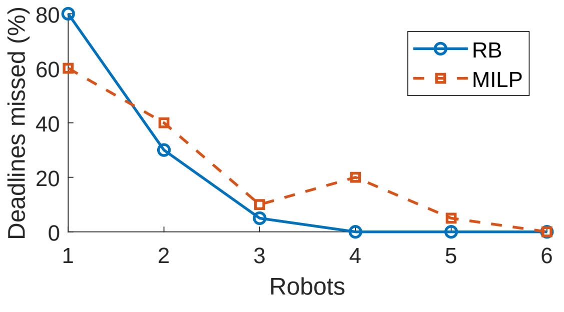


Figure 5-9: Comparison of rule-based scheduling (RB) and MILP scheduling for a task set without deadlines optimising for J_{makespan}

number of 34 002 constraints and 7 262 variables, of which 4 580 are integers, which means there are an enormous number of possible solutions. This leads to the conclusion that the MILP scheduler is no more suitable than a rule-based scheduler for a case with deadlines when a limit of 2 minutes of computation time is imposed.



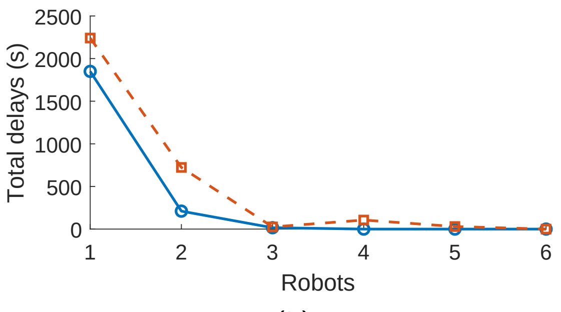


Figure 5-10: Comparison of rule-based scheduling (RB) and MILP scheduling for a task set with deadlines optimising for $J_{\text{deadlines}}$

5-6-3 Factory floor simulations

Two experiments are done using the factory floor map, which involves relatively large distances and a lot of free space for the robots to move in.

Without deadlines

The first experiment uses the randomly generated task set without deadlines from Figure 5-2a. Therefore the makespan objective function (5-2) is used. Simulations are done for 1 to 7 robots comparing the three integration methods RB-SR, MILP-SR and MILP-MR. Results are shown in Figure 5-11 and are reported in Table A-6.

Increasing the number of robots available to complete a set of tasks reduces the makespan (Figure 5-11a) and the sum of departure and arrival times (Figure 5-11c) since tasks can

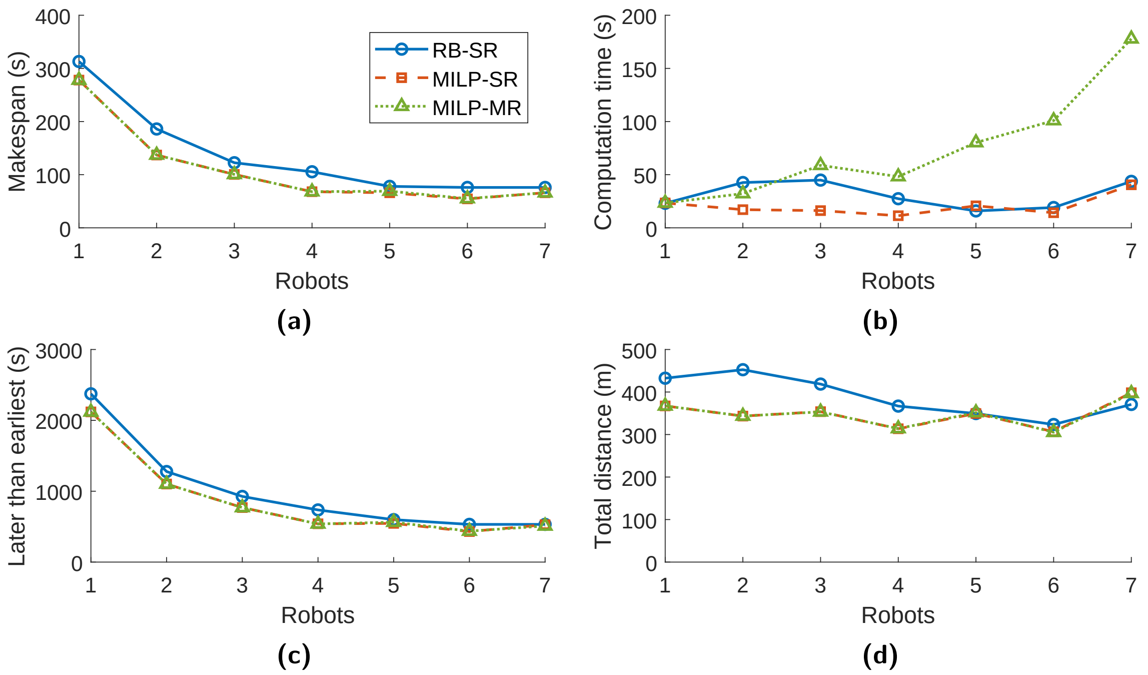


Figure 5-11: Comparison of three path planning and scheduling integration methods using the factory floor map and the task set without deadlines

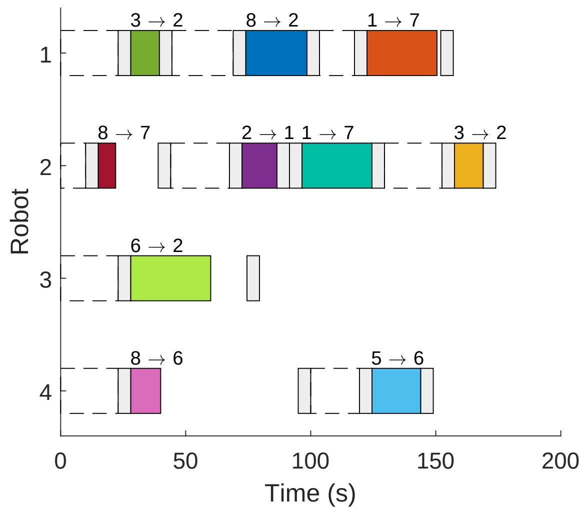
be completed in parallel instead of serially. However, for this task set the benefit of adding robots wears for more than 5 robots.

Conversely, a clear effect on the total distance driven does not become apparent for an increasing number of robots (Figure 5-11d). This is because tasks still have the same distance. Even if on the one hand less distance is travelled in between tasks with more robots, on the other hand larger distances may be travelled if that means tasks can be done in parallel to reduce the makespan.

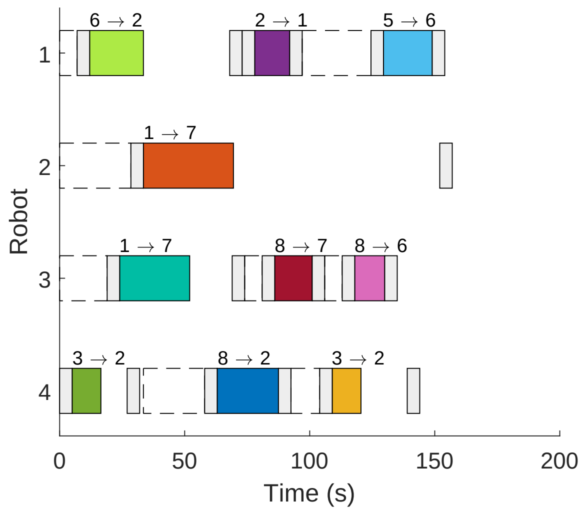
The effect of the number of robots on computation time for methods RB-SR and MILP-SR is also ambiguous (Figure 5-11b). Each requires roughly the same number of runs of the MILP path planner (Table A-6), so computation time is largely dependent on collision avoidance of previously planned paths and planning time horizons which produces some correlation with the total distance (Figure 5-11d). For MILP-MR, the effect of increasing numbers of robots is clear. Using more robots results in fewer runs of the path planner but for more robots at once. Planning paths for multiple robots at once dramatically increases the number of variables and constraints of the MILP problem, resulting in longer computation time.

