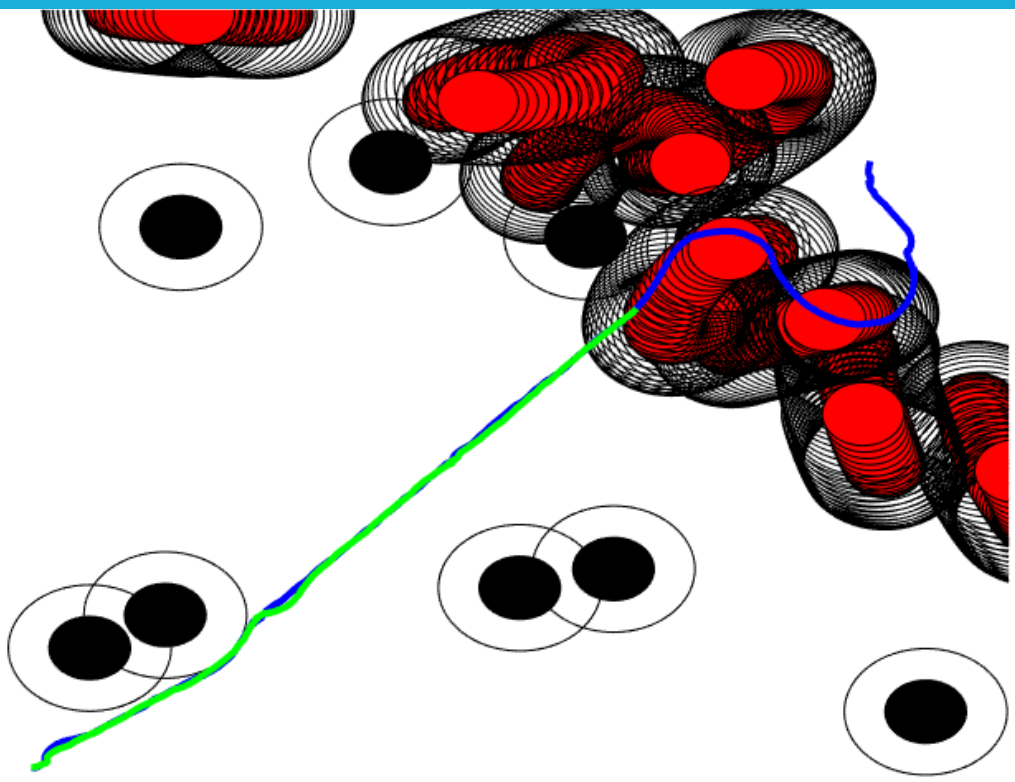
Development of a model predictive controller for motion planning in a dynamic urban search and rescue environment

K. Rado



TU Delft

Development of a model predictive controller for motion planning in a dynamic urban search and rescue environment

Thesis report

by

K. Rado

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on May 31, 2023 at 13:00

Thesis committee:

Dr. Ir. C.C. de Visser (chair)
Dr. A. Jamshidnejad
Dr.ir. E. Mooij
Dr. R.D. McAllister

Place:	Faculty of Aerospace Engineering, Delft
Project Duration:	July, 2021 - May, 2023
Student number:	4212169

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Faculty of Aerospace Engineering · Delft University of Technology

Contents

Nomenclature

List of Abbreviations

APF	Artificial Potential Function
GNSS	Global Navigation Satellite Systems
HL-RRT	Horizon-based Lazy RRT
IMU	Inertial Measurement Unit
LiDAR	Light Detection and Ranging
MCDM	Multiple Criteria Decision Making
MPC	Model Predictive Control
PoA	Point of Attraction
POMDP	Partially Observable Markov Decision Process
RRT	Rapidly-exploring Random Tree
SaR	Search and Rescue
SLAM	Simultaneous Localization and Mapping
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
uSaR	Urban Search and Rescue

List of Symbols

α_1	Reference trajectory weights for the MPC cost function
α_2	Input weights for the MPC cost function
Δt_{ctrl}	Control update step
η_e	Emergency maneuver scaling factor for the APF
η_s	Static obstacle scaling factor for the APF
κ	Current control time step
ω_{rob}	Angular velocity of the robot
ρ_{det}	Sensor/detection/planning radius
ρ_{obs}	Obstacle radius
ρ_{rob}	Robot radius
$\underline{\mathbf{u}}_{\text{rob}}(0, H_c)$	Vector of robot inputs from $t = \kappa$ to $t = \kappa + H_c$
$\underline{\mathbf{x}}_{\text{rob}}(1, H_p)$	Vector of robot states from $t = \kappa + 1$ to $t = \kappa + H_p$
$\mathbf{x}_{\text{ref}}$	Vector of reference states
θ	Heading angle
H_c	Control horizon for the MPC
H_p	Prediction horizon for the MPC
t	Current time
v_{rob}	Linear velocity of the robot
w_{max}	Maximum noise value

List of Figures

List of Tables

Summary

Disasters are devastating to both communities and the areas they live in. Earthquakes, hurricanes, fires, and many more, happen in both urban and rural areas. The disaster-stricken environment turns inhospitable, and areas become harder to traverse or even inaccessible. To expand the search range, speed up the victim search, and improve the safety of the rescue workers, search and rescue (SaR) operations gradually adopted the use of robots. The purpose of SaR robots is thus to expedite the SaR process. Designing a more autonomous system requires less operators per robot, and allows the benefit of scale to support the victim detection mission in a more time efficient way.

To improve the search process of robots in SaR, preliminary research was done on both a local and a global scale by comparing decision making, motion planning approaches and obstacle avoidance methods. This is to realize the mission goal of victim detection and dynamic obstacle avoidance. The problems the controller needed to address shaped the general design, where both motion planning and SaR literature pointed towards decoupling the local control problem, which includes obstacle avoidance, from the global optimization.

By addressing the general research objective, a key problem was found. The dynamic environment and lack of accurate knowledge on top of the post-disaster clutter means many algorithms for obstacle avoidance and motion planning cannot be used. To this end, the research question in the scientific article has been focussed from both the local and global design approach to a local motion planning design, a necessary step to realize better performance of SaR robots.

The scientific article develops such a controller by coupling a local greedy heuristic motion planner to a tube robust MPC trajectory tracker, which is robust to a bounded additive noise on the positions of the dynamic obstacles and the position of the robot itself. This is subsequently simulated in a case study to compare performance in terms of path length, time to reach the goal, and success rate in reaching the goal without collisions. Comparing the developed algorithm with two contemporary algorithms based on rapidly-exploring random tree (RRT) and artificial potential function (APF) theory, the MPC based algorithm outperformed them both in path length and success rate in cases with and without a computational budget.

The article in Part I contains the scientific article which addresses the research question of how effective a bi-level MPC approach is in robust control for trajectory generation and tracking in a dynamic environment. Part II contains the already graded preliminary report that discusses both the global and local perspective. It contains background information and the research questions. Some questions have already been answered at that stage, which meant the focus of the thesis was shifted to the development and evaluation of a local trajectory planner and tracker. Appendix A presents additional material useful for repeating the results of the main article.