The methods MILP-SR and MILP-MR outperform RB-SR in all cases, with an 11% to 36% lower makespan. This is because the MILP scheduler can assign tasks to robots so that robots can start a task at their own starting station, start a new task at the delivery station of a previous task or drive shorter distances between tasks. Figure 5-12a shows the final result after path planning according to the RB-SR method and Figure 5-12b shows the result for the MILP-SR method. It can be seen from Figure 5-12b that less time is spent driving between tasks with the MILP-SR method.

There is no notable difference between MILP-SR and MILP-MR, except when it comes to



(a) Integration method RB-SR



(b) Integration method MILP-SR

Figure 5-12: Scheduling and path planning results using integration methods RB-SR and MILP-SR. Robots are shown on the vertical axis, the tasks they have completed are shown as coloured blocks. The colours correspond to the colours from the task set in Figure 5-2b. Numbers above the blocks denote the pick-up and delivery station of the task. Dashed blocks represent driving from the finish station of one task to the start station of the next and grey blocks represent loading and unloading time. When there is empty space between arrival and unloading the robot is in the buffer of the station.

computation time. This due to the fact that in an environment with a lot of free space like the factory floor map, using the multi-robot path planner has negligible effect on path durations because there are plenty collision free paths with the same duration.

Optimise time windows

The second experiment with the factory floor map is done using the randomly generated task set with deadlines from Figure 5-2b. The scheduler objective function (5-3) is used to minimise the percentage of deadlines missed and the start and finish times of tasks to a lesser extent. Results are shown in Figure 5-13 and are reported in Table A-7.

Contrary to the previous experiment, in this case RB-SR outperforms MILP-SR and MILP-MR in some cases. This shows that even though MILP-SR and MILP-MR optimise their schedules for a minimal percentage of deadlines missed, they are not guaranteed to perform better than an integration method using a rule-based scheduler in this case. The reason for this is two-fold. First, due to having a stopping criterion of 120 seconds, the MILP is not able to find the optimal solution. This stopping criterion was decided on for practical reasons as well as because a test of a simulation running for 10 hours showed marginal improvement of the solution after the two-minute mark.

The second reason is that even if an optimal schedule were found by the MILP scheduler, it still uses a heuristic in the form of a pre-calculated duration table to estimate the duration of fulfilling tasks. These durations do not take collision avoidance between robots into account. Therefore, once paths have been calculated by the MILP path planner, more deadlines can be

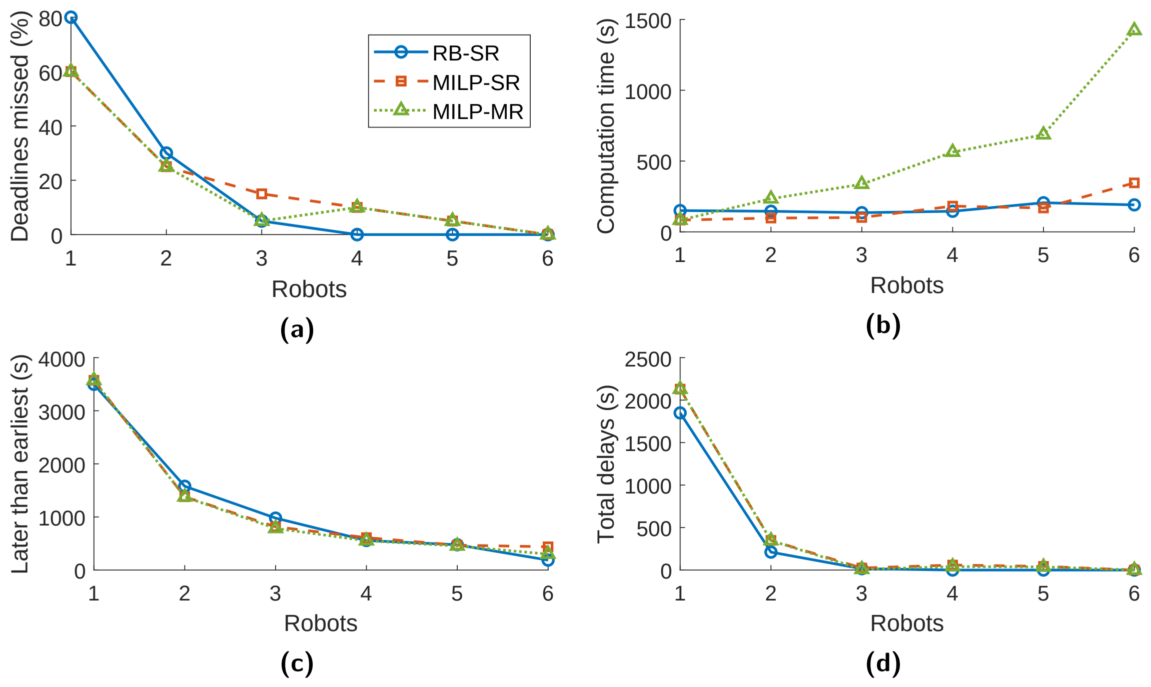


Figure 5-13: Comparison of three path planning and scheduling integration methods using the factory floor map and the task set with deadlines

missed than would have happened with another schedule. This is because the MILP scheduler does not take collision avoidance between robots into account which causes path durations to be longer.

The computation time follows the same pattern as with the experiment without deadlines. No experiments were done for upwards of 6 robots because computation time became longer than 30 minutes.

5-6-4 Corridor simulations

Two experiments are done with the corridor map shown in Figure 5-1b. This map features a lot of robot interactions, since robots cannot pass each other in the corridors and have to drive round obstacles to avoid other robots.

Optimise makespan

The first experiment with the corridor map uses the task set without deadlines from Figure 5-3a. Since there are no deadlines, scheduling objective function (5-2) is used to optimise for the makespan and start and finish times of tasks.