Part I

Scientific Article

Development of a model predictive controller for motion planning in a dynamic urban search and rescue environment

Student: Karlo Rado (MSc Student)
Supervisors: M. Baglioni & Dr. A. Jamshidnejad
Control and Operations, Faculty of Aerospace Engineering
Delft University of Technology
Delft, the Netherlands

***Abstract*—A key challenge for SaR robotics is to avoid dynamic obstacles in cluttered environments, with limited and noisy information. In this research, a controller for SaR robots is developed by coupling a local heuristic motion planner with a model predictive control (MPC) based trajectory tracker. Constraint tightening and tube-based control are used to make the MPC robust to model mismatch and additive measurement noise, while the motion planner is integrated with the MPC. The motion planner periodically supplies a reference trajectory to the trajectory tracker, but the MPC can request additional updates in case of a noticeable mismatch between the predicted and measured environment, based on a user-defined threshold. A case study is designed in MATLAB where a single robot needs to reach a goal through a cluttered environment with dynamic obstacles. Results from the case study show that the MPC method outperforms two state-of-the-art control approaches that are based on the rapidly-exploring random tree (RRT) and artificial potential function (APF) methods. In particular, the heuristic and MPC coupled controller showed a higher success rate in reaching the goal without collisions, and displayed a lower path length in cases with both low and high computational budget.**

***Index Terms*—Motion planning, mobile robots, local trajectory generation, dynamic obstacles, obstacle avoidance, tube-based model predictive control, robust control, urban search and rescue.**

ACRONYMS

APF Artificial Potential Function.

HL-RRT Horizon-based Lazy RRT.

MPC Model Predictive Control.

PoA Point of Attraction.

RRT Rapidly-exploring Random Tree.

SaR Search and Rescue.

uSaR urban Search and Rescue.

NOMENCLATURE

$\boldsymbol{\alpha}_1$	Reference trajectory weights for the MPC cost function
$\boldsymbol{\alpha}_2$	Input weights for the MPC cost function
Δt_{ctrl}	Control update step
η_e	Emergency maneuver scaling factor for the APF
η_s	Static obstacle scaling factor for the APF
κ	Current control time step
ω_{rob}	Angular velocity of the robot
ρ_{det}	Sensor/detection/planning radius
ρ_{obs}	Obstacle radius
ρ_{rob}	Robot radius
$\underline{\mathbf{u}}_{\text{rob}}(0, H_c)$	Vector of robot inputs from $t = \kappa$ to $t = \kappa + H_c$
$\underline{\mathbf{x}}_{\text{rob}}(1, H_p)$	Vector of robot states from $t = \kappa + 1$ to $t = \kappa + H_p$
$\mathbf{x}_{\text{ref}}$	Vector of reference states
θ	Heading angle
H_c	Control horizon for the MPC
H_p	Prediction horizon for the MPC
t	Current time
v_{rob}	Linear velocity of the robot
$w_{\max}$	Maximum noise value

I. INTRODUCTION

Over the years, developments in robotics and control have yielded improvements in both the range of applications and the quality of their implementation. One such field that has experienced a continuous improvement is the control of robots in both autonomous motion planning and semi-autonomous exploration. This has been demonstrated through successful performances of robots in autonomous driving, cave exploration, as well as Search and Rescue (SaR) missions [1].

One of the primary variables for SaR missions is the time it takes to rescue a victim. The faster they are found, the greater the chance for survival, as rescue operations can be prioritized [2]. As mentioned in [3], the survival rates in finding victims on the first day is 81%, while it is 36.7% on the second day. Past events also show that the time taken for arrival and starting the operations is one of the main limiting factors for further contribution of SaR teams [4]. Robotics could help

improve the operational efficiency, by accessing unreachable or dangerous regions even before the search party starts their operations [5].

Earthquakes claimed over a million of lives in the 20th century, while the 2010 earthquake in Haiti alone totalled around 222,500 deaths [6]. Due to the change in environment, areas could become dangerous to traverse for the human rescuer, thus requiring additional time and information to establish proper situational awareness. Failing to establish proper situational awareness could cause hazards such as fires or gas leaks to endanger both the victims and the rescuers' lives [5].

Post-disaster, structural damage and collapsed buildings reshape the environment, making some areas hard to reach or even inaccessible. In cases like this, obtaining any information is useful. Even knowing that there is no victim requiring immediate rescue in an area can allow the rescue workers to focus their efforts elsewhere. With this in mind, different robots and control algorithms have been developed depending on the SaR task they solve or support. Examples include multi-target tracking for preventing losses and maintaining an overview of the area, providing ad-hoc networks in post-disaster environments for communication, mapping the post-disaster environments for increased situational awareness, or supporting the victim search task [5], [7]. From the many different disciplines within SaR robotics, they each share some aspects.

One of these aspects concerns itself with the path-finding and obstacle avoidance in possibly unknown or dynamic environments. Consider for instance victims or rescuers in motion; a robot should not crash into them. Path planning and motion planning address the problem of dynamic obstacle avoidance. From the many surveys and review papers addressing this theory (for SaR or general purpose [5], [8], [9], [10], [11]) it can be seen there are various solutions for the same problem, though rarely for a cluttered, time-constrained, and complex environment, which is often present in indoor SaR scenarios. One of the few theories that can address this is the one that forms the basis for Model Predictive Control (MPC).

As mentioned in [12], the key advantage of MPC and why it is often chosen for motion planning is the ability to robustly account for bounded disturbances and modifying constraints on states and inputs in the presence of new information. Additionally it is often selected because it provides online (on-the-go) stability and convergence guarantees [12], and is both complete (finds a path if one is available) and optimal (finds the best path given some heuristic) [9]. MPC is a control theory based approach that is being developed continually, with implementations varying from fast linear controllers [13], to more complex non-linear stochastic ones [14]. These MPCs vary in computational speed, accuracy, and the degree to which they are or can be made robust to noise or disturbances.

From all these possible motion planning algorithms and modifications, papers often list their performance without testing them side-by-side, often citing qualitative values (as is the case in [15]). A limiting factor is the lack of a unified test platform [16], and that ones available are for more mature software-in-the-loop approaches like uSARSim (which simulate full subsystems [17]).

This work aims to adapt a greedy heuristic path planner combined with an MPC-based motion planner [18] for dynamic obstacle avoidance, robust to the presence of noise on the positions of the robot and dynamic obstacles. The controller is tested against two state-of-the-art dynamic obstacle avoidance papers, based on Artificial Potential Function (APF) [15] and Rapidly-exploring Random Tree (RRT) [19], for both nominal and noisy environments using both low and very high iteration budgets per simulation. The results suggest that the newly developed method has a larger success rate in reaching the goal without collisions, and generally lower path length in both noisy and nominal environments, in cases that are constrained by a low or very high computational budget. The investigation is then used to suggest various improvements to both the MPC-based approach, as well as the approaches used in the comparison papers.

The main contributions of this thesis stem from the following three sources:

1) A heuristic path planner is modified to perform trajectory planning for noisy dynamic obstacle avoidance. Consequently, the motion planner finds the shortest path and generates a trajectory using the positions and last-known linear and angular velocities of obstacles in the environment.
2) A controller architecture is designed where the motion planner algorithm is coupled with a trajectory tracking tube-based MPC, resulting in a control scheme robust to additive uncorrelated bounded noise. The contribution is that a noise robust tube-based MPC is designed to account for bounded additive noise that also affects dynamic obstacle positions.
3) A case study is developed in MATLAB to compare the performance of the MPC approach with state-of-the-art control schemes, for a single robot performance. Thereby the algorithms are compared numerically with respect to their shortest path length, shortest time to goal, and highest success rate.

In order to do this, first a review is done in Section II that looks into other motion planning approaches as well as the modelling of the environment. Using this information, Section III presents the design considerations for simulations in terms of the model dynamics, while the problem is described and resolved through the motion planning controller design. Following, the comparison algorithms are introduced in Section IV, where some possible sources of errors and their effects in their implementations are addressed. Having this, a case study is developed and discussed in Section V, with the results shown through simulations in Section VI. The results are evaluated in terms of the dependent variables (success rate, path length and time taken to reach the goal) in Section VII, and a minor discussion follows that helps suggest possible uses or improvements for each approach. Section VIII concludes the work done in this thesis and includes some recommendations for future research.