Figure 5-14a shows that MILP-MR outperforms RB-SR and MILP-SR for all numbers of robots. Neither RB-SR nor MILP-SR outperforms the other consistently. This leads to the conclusion that multi-robot path planning is responsible for the greater performance of MILP-MR and not MILP scheduling, since the single-robot methods do not outperform each

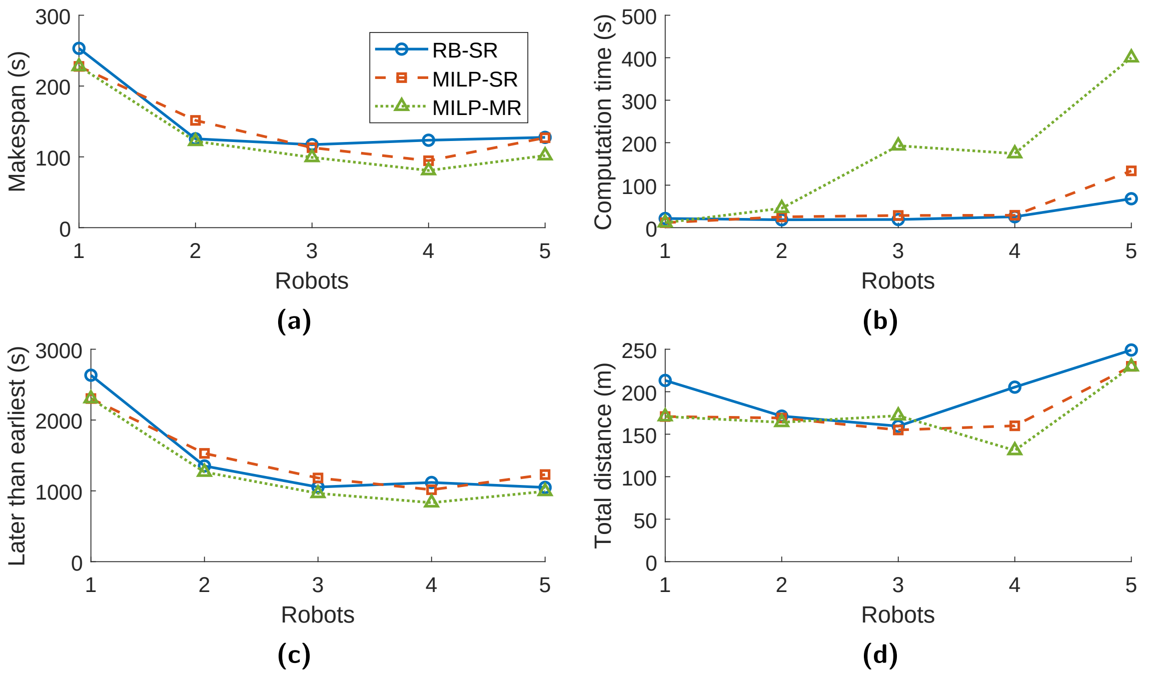


Figure 5-14: Comparison of three path planning and scheduling integration methods using the corridor map and the task set without deadlines

other. This better performance by MILP-MR does come at the cost of computation time as can be seen from Figure 5-14b.

Optimise time windows

The second experiment with the corridor map uses the task set with deadlines from Figure 5-3b. Objective function (5-3) is used for the scheduler to minimise the percentage of deadlines missed. Performance indicators are shown in Figure 5-15.

For this experiment MILP-MR misses fewer deadlines than RB-SR and MILP-SR for all simulations except with 4 robots. Figure 5-16 shows all the paths and the robot locations at $t = 9.25$ s for the simulation with method MILP-MR and 4 robots. The colours of the robots correspond to the path colours. Robot 1 is blue, robot 2 is green, robot 3 is red and robot 4 is cyan.

An anomaly occurs with the computation time of MILP-SR for the simulation with 5 robots. The reason for this is that in this specific case a highly constrained path planning problem occurs due to previously planned paths that takes a long time to compute.

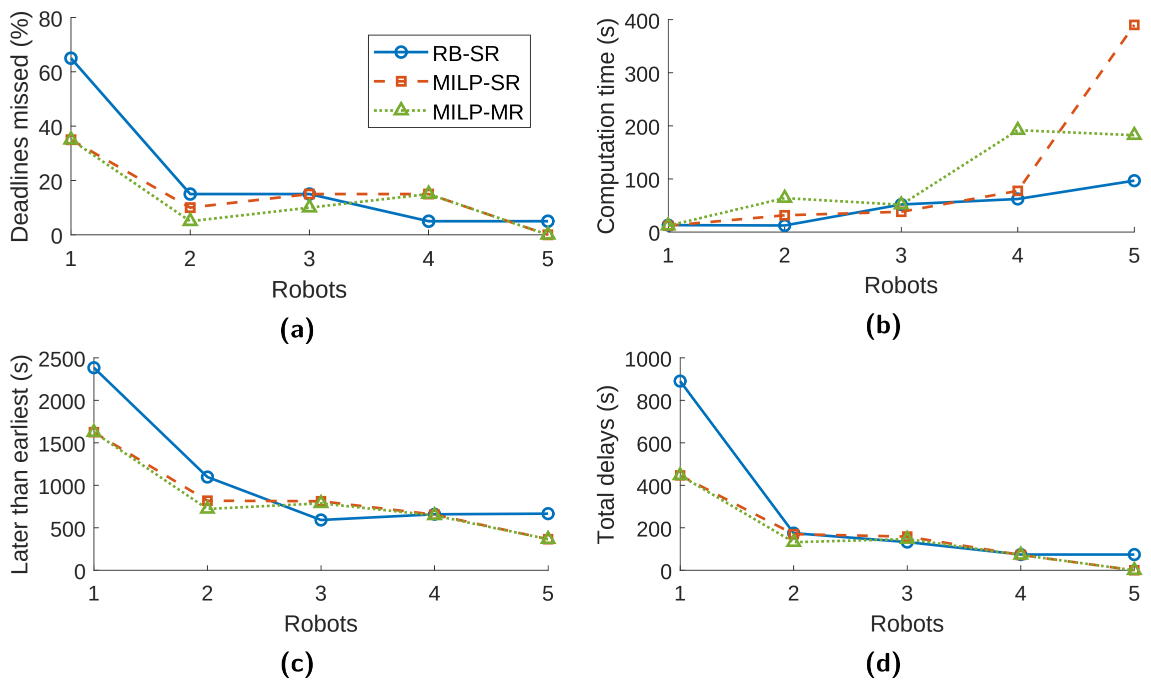


Figure 5-15: Comparison of three path planning and scheduling integration methods using the corridor map and the task set with deadlines

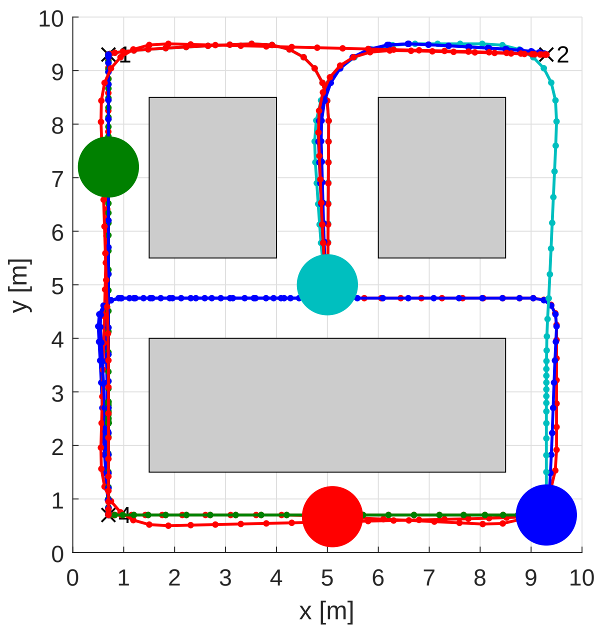


Figure 5-16: Paths and robot locations at time $t = 9.25$ s for the simulation with method MILP-MR and 4 robots using the corridor map

5-7 Conclusion

Four objective functions for the MILP path planner from Chapter 3 are compared. The cost function J_{pp} from (5-1) that measures path durations and has small weights on the state error and inputs is shown to perform best when considering path durations, computation time and a practical set of control actions. Computation time could be improved by selecting an objective function that purely measure duration, but this comes at the cost of high control actions and oscillating behaviour.

Two objective functions for the MILP scheduler are compared with a rule-based scheduling that uses the Earliest Deadline First (EDF) rule. The first minimises the completion time of the final task, also called the makespan, for a task set without deadlines and the second minimises the percentage of deadlines missed when deadlines are imposed on the tasks. The MILP scheduler is shown to be able to outperform rule-based scheduling in principle. However, a major issue is that it requires many hours of computation time to find an optimal solution. When a practical time limit of 120 s is imposed, depending on the number of robots it may be outperformed by the EDF rule. In a case with a task set with deadlines, it was even outperformed by EDF in nearly all tests.

Four more simulations are done comparing three integration approaches for task scheduling and path planning, with two different physical environments and for task sets with and without deadlines. The advantage of multi-robot path planning used by the MILP-MR method is negligible for a factory floor map where robots have plenty of space, while it does come at a significant cost of computation time. For a map with corridors that forces interactions between robots multi-robot path planning does have a positive effect on the makespan and the number of deadlines missed, though it still requires more computation time, especially with more robots.

The inconsistent performance of the MILP scheduler under limited computation time also becomes apparent with the integration methods MILP-SR and MILP-MR that use it. Multi-robot path planning as used by the method MILP-MR is able to recoup some of the losses by the MILP scheduler on the map with narrow corridors, allowing it to outperform the method RB-SR that uses rule-based scheduling in most cases but not all cases. This leads to the conclusion that multi-robot path planning can be beneficial in restricted environments, at the cost of larger computation time, but that it is dependent on the task set and the number of robots whether MILP scheduling is able to produce a better result than rule-based scheduling within an acceptable amount of computation time.

Chapter 6

Conclusions and future work

In this final chapter conclusions are drawn from the research of this thesis. Novel contributions are discussed and suggestions are made for further research on this topic.

6-1 Conclusions

In Chapter 2 an evaluation was made of the current state of the art on the topics of path planning and task scheduling. A wide body of literature exists on the path planning problem. Research was categorised into the topics of static environments, kinodynamic planning, dynamic environments and multi-robot path planning. Task scheduling as a topic of research is found in the fields of project management, optimisation and operations research. However, what has not received sufficient attention is an evaluation of methods to integrate task scheduling and path planning. Typically, path planners use initial and goal states as a given and task schedulers assume durations of tasks in advance. This research has therefore focussed on the development and evaluation of methods for integrating task scheduling and path planning.

The contribution of this thesis is threefold. First, in Chapter 3 a novel objective function is proposed for a path planner that uses an optimisation method called Mixed-Integer Linear Programming (MILP). The objective function from the literature is based on a linear quadratic regulator that aims to minimise the difference between the goal state of the robots and the state at all time instances within a finite time horizon. The new objective function minimises the duration to reach the goal state, which is more applicable in combination with scheduling, since to meet deadlines it is more important that tasks can be fulfilled quickly than that distances are short for example. The path planner does have some drawbacks. Obstacles have to be represented as polygons and produce an increasing number of constraints for more complex shapes. The vehicle model deviates somewhat from actual robot behaviour because it has to be linear. Linearity is also a cause for inaccuracies in distance and maximum speed calculations. A fixed time horizon is used which means large time horizons result in large problems. The free space around obstacles needs to be reserved to prevent the robot from cutting corners, reducing the total area of free space. And finally, the number of variables and constraints scales with the number of robots, resulting in large matrices for multi-robot path planning.

The second contribution of this thesis is a task scheduler for multiple robots using MILP that was introduced in Chapter 4. The new scheduling formulation was adapted from a task scheduler in the field of project management, to account for mobile robots completing tasks concurrently and driving between stations. The task scheduling problem is suited to a MILP approach, because a finite number of possible assignments of tasks to robots exists, mimicking the decision tree that is solved by a MILP optimiser. The formulation uses switching variables to relate start and finish times of tasks to a finite number of events. As a result the time horizon has no effect on the problem size. The problem size is only dependent on the number of robots and tasks.

The third contribution of this thesis consists of two methods for integrating task scheduling and path planning using the path planner from Chapter 3 and the task scheduler from Chapter 4. The aim of these integration methods is to generate a schedule and the corresponding paths of the robots that complete a set of tasks. The first method is called MILP-SR and plans paths for one robot at a time. The second method is called MILP-MR and uses multi-robot path planning. The methods are proposed in Chapter 5 and their performance is compared to that of a method called RB-SR that uses a rule-based scheduling approach in combination with single-robot path planning.

Experiments to evaluate the path planner, scheduler and integrated approach were done in Chapter 5. The novel path planning objective function was found to strike the best balance between path durations, total distance driven, energy use and computation time. The MILP scheduler is able to obtain a schedule with lower makespan or number of deadlines missed than a rule-based Earliest Deadline First (EDF) approach. However, when a limit of 120 s is set on the computation time for the practical reason of applicability in an on-demand scenario, the scheduler is not able to consistently outperform the rule-based approach, especially for task sets with deadlines where the objective is to minimise the number of missed deadlines. As a result, the integration methods MILP-SR and MILP-MR are not guaranteed to deliver better performance than the RB-SR method. However, in a narrow physical environment with many robot interactions the multi-robot path planner used by MILP-MR is able to improve upon the performance MILP-SR that plans paths for robots individually, regaining some of the losses caused by the MILP scheduler.

This leads to the conclusion that a different scheduling method that converges to a better solution than a rule-based method but in less time than the MILP scheduler proposed in this thesis, in combination with the multi-robot MILP path planner, could be a better approach for integrating scheduling and path planning. Since the MILP scheduler does not converge to an optimal solution in a practical timespan anyway, a nonlinear scheduling method that does not guarantee an optimal solution but at least outperforms EDF within a reasonable computation time is a better choice for a scheduler. Some examples of optimisation methods that could be used are genetic algorithms, particle swarm optimisation and ant colony optimisation.

6-2 Research questions

In Chapter 1 the main research question for this thesis was posed, as well as sub-questions concerning aspects of the main research topic. The main research question was formulated as follows:

What integrated task scheduling and path planning methodology for mobile robots assisting a production line can be developed, considering computational constraints, to optimise for the performance of the production line?

The sub-questions are answered below after which an answer to the main research question is given.

1. **How can the problems of task scheduling and path planning of mobile robots be formulated to allow for an integrated approach?**

   MILP formulations are proposed for the path planning problem and the task scheduling problem. The path planner has the objective to minimise durations of paths for robots completing tasks. Depending on whether deadlines are imposed on a set of tasks, the scheduler optimises for the makespan of a task set or the number of missed deadlines. The scheduler assigns tasks with pick-up and delivery locations to robots, while the path planner uses these locations to plan optimal paths to complete the tasks.

2. **What integrated task scheduling and path planning approach for mobile robots can be developed to optimise performance of a production line?**

   Two methods integrating the MILP scheduler and with a single-robot and a multi-robot path planner are proposed and compared with an integration method using rule-based scheduling and single-robot path planning. Two cases are used to model the performance of a production line. One uses a task set without deadlines, where the goal is to complete all tasks in as little time as possible. The second uses a task set with deadlines, where the objective is to minimise the number of missed deadlines.