II. Background theory

To address the victim search task, the best possible motion plan is to reach the goals without collisions, minimizing the path length or the mission time in the process. To this end, this section addresses the considerations for the motion planner. First, literature on the topics of MPC and noise robust theory is used to prepare the MPC implementation. Next, the motion planner is defined through taxonomy, its tasks and the knowledge it could or needs to assume. This helps define the problem, eliminating possible confusion in using wrong terminology. Following that, the obstacle and the robot (or vehicle) dynamics commonly used in theory are also looked into. Lastly, the key features used to simulate sensors and the environment are presented to help form the basic principles for the design of the case study.

A. Model predictive control

MPC is a control approach that works by minimizing or maximizing an objective function over a prediction horizon H_p. It does this by changing control inputs u over a control input horizon H_c ($< H_p$). Any predicted states, x, which are often used in the objective function, are found using a system model. Meanwhile, the states and inputs can be constrained to certain values within these horizons. This method results in a control scheme shown in Figure 1.

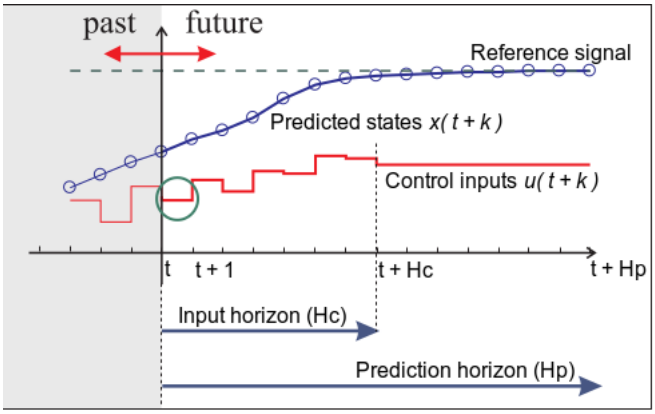


Fig. 1. Illustration to clarify the predictive and control (input) horizons in a reference tracking model predictive control approach.

The main reasons why MPC is often selected for motion planning is the ability to combine planning with online stability and convergence guarantees [12]. As shown in Figure 1, the control inputs ensure that the predicted outputs of the system follow the reference trajectory in a stable manner. Generally, stability is achieved using either a contraction constraint, a terminal set or a terminal weight [20]. Terminal sets are often pre-computed (such as in [21], [22]), but there are also online terminal set approaches, such as [23], where the terminal set is tightened over time based on bounded disturbances.

Though it is possible to simply use one source for stability (such as terminal cost in [24]), approaches can also use a combination. Examples are [25] that combines terminal cost and constraint tightening, or [26] using terminal cost and terminal set constraints for autonomous driving. This way the stability constraints can be tuned to be less conservative, especially when considering dynamic environments. The impact of these conditions on the control inputs is directly related to the inherent stability of a system, meaning these could result in more conservative control inputs in case of an autonomous car on the highway, compared to an autonomous non-differentially constrained vacuuming robot. In cases such as the autonomous cars, some instability can be accepted in favor of a larger control input range. Not having to expend computational effort in this direction can become beneficial when considering one of the main practical limitations in using MPC, the computational cost, which limits the prediction and control horizons in implementations.

There are other conditions to consider when designing an MPC, such as whether the model is continuous (e.g. [27], [28]) or discrete in time (e.g. [24], [29]), or whether the MPC should be linear [13] or non-linear [30]. The reasons behind selecting a specific approach before analysis is not always straightforward. For example, despite linearized approaches generally trading off computational speed for accuracy, [24] shows that a non-linear MPC could perform quicker computationally, while having the benefit of higher model accuracy over the linearized version. On a separate note, another major difference is whether the MPC is designed probabilistic (as done in [28], [31] or [32]), or deterministic (such as [33], [34], [18]). In a probabilistic MPC the design changes substantially, where the constraints become chance constraints and are defined by how likely they are fulfilled, using the standard deviation around the expected value. Avoiding the entire possible conflict region either leads to an overly conservative control action, or is simply infeasible. It should be noted that robust avoidance of obstacles must guarantee avoidance under the specified bounds, while probabilistic avoidance of obstacles has no such guarantees.

B. Robustness

The perception and characterization of obstacles can be uncertain due to sensor errors or linearization of non-linear obstacle motion. To account for this, a design robust to noise is used. There are several approaches, but robustness is always characterized by *bounded* noise, whereas *unbounded* noise is a probabilistic problem. Even when using linear noise bounds, both the trajectory (velocity noise) or simply the position can be influenced. The velocity noise would therefore have a time-dependent influence on the solution space. When using open-loop control (i.e. no state-feedback) and given a large enough prediction time, the accumulation of noise can cause the problem to be infeasible [20]. One approach is to limit the prediction window, with the main drawback of sub-optimal control actions. The other approach is to use closed-loop control, which can be more computationally expensive, but is mainly harder to implement because obstacle noise would need to be fed into the robot dynamics. Closed-loop control is therefore dependent on the definition of the robot and obstacle motion, making it hard or impossible to generalize.

One concern that must be addressed before deciding on the obstacle dynamics is understanding the probabilistic ap-

proach, and the consequences when using a simpler, non-linear approach. First, when using probabilistic control, the constraints are defined through probability distributions and likelihood of fulfilment [28]. Avoiding the entire possible conflict region either leads to an overly conservative control action, or an infeasible problem. Thus, rather than accepting a drastically limited solution space, motion planning is instead done either using probabilistic programming or through the nominal behavior. With probabilistic programming, deterministic constraints are re-defined into chance constraints, and the robot becomes probabilistically safe. The degree of safety is dependent on the likelihood of constraint satisfaction that the planner incorporates around the expected values of the solution. Although this approach is valid for urban Search and Rescue (uSaR), the intent of this work is robust planning, with the aim to guarantee safety.

Several approaches exist to make an MPC robust to noise, through either min-max control (which considers the worst-case disturbances), using stability theory (constraint tightening), or using closed loop control [20], [12]. Closed loop noise feedback is also known as tube MPC. Closed loop control attempts to reduce the cumulative effect of disturbances through feeding back the noise. In [12] the authors claim min-max MPC is the optimal solution to linear robust control, but that the high computational cost limits its use in vehicle navigation. Constraint tightening reduces the configuration space by expanding the constraints by a margin for each time step. In case the noise is time marched this could quickly cause the problem to grow infeasible, so usually it is combined with a feedback term to limit the reduction in solution space [12].

A tube MPC, on the other hand, instead reduces the cumulative effect of disturbances through feeding back the noise, ensuring the state eventually converges to the nominal state. Note that tube robust control is not limited to linear or linearized MPC approaches, see [35]. This form of active control fits the vehicle control problem, since active control is not as conservative in limiting the possible input envelope. A simplified way to look at tube robust control is (inspired from [36]):

1) First the nominal control input v is calculated.
2) The nominal system dynamics are used to relate the noise on the state to the input (using the state space $x^+ = Ax + Bu$).
3) A difference between the nominal and measured values (state, inputs, reference trajectory) is found $d = x_{\text{nom}} - x_{\text{meas}}$.
4) Pole placement is performed to find a nilpotent matrix K such that $\lim_{n\to+\infty}(A + BK)^n = 0$.
5) The pole placing matrix is combined with the noise and fed back into the nominal control input, resulting in the actual control input $\mu = v + Kd$.

C. Motion planning

According to [37], [9] and [11], motion planning considers a sequence of inputs that drives the initial state to a goal state. While path planning considers a time independent sequence, motion planning considers the trajectory. This is why different sources of knowledge are used to resolve different problems. There are several ways knowledge can be grouped for planning, but one approach is dividing it as global and local. While global knowledge would contain a possibly outdated map of the whole environment, local knowledge is the knowledge directly obtained from the sensors.

In time-varying environments, not using both types of knowledge can cause problems. On one hand, when relying on global knowledge, [1] noted an example at Fukushima, where rubble on an emergency staircase blocked a path otherwise thought feasible. Dogmatic following of outdated information could, following the logic of that example, bring more harm than good. On the other hand, a problem when using only local knowledge is most evident in an APF, where a robot can get stuck in a locally optimal solution, such as in Figure 2.

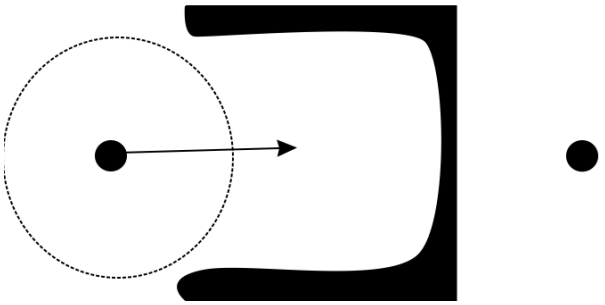


Fig. 2. Possible challenge for a purely local controller. The dashed circle denotes the sensor range, and thus local knowledge.

To avoid these, a combination of a local controller and a global controller is identified as the solution. This is not to say it is not possible to solely rely on local knowledge, as [38] identifies a specific sub-problem of target detection that does not use global knowledge as a *hunting* problem. Algorithms in this vein often rely on distributed control and communication or marking of the environment. Many other approaches to solve the motion planning problem are summarized and evaluated in more detail in the review papers such as [9], [39] and [11].

On a separate note, online performance is highly desired, and one way to achieve this is using linear programming. The problem is that the circular shape of the obstacles and the Euclidean L_2 norm used for the distance to obstacles are non-linear. To resolve this, literature suggests using distance approximations or free-space pre-computation.

- Distance approximations (inspired from computer vision) have been developed for mixed-integer linear programming which are a linear combination of the L_1- and L_∞-norms, see [40] for a comparison. If required, a generalized linear approximation is suggested in [41] based on a user-defined set of directions and desired accuracy.
- Pre-computation instead changes the problem by translating obstacle constraints into smaller, convex, obstacle-free regions (such as in [42] or [43]). Here a smaller solution space is traded-off for a decrease in computa-

tional complexity. Combining this with linearization, [13] limited the solution space through a set of iteratively defined linear constraints, such that only linear programming needs to be used in the optimization.

Unfortunately, as is the case in most other simplifications, there are either downsides that are not acceptable or there is a limited use to these approaches. The distance approximations are usually non-isotropic, and require several constraints rather than a single constraint per simulated obstacle, oftentimes resulting in mixed-integer programming. Though often still faster than non-linear programming, the reduction in solution space is a problem for a cluttered environment. Pre-computation, on the other hand, is guaranteed to reduce the computational time, especially in a cluttered environment. Though interestingly, it also has the largest drawback in a cluttered environment, because it works by limiting the solution space. Any optimal path is found within a sub-space of the free space, thus solving a smaller problem rather than the global problem. Due to the borders between the regions, the initial and terminal state per sub-space can influence the global solution negatively, yielding a sub-optimal global solution. Another, lesser downside, is that the sub-space computation is biased towards maximizing the area. The approach in [13] would therefore bias the path towards larger pockets, unless heavily guided by a global planner. These reasons are why the approach of pre-computing obstacle free regions is avoided, despite performing well computationally.

D. Robot and obstacle modelling

1) Robot model: The robot model is defined by the system dynamics, which in turn heavily depend on the type of robot used. Aerial drones have more degrees of freedom than ground robots, but also greater agility. However, because of flight-time limitations, ground robots are better suited for more extensive searches [44] and are therefore selected in this work. Robot dynamics can also be described in terms of the underlying locomotion system, or even how the actuation subsystem acts on the locomotion subsystem. Several different forms of locomotion exist for both aerial and terrestrial robots, each with their own benefits and drawbacks (see [45] for a limited study on UAVs, and [46] for terrestrial). Further still, it is even possible to define the dynamics as armature voltages supplied to the motors by the controller instead of the general linear and angular velocities often used in literature (see [47] for details). Prioritizing the development and design of a robust MPC, rather than a detailed optimized robot controller, an easy to implement non-linear model is preferred. For this reason, a circular 2-dimensional wheeled terrestrial robot using variable linear and angular speed is selected, as done in [18], referred to as the differential drive mobile robot [47].

Considering the possible complexity of modelling, approaches exist to simplify the robot model. Simplifications act on the concept level of the problem, calculations or assume certain value ranges. There are many concepts to interpret the motion and path planning problem, based on the dynamics and vehicle motion, as [9] identified 9 of them. The simplest form of these is the point-vehicle assumption. Under the point-vehicle assumption, the robot is transformed into point by finding the smallest possible radius that defines a ball that could cover the entire vehicle. Following, all obstacles are expanded with the covering radius, while the vehicle is "shrunk" to a point. The free space thus becomes the configuration space. To clarify, *free space* is the region unoccupied by obstacles while *configuration space* is the region where the robot could exist in and move through without touching any obstacles [37].

The problem is that the point-vehicle assumption does not account for any differential constraints due to vehicle dynamics. In other words, the limits of the dynamics (such as maximum acceleration) do not influence the configuration space. By adding these constraints the problem changes to a point-vehicle problem with differential constraints [9]. Subsequently a sensor range is included to ensure the controller is viable for real-life applications. At this point the problem is identified as a jogger's problem [9] in a time-varying environment. The metrics used to evaluate the performance of an algorithm for this problem are given in [9] to be the success rate (obstacle avoidance and reaching the goal), path length and mission time.

Two additional approaches to simplify the robot model are addressed here. If exploring the full range of states and inputs is a computational bottleneck, a library of maneuvers can be used instead, also known as motion automaton. Though specifically aimed to reduce the 3D complexity, using motion primitives (e.g. turn right, serpentine, U-turn) to supplement the controller means that the full range does not always need to be explored, reducing the computational burden (for a more in-depth discussion see [9]). The other approach is linearization. Though not a guarantee, as shown in [24], it is often used to speed up computation. In fact, linearizing a trigonometric term in the system model often does result in an improvement of computational speed. The main downside here is linearization error, as the non-linear dynamics do not perfectly translate to a linear form.

2) Obstacle model: Besides the robot model, the obstacles would also need to be defined. As described in [18], following the simplifications for the ad-hoc simulator, it is possible to model simple obstacles as circles, and more complex obstacle shapes as a combination of these circular obstacles. The choice to design obstacles as circles or groups of circles is a conceptual assumption, reducing the solution space since every obstacle is replaced by a circle or a set of circles covering it. Note that this approach does have a possible, though not very realistic, direct application. When using a point cloud ranging system such as Laser Imaging, Detection, And Ranging (LIDAR) each point-distance combination that is measured can be plotted as a circular obstacle with a radius as a function of its distance. This, however, is not very realistic since at the time of writing LIDARs have a downside that they are sensitive to weather conditions such as fog or rain [17].

Regardless, with the point vehicle modelling, any bounded uncertainties regarding obstacle position can be added as a radius on top of the obstacle or robot radius. There are some other assumptions for the obstacles in [18], one of them is that the robot correctly identifies the center of the obstacle. The drawback is that in reality, interpreting the environment

would require computation which is not done here, introducing an unspecified computational overhead. Besides, there is also an assumption on the placement of obstacles to ensure that the problem is feasible, which may not always hold true in time-varying environments, since moving obstacles may temporarily cause the problem to become infeasible. As such some parts need to be addressed in the redesign.

A dynamic environment could be as simple as static obstacles appearing/disappearing with respect to the initial map (causing the initial plan to be infeasible, or sub-optimal). However, a more interesting case is when obstacles move (addressed in [10] and [11]). Dynamic obstacle avoidance is a large field, but it assumes only one of a few types of motion. Different algorithms are identified for a *linear* obstacle dynamics compared to a fully unknown, *reactive*, or *non-linear* obstacles [12].