3. **How do integrated approaches to path planning and task scheduling of mobile robots assisting a production line compare, as measured by the performance of the production line?**

   Between the two methods using MILP scheduling, the method that uses multi-robot path planning only outperforms the single robot-path planning method in an restrictive environment where robots have to take detours to avoid collisions. The MILP scheduler requires very long computation time to reach an optimal solution. When a practical computation time limit is imposed, integration methods using MILP are not able to consistently outperform the method that uses rule-based scheduling. The performance becomes even worse for task sets with deadlines.

The main research question can therefore be answered as follows. A multi-robot MILP path planner and MILP scheduler can be integrated to handle both task scheduling and path planning of a set of tasks with or without deadlines. Practical limits on computation time can cause methods integrating these path planning and scheduling optimisations to perform worse than a method using a rule-based scheduling approach.

6-3 Recommendations

Based on the conclusions from the research of this thesis that are drawn above, the following recommendations are made to Prodrive Technologies:

- Before applying a multi-robot MILP path planner to a specific physical environment and number of robots, investigate whether the case would benefit from multi-robot path planning or if single-robot path planning would suffice. Multi-robot path planning comes at a significant cost of computation time while the benefits are not always apparent.
- Design factory floors with enough room for robots to pass each other without drastically having to change their route. This allows for the use of less sophisticated path planning techniques and can help to reduce computation time for finding high-quality paths.
- Simulate the effects of collision avoidance between robots before committing to a certain schedule. Especially in restrictive environments, detours to prevent collisions can have a drastic effect on the feasibility and quality of a schedule.
- Evaluate rule-based and other fast scheduling methods for a possible integration with a MILP path planner.

6-4 Future work

Many challenges and unresolved questions remain when it comes to integrating task scheduling and path planning. An overview is made of topics for further research.

Rescheduling

The methods MILP-SR and MILP-MR generate a schedule one time, after which assignments of tasks to robots are fixed. Since the MILP scheduler uses a table to estimate the duration of completing a task, task durations after path planning can be different than was estimated. Generating a new schedule with the tasks remaining could result in a better overall schedule.

Dynamic duration table

The duration table used by the scheduler has fixed durations between stations. By making duration estimates dependent on parameters such as the number of robots, their current positions and the task set their accuracy could be improved, which could result in a more accurate schedule. The dynamic duration table could be implemented using a custom model or by training a model with machine learning.

Path planning objective function with deadlines

The MILP path planner in this thesis minimises the duration for robots to reach their goals, which can cause some robots to arrive early while others arrive late. By incorporating task deadlines into the path planner objective function, priority could be given to robots that have more trouble being on time.

Vehicle model

The linear vehicle model used by the path planner is a simplified model of a mobile robot. Other vehicle models could model the vehicle more accurately or provide computational benefits. For example, the force control model could be extended to include friction. Multiple vehicle models could be used depending on the location of the robot, for example to restrict maximum speed in certain zones for safety purposes.

Unknown obstacles

In this research, obstacle positions are assumed to be known over the full length of the time horizon. In reality, obstacles unknown to the path planner may appear an need to be avoided. The effect of avoidance of unknown obstacles on the integration of task scheduling and path planning needs to be further investigated.

Model predictive control

Longer time horizons for the MILP path planner result in larger constraint matrices and a larger number of variables, which can cause computation time to become prohibitively long. When paths need to planned over long time horizons or distances, model predictive control can be used to plan paths over a shorter prediction horizon. The first resulting control actions are applied and a new path is planned with prediction horizon farther into the future, until the goal is reached. In this way the single optimisation with a long time horizon is split into multiple optimisations with shorter time horizons, which can reduce computation time. However, the resulting path is not guaranteed to be globally optimal.

Robot buffers

The integration methods developed in this thesis do not plan manoeuvres for sending robots to a buffer location after completing a task to clear space at a station. Instead, robots disappear after unloading and reappear for a new task if the station is free. Research could be done into developing an efficient strategy for robots that momentarily do not have an assignment. For example, if there are more stations than robots, a robot could be sent to an unoccupied station if it needs to make way for another robot at its current station. Another possibility would be to reserve certain zones in accessible locations in the physical environment as buffer zones where robots would go after completing a task.

Nonlinear optimisation

Linear formulations were made for both the path planning and the task scheduling problem. The main reason for this is that these models are adequately accurate and that linear programming solvers are well-researched and very efficient. An evaluation could be made of the effect of using a more accurate nonlinear vehicle model, one that takes steering angle into consideration for example, on computation time and solution quality. Since experiments with some tasks sets have shown that the MILP scheduler cannot reach an optimal solution within an acceptable time frame, other formulations using MINLP (Mixed-Integer Nonlinear Programming) or other nonlinear optimisation methods such as genetic algorithms, particle swarm optimisation or ant colony optimisation might be attempted. Even if they do not guarantee an optimal solution, they may find a better solution under limited computation time.

Single problem formulation

A final suggestion is the development of a single problem formulation that optimises the schedule as well as the underlying paths. The integration methods in this thesis exchange information between the scheduler and the path planner, each optimising for its own objective function. Instead, a single objective function could be developed to optimise a schedule, while task durations directly depend on the paths that robots take to complete the tasks. However, as computation time is an issue for both the path planner and the scheduler in this thesis, such a combined optimisation is likely to be very computationally expensive.

Appendix A

Tables

This appendix contains tables with data that are the basis for simulations and figures in this report.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0	14	23.5	19	27.5	26	28	33
2	14	0	11.5	14.5	18	21.5	23.5	24.5
3	23.5	11.5	0	24	16.5	28	23	23
4	19	14.5	24	0	21	10	14.5	19.5
5	27.5	18	16.5	21	0	20	14.5	10
6	26	21.5	28	10	20	0	7	12
7	28	23.5	23	14.5	14.5	7	0	7
8	33	24.5	23	19.5	10	12	7	0

Table A-1: Minimum durations in seconds between stations of the factory floor map from Figure 5-1a

	1	**2**	**3**	**4**	**5**
1	0	6.5	7	6.5	11.25
2	6.5	0	7	11.25	6.5
3	7	7	0	7.5	7.5
4	6.5	11.25	7.5	0	6.5
5	11.25	6.5	7.5	6.5	0

Table A-2: Minimum durations in seconds between stations of the corridor map from Figure 5-1b

Case	Objective	J_1	J_2	J_3	J_4	1 Computation time (s)	2 Relative gap tolerance (%)	3 Distance (m)	4 Energy (10^3 J)	5 Duration (s)
1	J_1	3556	34035	119	154.7	120.0	13.10	113.8	14.0	59.5
	J_2	3545	31491	112	145.5	120.0	11.20	110.9	10.5	56
	J_3	10910	150347	107	259.6	38.6	0.00	112.5	15.6	53.5
	J_4	3488	31205	107	139.6	87.5	0.00	110.5	12.4	53.5
2	J_1	4576	48650	168	218.2	120.0	9.97	193.9	9.0	84
	J_2	4610	44800	155	202.2	120.0	9.32	159.2	22.0	77.5
	J_3	15387	231315	151	385.6	55.6	0.00	115.4	32.4	75.5
	J_4	4444	43207	151	196.3	114.9	0.00	154.3	21.0	75.5
3	J_1	3540	35028	159	195.3	53.2	0.00	148.2	16.4	79.5
	J_2	3566	34535	159	195.3	40.2	0.00	149.8	15.2	79.5
	J_3	15729	227716	155	386.1	31.8	0.00	152.4	25.3	77.5
	J_4	3563	34915	155	191.4	31.8	0.00	149.0	17.7	77.5