Linear obstacles are the simplest case, and move with a constant linear velocity. Many algorithms are designed around them as a result. A way to use these algorithms on other obstacles types is through successive linearization of the dynamics. Obstacles in uSaR, however, can take many different forms, even as victims or other robots limiting the configuration space. Victims or robot obstacles act on some conditions, changing their dynamics if those are fulfilled, making them reactive obstacles. *Reactive obstacles* are vital when testing for adversarial or multi-agent environments, but not necessary for the initial design of the motion planner. If needed, reactive motion can be approximated through a timed or conditional change in the obstacle dynamics, as done in [15]. *Non-linear obstacles* instead have non-linear dynamics, which may or may not be known to the motion planner. However, in reality, any prediction based solely on perceived data is a probabilistic problem. In other words, the estimated future position would be defined by a bounded probabilistic distribution. To account for this, some approaches are fully probabilistic, defining *probabilistic obstacles* where constraints become chance constraints and likelihood of success is used as a metric to define the best "possible" trajectory [28]. Though possibly of interest for future development, probabilistic obstacles are outside the scope of this work. In this case, the objective is to achieve dynamic obstacle avoidance, so *non-linear* obstacle dynamics are sufficient for the case study.

It should also be noted that errors are introduced by sources besides simplifying assumptions (such as linearization). In reality, a robot rarely knows its position with a 100% accuracy, due to sensor limitations, model inaccuracies, or changes in robot dynamics (resulting from damage, rugged environments, or lack of tuning). To represent these errors, a bounded uncorrelated additive noise (w) is added to the state, where the bounds are based on worst-case estimates. Note that this is a conservative estimate, as sensor noise is often correlated to the magnitude of the sensed value (larger speed means larger error), so a dynamic set of bounds would be less conservative, but still safe. However, keeping all different noise sources uncorrelated, and simply representing them as a set of scalar bounds, is a simplifying assumption that should be rectified when working with a real robot. Detailed noise characterization is considered to be outside the scope of this work (though [31] can be a good start, since it lists some approaches used for stochastic MPC). In this work, a robust approach to account for bounded noise is used, where the error bound on the predicted dynamics can be designed around.

E. Simulation of the dynamic environment

1) Sensors: Depending on what is developed (the controller, another subsystem, or some combination/interaction thereof), different types of fidelity are required for the simulation. While a highly realistic simulation is used to validate robot performance with different sensors and weathers before in-situ testing [17], in the early development stages a simple environment with only key features for a subsystem is needed. This is confirmed by other contemporary literature, since papers on autonomous vehicle path and motion planners usually use ad-hoc simulators when evaluating performance [16]. There are many implicit and explicit assumptions in ad-hoc simulators. When working with noise, ad-hoc simulators tend to assume the sensors and other subsystems have a bounded error despite the reality being highly complex. For instance, a glass obstacle may be hard to distinguish from free space with only a visual camera, while rustling plastic bags would influence measurements from a microphone [48]. Regardless, the purpose behind these simulations is often to prove some form of motion control, and deferring proper noise characterization and inaccuracies for another time or to the perception subsystem.

In this work the environment is kept as simple as possible, since a simulation that only tests the key features of dynamic obstacle avoidance is sufficient to test and develop the planners for time-varying environments. In the case that real sensors need to be used, additional computational overhead would come from algorithms such as sensor fusion, which are designed to improve the sensor subsystem (see [49], [50], [51] or [52] for details on sensor fusion). A simple simulation thus lowers computational overhead, which means that during hardware tests additional delays could be incurred, a consideration for future work.

2) Dynamic environment: Before identifying the key features, the environment that describes uSaR must be clarified. uSaR is usually performed in online and real-time dynamic environments, which means the controller must find a solution on-the-go while time-constrained. Various attempts are made to describe uSaR, though each approach differs in the details (such as degree and/or presence of fire spread, victim knowledge, etc.). Though a clear and strictly defined consensus is lacking, the general perception of a search task in uSaR encompasses static victims and dynamic environments, in a knowledge limited environment [5]. Victims are assumed static since any non-trapped and mobile victim would rescue themselves. Two key features can thus be identified in the form of the static goal or region, and dynamic environment.

According to [53], an environment is only *dynamic* if the predicted model of the environment can (and does) turn out to be wrong. This means that it is not sufficient to simply define the model of the moving obstacles. Instead, it must also be ensured that a lack of knowledge exists when solving

the motion planning problem, which would affect the future state prediction of the dynamic obstacles. From a conceptual standpoint, however, moving obstacles can come in the form of wildlife, hanging lamps, victims or other robots, so moving obstacles are included as a key feature.

Given enough time, there are several algorithms that can find the best possible trajectory, as explained in [9]. The problem is that in reality, there is a limited amount of time (and knowledge) to find the best motion plan, which in turn influences the best possible trajectory [54]. To better evaluate the motion planners, time-marched simulations are expected to equalize the conditions for obstacle avoidance regarding computational time. If an online controller takes too long to update, a change in the environment may not be accounted for before the next update step, causing a collision or a substantial inefficiency in the planning, where another planner might improve. As such, the final key feature in the simple simulator is the online and time-marched condition.

It should be clarified that the time-marching condition is not always strictly adhered to throughout motion planning literature. Technically, this refers to a system that enforces strict deadlines, and updating a process despite a possible lack of up to date inputs. This is not often used throughout literature (see for example [55], [54], [56], [15] and [19]). Instead, hardware tests are done soon after and tend to validate the results (for example, [24]).

As a final consideration for the simulator, there are many other aspects which are not considered key factors, despite being key in some other simulations. Details such as fire spread (such as [57], [58]) and the predicted motion of humans (see [59], [60]) are vitally important in most scenarios, but these are not accounted for here. The main reason is that this defines the global planning, which sets the goals based on a variety of criteria, while the local planner simply needs to avoid danger or obstacles. Designing an autonomous system that addresses a mission optimal trajectory is deferred to future work and recommendations.

To conclude, circular obstacles, where some follow a curved trajectory with accelerating and decelerating velocities, are placed in the environment of the robot. These either temporarily block the path, serve as background noise (increasing information that needs to be interpreted), or directly threaten a collision unless avoided. Uncertainties are included in obstacle positions and motions, the extent based on the number of simplifying assumptions required for computational efficiency. This has to be addressed by a controller design robust to this noise, and a planner that supports it.

III. Problem definition

To address the research objectives, the details of the design and influence of other subsystems on the motion planner and trajectory tracker is reduced. Thus, the design is limited to autonomous control, considering the possibly communication denied environments [1]. The uSaR search task encompasses nonlinearly moving obstacles, where a global goal or sequence of goals is given to the heuristic motion planner. A dynamic obstacle avoiding motion planner and trajectory tracker are designed for a differential drive mobile ground robot. The control algorithm is made deterministic to ensure the collision avoidance is robust to additively bounded noise on the positions of both the robot and the dynamic obstacles. The aim is to create a motion planner that reaches a specified goal using local knowledge, avoiding non-linearly moving dynamic obstacles in the process. This is done in a time-marched simulated environment. Both the dynamic obstacles and robot position are affected by a bounded uniformly random generated noise.

The control approach is limited to a local controller, and any design considerations that use global knowledge, such as fire spread or human motion modelling, are deferred to future work. Any detailed human-to-robot or robot-to-robot interaction is ignored at this stage. The communication subsystem can technically be added as an afterthought (with additional tuning), by means of a constraint or a goal with respect to the motion planner. Using the same logic, supervisory control can be implemented by using a goal that can be redefined throughout the search process. There are limits to this approach, mostly because there will be some design debt due to a lack of exploiting possible synergies, such as decentralized localization, among others [44]. Regarding the simulation fidelity and case study, a simplified sensor model is used, where the robot can detect everything within a radius. Detailed design of the perception subsystem, however, cannot be done at this stage of the development.

A. Model dynamics

The development of the controller is one of the main purposes of this work, meaning the design problem is defined through the models and controller design. To this end, the robot and obstacle dynamics used in the simulation, heuristic motion planner and trajectory tracking MPC are discussed.

1) Robot model: The model used in the MPC is the differential drive mobile robot [47]. The equations of motion are using the explicit Euler method of integration are shown in (1). Here the state (2a) is defined using the x and y position, as well as the angle of rotation θ with respect to the inertial frame. The input (2b) consists of the variables that are linear (v) and angular velocity (ω).

$$\Delta x = \Delta t_{\text{ctrl}} \left(v_0 \cos\theta_0 + \Delta v \cos\theta_0 - \Delta\theta v_0 \sin\theta_0 \right) \quad (1a)$$

$$\Delta y = \Delta t_{\text{ctrl}} \left(v_0 \sin\theta_0 + \Delta v \sin\theta_0 + \Delta\theta v_0 \cos\theta_0 \right) \quad (1b)$$

$$\Delta\theta = \Delta t_{\text{ctrl}} (\omega_0 + \Delta\omega) \quad (1c)$$

$$\mathbf{x}_{\text{rob}} = [x, y, \theta]^T \quad (2a)$$

$$\mathbf{u}_{\text{rob}} = [v, \omega]^T \quad (2b)$$

While the equations shown here are integrated over a control time step using the Euler method, it is also possible to use Runge-Kutta or other methods. The equilibrium point (around which is linearized) is subscribed with "0", meaning (v_0, ω_0) are the inputs at the linearized point. The robot model is required and used by the simulation and the trajectory tracking MPC. The heuristic planner instead has no such constraints, as obstacles have a sufficient radius to ensure a turn is feasible. The MPC uses the linearized set of equations shown in (1), while the simulation uses the non-linear form.

2) Obstacle model: The uSaR environment is dynamic, with dynamic obstacles possibly taking the shape of rescuers, animals or other robots. In order to ensure it is a dynamic environment as defined in [53], just having moving obstacles is insufficient. Instead, the internal model used by the robot to predict the obstacle motion must possibly turn out to be wrong. One way is to obtain model mismatch solely through additive noise, assuming knowledge the full set of nominal obstacle equations of motion as shown in (3).

$$\dot{x} = f\left(v_x, \Delta t_{\text{ctrl}}\right) \tag{3a}$$
$$\dot{y} = f\left(v_y, \Delta t_{\text{ctrl}}\right) \tag{3b}$$
$$\dot{v_x} = f\left(\frac{(x_{\text{att}} - x)}{x_1} a_x, \Delta t_{\text{ctrl}}\right) \tag{3c}$$
$$\dot{v_y} = f\left(\frac{(y_{\text{att}} - y)}{y_1} a_y, \Delta t_{\text{ctrl}}\right) \tag{3d}$$
$$\mathbf{x}_{\text{obs}} = [x, y, v_x, v_y]^T \tag{4a}$$
$$\mathbf{u}_{\text{obs}} = [a_x, a_y, x_{\text{att}}, y_{\text{att}}]^T \tag{4b}$$

Where "$f(\cdot, \Delta t_{\text{ctrl}})$" represents Runge Kutta 3/8 rule, used to integrate a term over the control timestep Δt_{ctrl}. The obstacle dynamics are specifically designed to ensure that their motion continuously obstructs the optimal trajectory. While the Point of Attraction (PoA) $(x_{\text{att}}, y_{\text{att}})$ and acceleration terms (a_x, a_y) do not vary over time, the position (x, y) and velocity terms (v_x, v_y) do. The variables x_1 and y_1 can serve as values to ensure the fraction remains between -1 and 1. Using these kinematics, the dynamic obstacles would oscillate or circle around the point of attraction, ensuring that the robot cannot simply wait for the dynamic scenarios to be resolved, see Figure 3.

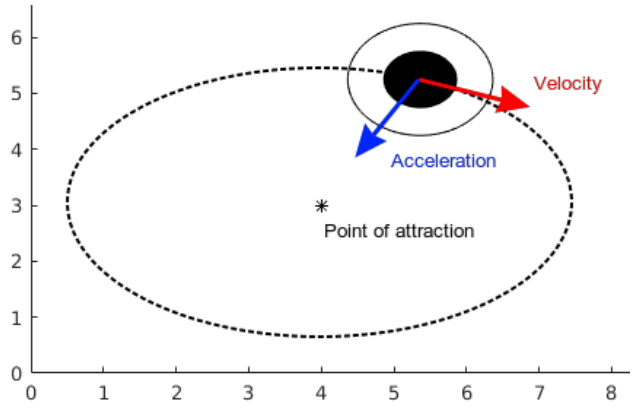


Fig. 3. Moving obstacle dynamics using the variables from (3). The dashed line is the ideal trajectory. The black circle is the obstacle itself, while the empty circle surrounding it is the expanded radius form following the point-vehicle assumption.

In reality, full knowledge of the dynamic obstacle motion, (3), cannot be assumed. The heuristic planner assumes the obstacles have a constant linear and angular velocity (circular motion) between update steps, meaning they follow (1) with the difference that v and ω are constant and that the Runge-Kutta 3/8 rule is used instead of the explicit Euler method.

In the MPC, the obstacle motion prediction is instead assumed completely linear, limiting the assumed knowledge of the obstacle dynamics. The linearization error can also be added to the additive noise (similar to [25]). These velocities are sampled every timestep, see (5) for the obstacle model used in the MPC, using v_x and v_y as the velocities sampled in the x and y direction, respectively. The MPC uses the explicit Euler integration to predict the future position of the obstacles.

$$\Delta x = v_x \Delta t_{\text{ctrl}} \tag{5a}$$
$$\Delta y = v_y \Delta t_{\text{ctrl}} \tag{5b}$$
$$\mathbf{x}_{\text{obs}} = [x, y]^T \tag{6a}$$
$$\mathbf{u}_{\text{obs}} = [v_x, v_y]^T \tag{6b}$$

B. Control system design

The control system gives a clearer view of what is being developed and how subsystems could interact with the motion planner. For now, it should only be noted that it is hard to guarantee that the robot will always reach the goal, and specific circumstances that must be identified may influence this. The controller has thus been designed by feeding the outputs of the motion planner into the motion tracking MPC, while the MPC can request a re-planning when needed, as illustrated in Figure 4.

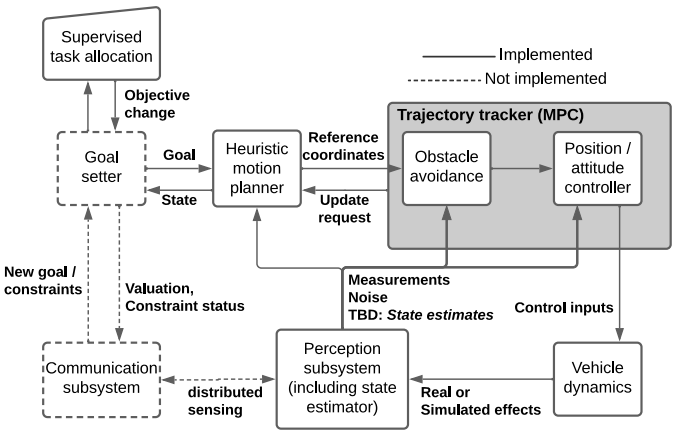


Fig. 4. Generalized controller design.

To improve the computational time and responsiveness it is opted to separate the motion planning into a planner and reference tracker. This is inspired from [61], [30] and others where computational performance is desired. In each case, a hierarchical design is used where the motion plan is optimized separately due to the size of the problem, while leaving subsequent control actions to a responsive trajectory tracking algorithm. As can be seen from the design, the obstacle modelling is done separately.