Table A-3: Simulation results for three cases comparing four objective functions for the path planner using the factory floor map

Method	Robots	J_{makespan}	Scheduler computation time (s)	Time windows missed (%)	Total delays (s)	Later than earliest (s)	Makespan (s)
RB	1	336.75	0	0	0	2374.5	313
	2	177.72	0	0	0	1221.5	165.5
	3	129.33	0	0	0	882.5	120.5
	4	111.78	0	0	0	677.5	105
	5	80.46	0	0	0	544.5	75
	6	72.71	0	0	0	470.5	68
	7	72.52	0	0	0	451.5	68
	8	42.04	0	0	0	303.5	39
MILP	1	302.1	120	0	0	2360.5	278.5
	2	166.98	120	0	0	1198.5	155
	3	115.85	120	0	0	834.5	107.5
	4	100.44	120	0	0	694.5	93.5
	5	87.15	120	0	0	615	81
	6	65.23	120	0	0	473.5	60.5
	7	55.12	120	0	0	362	51.5
	8	35.7	120	0	0	269.5	33

Table A-4: Simulation results comparing rule-based and MILP scheduling using the factory floor map and task set without deadlines

Method	Robots	J_{makespan}	Scheduler computation time (s)	Time windows missed (%)	Total delays (s)	Later than earliest (s)	Makespan (s)
RB	1	127.91	0	80	1850	3498	459
	2	58.55	0	30	210.5	1574.5	260
	3	26.61	0	5	16	955	197
	4	16.94	0	0	0	529	157
	5	13.98	0	0	0	261	157
	6	12.21	0	0	0	122	157
MILP	1	110.16	120	60	2241	3723	472
	2	73.13	120	40	723	2020	289
	3	31.57	120	10	25.5	864	204.5
	4	41.4	120	20	106	847	194.5
	5	24.98	120	5	30	705	187
	6	18.1	120	0	0	517	157

Table A-5: Simulation results comparing rule-based and MILP scheduling using the factory floor map and task set with deadlines

Method	Robots	Scheduler computation time (s)	Path planner computation time (s)	Total distance (m)	Energy (10^3 J)	Time windows missed (%)	Total delays (s)	Later than earliest (s)	Makespan (s)	Path planner MILP runs
RB-SR	1	0	23.1	432.5	28.7	0	0	2374.5	313	15
	2	0	42.6	452.5	31.8	0	0	1277.5	186	15
	3	0	44.9	418.7	27.2	0	0	926.5	122.5	14
	4	0	27.4	366.9	22.3	0	0	736	105.5	14
	5	0	15.9	349.4	32.8	0	0	599	78	15
	6	0	19.1	323.6	31.4	0	0	532	76	13
	7	0	43.7	370.8	43.7	0	0	532	76	14
MILP-SR	1	60	23.4	367.3	23.4	0	0	2119.5	278	13
	2	60	17.1	343.4	23.6	0	0	1101.5	137	13
	3	60	16.2	353.8	23.2	0	0	769.5	100.5	13
	4	60	11.4	313.6	22.2	0	0	540.5	68	12
	5	60	20.7	349.5	23.4	0	0	546	66	13
	6	60	14.2	306.6	21.3	0	0	431	54.5	12
	7	60	40.5	398.3	35.1	0	0	527.5	66	14
MILP-MR	1	60	23.4	367.3	23.4	0	0	2119.5	278	13
	2	60	32.2	343.5	23.6	0	0	1101.5	137	7
	3	60	58.8	353.3	22.9	0	0	769.5	100.5	5
	4	60	48.2	314	22.3	0	0	538.5	68	4
	5	60	80.2	351.6	24.3	0	0	562.5	69	5
	6	60	100.9	305.3	20.3	0	0	437	55	3
	7	60	177.8	396.8	35.8	0	0	509.5	66	4

Table A-6: Simulation results comparing three path planning and scheduling integration methods for the factory floor map and a task set without deadlines

Method	Robots	Scheduler computation time (s)	Path planner computation time (s)	Total distance (m)	Energy (10^3 J)	Time windows missed (%)	Total delays (s)	Later than earliest (s)	Makespan (s)	Path planner MILP runs
RB-SR	1	0	150.3	679.5	44.7	80	1849.5	3497.5	458.5	20
	2	0	145.4	672.1	53	30	211	1578	259.5	19
	3	0	135.4	612	55.7	5	16	978.5	196.5	19
	4	0	145.3	585.9	57.1	0	0	557	157	18
	5	0	205.6	703.1	80.9	0	0	476	157	18
	6	0	190.7	468.6	58.8	0	0	187	157	16
MILP-SR	1	120	83.1	592.3	43.6	60	2129.5	3574	451.5	19
	2	120	97.3	665.4	48	25	349	1380.5	262.5	19
	3	120	101.4	715.8	64.7	15	24	814.5	205	20
	4	120	181.8	673.8	58.8	10	59	611	183	19
	5	120	168.5	747.3	74.8	5	43.5	468.5	198.5	20
	6	120	345.9	691.4	76.9	0	0	438.5	157	19
MILP-MR	1	120	83.1	592.3	43.6	60	2129.5	3574	451.5	19
	2	120	233.7	665.8	48.5	25	347	1377.5	262	13
	3	120	336.8	717.3	65	5	8.5	784	199.5	9
	4	120	563	675.4	59.1	10	41	552	174	10
	5	120	686.2	746.6	66.8	5	37	454	196.5	10
	6	120	1422.5	690.3	69.3	0	0	300	157	7

Table A-7: Simulation results comparing three path planning and scheduling integration methods for the factory floor map and a task set with deadlines

Method	Robots	Scheduler computation time (s)	Path planner computation time (s)	Total distance (m)	Energy (10^3 J)	Time windows missed (%)	Total delays (s)	Later than earliest (s)	Makespan (s)	Path planner MILP runs
RB-SR	1	0	21.6	213.3	20	0	2633.25	0	253.25	18
	2	0	18.6	171.3	14.8	0	1351.5	0	125.5	16
	3	0	19.2	159.5	19.8	0	1053.75	0	117.25	15
	4	0	25.7	205.4	27	0	1118.25	0	123.5	17
	5	0	68.1	249.1	33.7	0	1048.5	0	127.5	19
MILP-SR	1	60	11.7	170.8	16	0	2303.75	0	227.75	17
	2	60	25.4	169.1	14.7	0	1529.5	0	151.5	16
	3	60	28.7	155	12	0	1182	0	113	14
	4	60	29.2	159.9	18.5	0	1016	0	94.5	12
	5	60	133.7	229.9	24.5	0	1230.25	0	127	19
MILP-MR	1	60	11.7	170.8	16	0	2303.75	0	227.75	17
	2	60	45.7	164	14.5	0	1265.75	0	122	9
	3	60	192.8	171.8	16.7	0	964.75	0	99.25	6
	4	60	174.4	130.9	14.7	0	833	0	80.75	7
	5	60	400.2	229.5	25.4	0	993	0	101.8	10

Table A-8: Simulation results comparing three path planning and scheduling integration methods for the corridor map and a task set without deadlines