C. Heuristic path planner redesign

In the end, inspired by the simplicity of the ad-hoc simulator and jogger's problem representation, an equally simple approach is taken as a baseline. Here the shortest-path generating algorithm from Jamshidnejad and Frazzoli's work [18] was chosen, but there are two problems. One drawback of this approach is that it relies on local knowledge and the problem in Figure 2 could persist. The other drawback is that it does not consider dynamic obstacles.

1) Original algortihm: In this algorithm, a greedy heuristic approach is taken, where circular obstacles and obstacle belts are avoided by charting a path using tangential lines and arcs. The discussion in this section does not concern the detailed description of the original shortest path algorithm from [18]. Instead the focus here is on the changes that account for dynamic obstacle avoidance and temporary goal selection.

To summarize, the path generation is purely a minimum distance optimization, which is fed into the MPC. It works by drawing a line between the position of the robot at the time of planning and a temporary goal, forming upper and lower tangent lines around interfering obstacles, and selecting the shorter of the two. This is repeated until a collision-free path is found between the two points (see Figure 5). A random path is chosen in case the two paths have the same length.

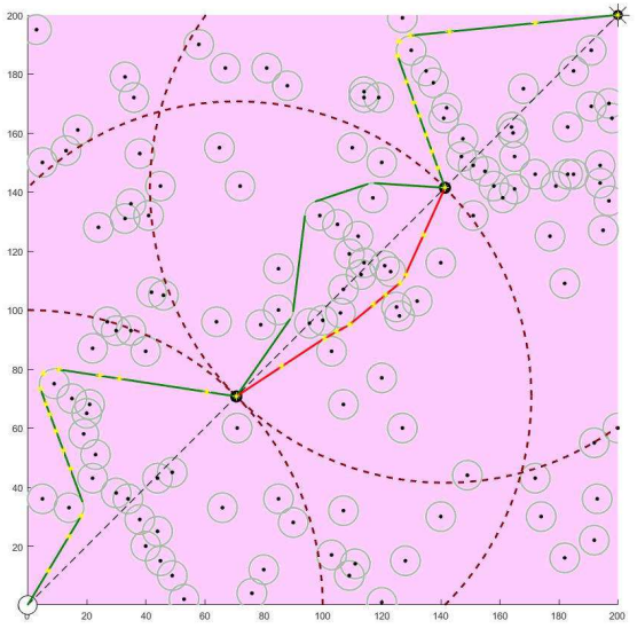


Fig. 5. Shortest paths found using the heuristic path planning algorithm, selected path is green-red-green. Image taken from [18].

Coordinates (yellow stars in Figure 5) are selected along the shortest path and provide a reference trajectory for the local controller. This way the local controller avoids the possible trap shown in Figure 2 to the extent defined by the sensor range. The way coordinates are selected can be "equidistant points on the shortest path trajectory" as one of the suggestions by the authors. This method has thus also been reproduced in a similar environment, see Figure 6, before moving on to modifications.

2) Self-intersection clarification: Before the modifications to the algorithm are elaborated, one point should be clarified. When the heuristic algorithm draws tangent lines it is possible that (during the conflict-resolution iterations), the path becomes self-intersecting. The moment a self-intersection is detected on a path (at the end of step 5 in [18]), the two tangent lines connecting to the "inner" obstacle are deleted, and a new tangent line is drawn between the "outer" obstacles.

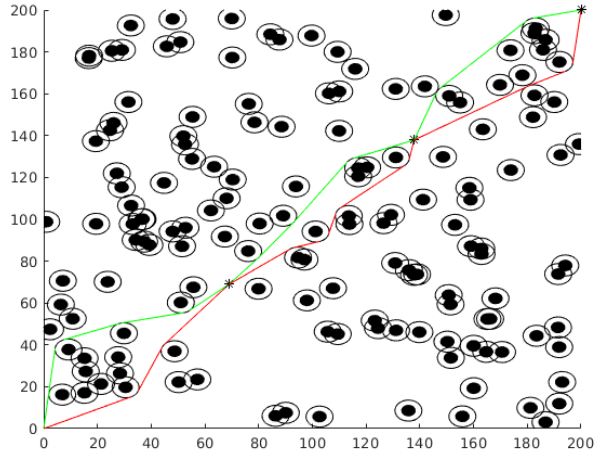


Fig. 6. Red and green lines following the implementation of [18] in a static environment. Stars are temporary goals, and also where re-planning occurs.

For visualization, see Figure 7 where the path changes from the green line to the black line.

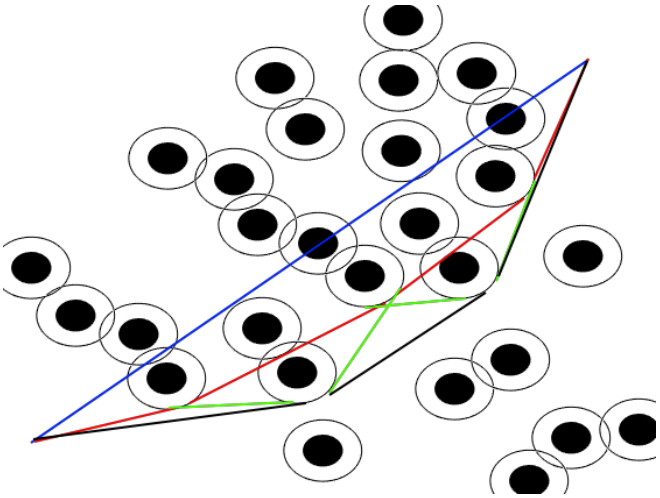


Fig. 7. Path smoothing in case of self-intersection, showing the start-goal line (blue), first iteration (red), final iteration (green), and subsequent smoothing (black).

3) Unreachable goals modification: The first modification concerns unreachable goals. The algorithm in [18] assumes no knowledge besides that which falls within the detected zone (see the light blue area in Figure 8). In case a temporary goal falls in a hard-to-reach, or impossible to reach area, the original algorithm would struggle to create a feasible path. Instead, the furthest feasible point that can be reached following the "shortest" red or green path is selected as a temporary goal, in case the temporary goal cannot be reached (in Figure 8).

4) Dynamic obstacle modification: The second set of modifications are done to make the algorithm compatible with dynamic obstacles. To this end, a dynamic obstacle is translated into a set of static obstacles using their predicted dynamics, see (1). First the case of perfect knowledge is addressed, and subsequently the case of limited knowledge and noise.

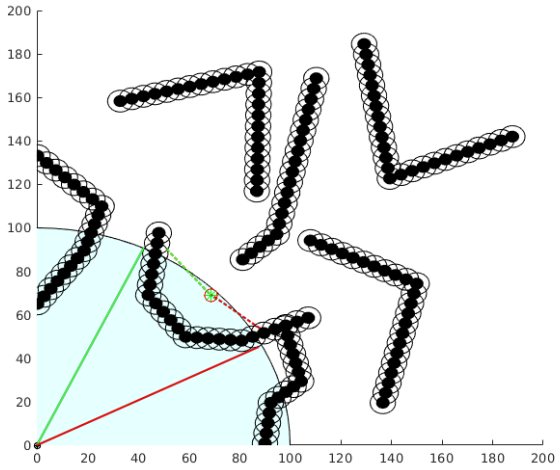


Fig. 8. Setting of a new temporary goal (the end-point of either the red or green line). Only considered if the original temporary goal cannot be reached (see dashed lines that go past the light-blue detected zone).