Method	Robots	Scheduler computation time (s)	Path planner computation time (s)	Total distance (m)	Energy (10^3 J)	Time windows missed (%)	Total delays (s)	Later than earliest (s)	Makespan (s)	Path planner MILP runs
RB-SR	1	0	13.1	178.6	21.1	890.5	2383.75	65	305.5	19
	2	0	12.4	155.5	22.6	174.75	1097	15	206.5	17
	3	0	51.9	195.8	24.3	132.75	591.75	15	206.5	18
	4	0	62.3	157.7	20.7	74	658.75	5	189	17
	5	0	96.8	200	26.1	74	666.25	5	189	19
MILP-SR	1	120	12	161.6	18.5	446.5	1624.75	35	269.5	17
	2	120	31.6	191.6	24.3	170	818.75	10	212	19
	3	120	38.2	184.5	23.3	158.5	812.25	15	200	19
	4	120	77.6	179.5	24.6	72	654.75	15	217.25	17
	5	120	390.1	208.7	24.9	0	365.5	0	189	18
MILP-MR	1	120	12	161.6	18.5	446.5	1624.74	35	269.5	17
	2	120	64	195.7	25.3	132.25	721.75	5	189.25	12
	3	120	51.5	195.1	23.6	146.75	787.75	10	192.75	13
	4	120	191.9	186.8	27.6	72	645.25	15	217.25	12
	5	120	182.4	187.7	27.6	0	365.5	0	189	11

Table A-9: Simulation results comparing three path planning and scheduling integration methods for the corridor map and a task set with deadlines

Appendix B

Figures

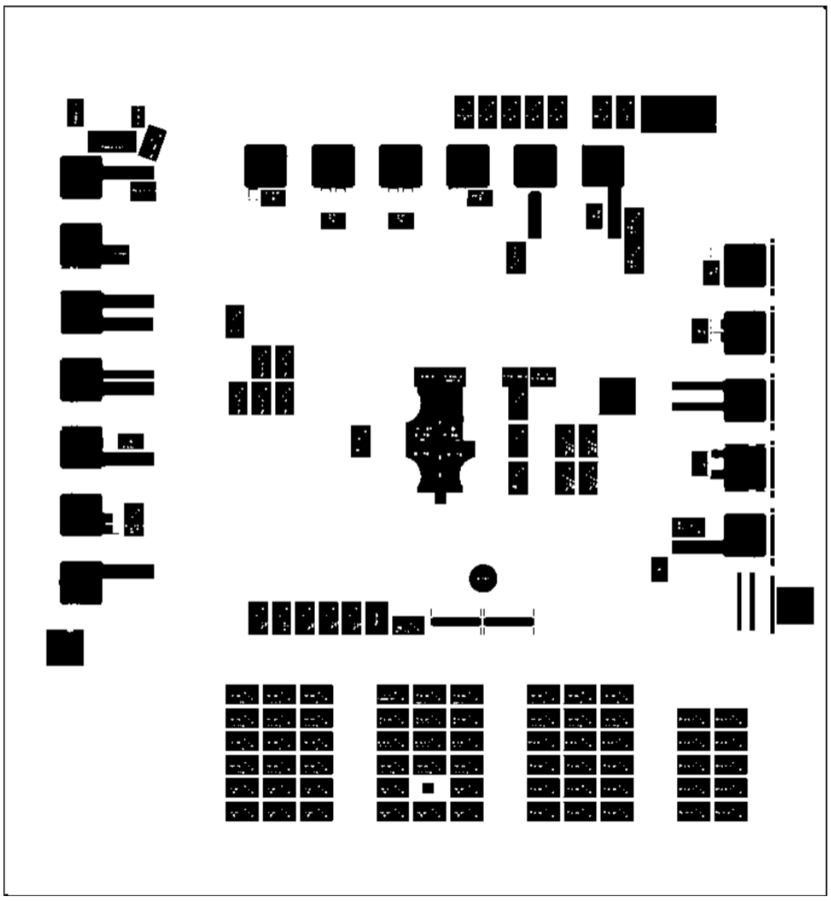


Figure B-1: Two-dimensional scan of a factory floor at Prodrive Technologies

Bibliography

[1] S. An and H. J. Kim, “Simultaneous Task Assignment and Path Planning Using Mixed-Integer Linear Programming and Potential Field Method,” in *International Conference on Control, Automation and Systems*, Gwangju, Korea, 2013, pp. 1845–1848.

[2] C. Artigues, P. Brucker, S. Knust, O. Koné, P. Lopez, and M. Mongeau, “A Note on "Event-based MILP Models for Resource-constrained Project Scheduling Problems",” *Computers and Operations Research*, vol. 40, pp. 1060–1063, 2013.

[3] F. Balarin, L. Lavagno, P. Murthy, A. Sangiovanni-Vincentelli, and Others, “Scheduling for Embedded Real-Time Systems,” *IEEE Design & Test of Computers*, vol. 15, no. 1, pp. 71–82, 1998.

[4] J. Barraquand, L. E. Kavraki, J.-C. L. R. Motwani, T.-Y. Li, and P. Raghaven, “A Random Sampling Scheme for Path Planning,” *The International Journal of Robotics Research*, vol. 16, no. 6, pp. 759–774, 1997.

[5] T. Bektas, “The Multiple Traveling Salesman Problem: An Overview of Formulations and Solution Procedures,” *The International Journal of Management Science*, vol. 34, no. 3, pp. 209–219, 2006.

[6] A. Bemporad and M. Morari, “Control of Systems Integrating Logic, Dynamics, and Constraints,” *Automatica*, vol. 35, pp. 407–427, 1999.

[7] R. A. Brooks and T. Lozano-Pérez, “A Subdivision Algorithm in Configuration Space For Find Path with Rotation,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 15, no. 2, pp. 224 – 233, 1985.

[8] Z. Butler, “Corridor Planning for Natural Agents,” in *IEEE International Conference on Robotics and Automation*, 2006, pp. 499–504.

[9] B. Chazelle, “Approximation and Decomposition of Shapes,” in *Algorithmic and Geometric Aspects of Robotics*, J. T. Schwartz and C. K. Yap, Eds. Hillsdale, NJ: Lawrence Erlbaum Associates, 1987, pp. 145–185.

[10] Y.-L. Chen, L.-J. Hsiao, and K. Tang, “Time Analysis for Planning a Path in a Time-window Network,” *Journal of the Operational Research Society*, vol. 54, pp. 860–870, 2003.

[11] S. Coenen, “Motion Planning for Mobile Robots - A Guide,” *Repository Eindhoven University of Technology*, 2012.

[12] E. W. Dijkstra, “A Note on Two Problems in Connexion with Graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[13] M. Elbanhawi and M. Simic, “Sampling-Based Robot Motion Planning: A Review,” *IEEE Access*, vol. 2, pp. 56–77, 2014.

[14] P. Fiorini and Z. Shiller, “Motion Planning in Dynamic Environments Using Velocity Obstacles,” *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998.

[15] D. Fox, W. Burgard, and S. Thrun, “The Dynamic Window Approach to Collision Avoidance,” *IEEE Robotics and Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[16] T. Fraichard and H. Asama, “Inevitable Collision States - A Step Towards Safer Robots?” *Advanced Robotics*, vol. 18, no. 10, pp. 1001–1024, 2004.

[17] E. Frazzoli, M. A. Dahleh, and E. Feron, “Maneuver-Based Motion Planning for Nonlinear Systems With Symmetries,” *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 1077–1091, 2005.

[18] B. Friedland, *Control System Design: An Introduction to State-Space Methods.* Mineola, NY, USA: Dover Publications Inc., 2005.

[19] P. E. Hart, N. J. Nilsson, and B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths,” *IEEE Transactions of Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[20] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, “Randomized Kinodynamic Motion Planning with Moving Obstacles,” *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[21] D. Hsu, J.-C. Latombe, and H. Kurniawati, “On the Probabilistic Foundations of Probabilistic Roadmap Planning,” *The International Journal of Robotics Research*, vol. 25, no. 7, pp. 627–643, 2006.

[22] L. Jaillet and T. Siméon, “A PRM-based Motion Planner for Dynamically Changing Environments,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004, pp. 1606–1611.

[23] A. Kamphuis and M. H. Overmars, “Finding Paths for Coherent Groups using Clearance,” in *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, 2004, pp. 19–28.

[24] I. Kara and T. Bektas, “Integer Linear Programming Formulations of Multiple Salesman Problems and its Variations,” *European Journal of Operational Research*, vol. 174, no. 3, pp. 1449–1458, 2006.

[25] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic Roadmaps for Path Planning in High-dimensional Configuration Spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[26] O. Khatib, “Real-Time Obstacle Avoidance for Manipulators and Mobile Robots,” *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[27] O. Koné, C. Artigues, P. Lopez, and M. Mongeau, “Event-based MILP Models for Resource-constrained Project Scheduling Problems,” *Computers and Operation Research*, vol. 38, pp. 3–13, 2011.

[28] Y. Koren and J. Borenstein, “Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation,” in *IEEE International Conference on Robotics and Automation*, no. April, 1991, pp. 1398–1404.

[29] F. Lamiraux, E. Ferre, and E. Vallée, “Kinodynamic Motion Planning: Connecting Exploration Trees Using Trajectory Optimization Methods,” in *IEEE International Conference on Robotics and Automation*, 2004, pp. 3987–3992.

[30] G. Laporte and Y. Nobert, “A Cutting Planes Algorithm for the m-Salesmen Problem,” *Journal of the Operational Research Society*, vol. 31, no. 11, pp. 1017–1023, 1980.

[31] S. M. LaValle, “Rapidly-Exploring Random Trees: A New Tool for Path Planning,” 1998.

[32] S. M. LaValle, *Planning Algorithms.* Cambridge University Press, 2006.

[33] S. M. LaValle, “Motion Planning: Wild Frontiers,” *IEEE Robotics and Automation Magazine*, vol. 18, no. 2, pp. 108–118, 2011.

[34] S. M. LaValle and J. J. Kuffner, “Randomized Kinodynamic Planning,” *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[35] T. Y. Li and Y. C. Shie, “An Incremental Learning Approach to Motion Planning with Roadmap Management,” in *IEEE International Conference on Robotics and Automation*, no. May, 2002, pp. 3411–3416.

[36] T. Lozano-Pérez, “Spatial Planning: A Configuration Space Approach,” *IEEE Transactions on Computers*, vol. 32, no. 2, pp. 108–120, 1983.

[37] J. Melvin, P. Keskinocak, S. Koenig, C. Tovey, and B. Y. Ozkaya, “Multi-Robot Routing with Rewards and Disjoint Time Windows,” *IEEE International Conference on Intelligent Robots and Systems*, vol. 7, pp. 2332–2337, 2007.

[38] A. Modares, S. Somhom, and T. Enkawa, “A Self-Organizing Neural Network Approach for Multiple Traveling Salesman and Vehicle Routing Problems,” *International Transactions in Operational Research*, vol. 6, no. 6, pp. 591–606, 1999.

[39] C. Reinl and O. von Stryk, “Optimal Control of Multi-Vehicle-Systems Under Communication Constraints Using Mixed-Integer Linear Programming,” in *International Conference on Robot Communication and Coordination*, Athens, Greece, 2007.

[40] T. Schouwenaars, B. De Moor, E. Feron, and J. How, “Mixed Integer Programming for Multi-Vehicle Path Planning,” in *European Control Conference*, Porto, Portugal, 2001, pp. 2603–2608.

[41] A. Stentz, “Optimal and Efficient Path Planning for Partially-Known Environments,” in *IEEE International Conference on Robotics and Automation*, 1994, pp. 3310 – 3317.

[42] L. Tang, J. Liu, A. Rong, and Z. Yang, "A Multiple Traveling Salesman Problem Model for Hot Rolling Scheduling in Shanghai Baoshan Iron & Steel Complex," *European Journal of Operational Research*, vol. 124, no. 2, pp. 267–282, 2000.

[43] M. Turpin, K. Mohta, N. Michael, and V. Kumar, "Goal Assignment and Trajectory Planning for Large Teams of Interchangeable Robots," *IEEE International Conference on Robotics and Automation*, vol. 37, pp. 842–848, 2013.

[44] J. P. Van Den Berg, "Path Planning in Dynamic Environments," Ph.D. dissertation, Utrecht University, 2007.

[45] J. P. Van Den Berg, D. Ferguson, and J. J. Kuffner, "Anytime Path Planning and Replanning in Dynamic Environments," in *IEEE International Conference on Robotics and Automation*, no. May, 2006, pp. 2366–2371.

[46] J. P. Van Den Berg, M. Lin, and D. Manocha, "Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation," in *IEEE International Conference on Robotics and Automation*, 2008, pp. 1928–1935.

[47] J. P. Van Den Berg and M. H. Overmars, "Prioritized Motion Planning for Multiple Robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 2217–2222.

[48] J. P. Van Den Berg, J. Snoeyink, M. Lin, and D. Manocha, "Centralized Path Planning for Multiple Robots: Optimal Decoupling into Sequential Plans," *Robotics: Science and Systems V*, 2009.

[49] T. van den Boom and B. De Schutter, *Optimization in Systems and Control*. Delft: Delft University of Technology, 2015.

[50] G. Wagner and H. Choset, "Subdimensional Expansion for Multirobot Path Planning," *Artificial Intelligence*, vol. 219, pp. 1–24, 2015.

[51] H. P. Williams, *Model Building in Mathematical Programming*, 3rd ed. New York, NY, USA: Wiley, 1993.

Glossary

List of Acronyms

MPC	Model Predictive Control
MILP	Mixed-Integer Linear Programming
EDF	Earliest Deadline First
RB-SR	Rule-based scheduling, single robot path planning
MILP-SR	MILP scheduling, single-robot path planning
MILP-MR	MILP scheduling, multi-robot path planning

List of Symbols

$\mathcal{A}_p$	Robot p in set of robots $\mathcal{A}$, with $p = \{1, \dots, P\}$
$\boldsymbol{q}$	Weighting vector for the state vector $\boldsymbol{s}$ in the objective function
$\boldsymbol{r}$	Weighting vector for the control action vector $\boldsymbol{u}$ in the objective function
$\boldsymbol{s}_k$	State vector at time instance k
$\boldsymbol{u}_k$	Control action vector at time instance k
$\mathcal{E}_e$	Event e, with $e = \{0, \dots, L-1\}$
$\mathcal{O}_m$	Obstacle m in set of obstacles $\mathcal{O}$, with $m = \{1, \dots, M\}$
k	Discrete time instance, with $k = \{1, \dots, K\}$
T_n	Task n, with $n = \{1, \dots, N\}$