1) The static obstacle avoiding red and green paths are generated, after which points are sampled on the paths per time step (from t_1 to t_{end}).
2) For each time step the robot position is tested with the obstacle positions of the past (t_{c-1}), current (t_c) and future (t_{c+1}) time steps.
3) When a collision is predicted on a path, the dynamic obstacle causing the collision is treated as an obstacle belt, and a new collision-free path is generated.
4) Repeat steps 1-3 until the shortest path trajectory is found

A downside of generating the belt from t_{c-1} to t_{c+1} is that it requires several iterations if the robot moves in front of a moving obstacle. To avoid this, the solution is biased away from future time steps, by defining the obstacle belt from t_{c-1} to t_{c+3}, see Figure 9.

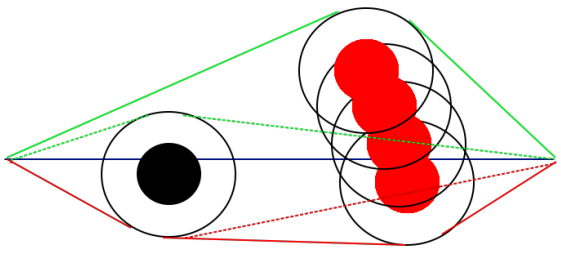


Fig. 9. Dashed line shows original trajectory, full lines show new trajectories around the dynamic obstacle (red). These trajectories are tested again for collision-avoidance, and the shorter one will be chosen.

5) Obstacle noise: Though this works in scenarios with perfect knowledge, a final set of modifications needs to be made for the heuristic planner to be robust to noise. In case of noise around obstacles, a common method to make the planner robust is to expand the obstacle radius by the bounded value $w_{\max}$ (increasing the radius of the hollow circles in Figure 6). This bounded value is defined by both the state noise from the robot and the possible perception noise. When considering time-marched noise, the bounds grow as time is increased. This mostly depends on the degree of inaccuracy in modelling the obstacles. To re-iterate, the heuristic planner assumes constant angular and linear velocity of obstacles.

Unfortunately, in reality, motion prediction cannot be assumed to be fully accurate, and noise accumulation is a problem. Generally though, the predicted obstacle positions will be more accurate the closer in the future it is (less time steps to march the possible error). But even then, sudden shifts in obstacle dynamics or other unexpected events could cause sufficiently large offsets that invalidate the shortest path trajectory. To avoid a large influence of noise to the heuristic motion planner either a frequent update step can be employed, or a closed-loop noise feedback could somehow be implemented.

Regardless of the approach taken to address this, with no certain knowledge of the environment, a long-term prediction leads to either excessive control actions or only a probabilistically safe control method. At this point it is important to realize that the benefit of a bi-level controller is to delegate these conceptual "obstacles" to the reference tracking MPC. If the local controller notices a significant (user-defined) deviation, it can request the heuristic planner to re-evaluate the reference trajectory based on newer information. Meanwhile, the MPC must keep the robot safe.

Therefore, the ideal path is kept as a guideline, with the initial time steps weighed heavier than the final ones, such that other objectives in the MPC help guide the motion planner. In the ideal case, the assumptions are that the obstacles have constant linear speed, and constant angular velocity. With these assumptions in mind, the path-planner needs to be tested.

D. Model predictive controller design

MPC was identified as a possible control scheme for the reference tracker, since it can track a reference signal based on certain heuristics and subject to constraints while robust to a bounded noise [20]. Additionally, [18] has a MPC motion planner combined with their path planner. Unfortunately, their approach does not account for noise and is limited to static obstacles, which is why it is redesigned. In their case, a dynamic obstacle could temporarily block the goal, leading to a possible conflict between the obstacle avoidance and goal reaching constraints. Another reason for starting with a new MPC is that it makes testing and development easier. Thus, even though [18] is used as inspiration, a new MPC is developed for the motion planner. In order to do so, the nominal (noiseless) case must first be expressed in terms of the system model (1), dynamic obstacle model (5), integration method, constraints (10) and objective function (9). Finally the short-term planner is made robust to noise by using a tube robust approach on the noise acting on the state and obstacle positions.

1) Method of integration: The integration method for both models is chosen to be Euler's explicit integration, primarily because the system (1) and dynamic obstacle models (5) used by the MPC are already linear. Initial testing also suggested

that using higher order integration was counterproductive for computationally limited control. As shown in Figure 1 a prediction H_p and control input horizon H_c are used. Because of the linear system model in (1), any future states have a linear relationship with any previous predicted states and inputs. Thus the first predicted state $\mathbf{x}(\kappa+1)$ is defined by the initial state $\mathbf{x}(\kappa)$ and the first input $\mathbf{u}(\kappa)$, and subsequent states follow the same logic. The linear relationship can then be translated to matrix notation, as a linear combination of terms translates into the future state, see (8). For brevity, $\underline{\mathbf{x}}(1, H_p)$ is a vector containing the states from $\mathbf{x}(\kappa+1)$ to $\mathbf{x}(\kappa+H_p)$ and $\underline{\mathbf{u}}(0, H_c)$ is a vector of the inputs from $\mathbf{u}(\kappa)$ to $\mathbf{u}(\kappa+H_c)$, assuming $\mathbf{u}(\kappa+i) = \mathbf{u}(\kappa+H_c)$ between $H_c \leq i < H_p$.

$$\mathbf{x}(\kappa+1) = A(\kappa)\mathbf{x}(\kappa) + B(\kappa)\mathbf{u}(\kappa) \tag{7a}$$

$$\begin{aligned}\mathbf{x}(\kappa+2) = A(\kappa)\left(A(\kappa)\mathbf{x}(\kappa) + B(\kappa)\mathbf{u}(\kappa)\right) \\ +B(\kappa)\mathbf{u}(\kappa+1)\end{aligned} \tag{7b}$$

$$\underline{\mathbf{x}}(1, H_p) = A_{\text{big}}(\kappa)\mathbf{x}(\kappa) + B_{\text{big}}(\kappa)\underline{\mathbf{u}}(0, H_c) \tag{8}$$

Following state-space notation, the future state is described through the state $A(\kappa)$ and input matrix $B(\kappa)$ linearized at timestep κ. These are re-linearized every timestep. Future states can be written in matrix format using $A_{\text{big}}(\kappa) = [A(\kappa), A(\kappa)^2, \ldots, A(\kappa)^{H_p}]^T$, where $A \in R^{n\times n}$, with n the number of states. $B_{\text{big}}(\kappa)$ follows a similar structure resulting in a matrix sized $nH_p \times mH_c$, with $B(\kappa) \in R^{n\times m}$, where m is the number of inputs. The last rows in the $B_{\text{big}}(\kappa)$ matrix are $[A(\kappa)^{H_p-1}B(\kappa), A(\kappa)^{H_p-2}B(\kappa), \ldots, \sum_{n=0}^{H_p-H_c} A(\kappa)^n B(\kappa)]$.

2) Objective function: For its objective function in (9), something simple was chosen since the MPC must repeatedly minimize it until the goal is reached or deemed unreachable by the heuristic planner. In part to streamline development, the objective function here only uses the offset from the reference trajectory and a term used to represent the kinetic energy, weighed by the vectors $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ respectively. The first term is used to minimize the path length by following the trajectory from the heuristic planner. The second term reduces the speed of the robot, intended to improve energy use and collision avoidance by reducing the kinetic energy. Following the jogger's problem definition in [9], a lower speed would assist the obstacle avoidance.

$$\min_{\mathbf{u}} \left(\boldsymbol{\alpha}_1 \left(\underline{\mathbf{x}}(1, H_p) - \mathbf{x}_{\text{ref}}\right)^2 + \boldsymbol{\alpha}_2 \left(\underline{\mathbf{u}}(0, H_c)\right)^2\right) \tag{9}$$

Where $\underline{\mathbf{x}}(1, H_p)$ means all timesteps from the first predicted state $t = \kappa+1$ to the last predicted state $t = \kappa+H_p$ are used. Also, the values of the reference trajectory $\mathbf{x}_{\text{ref}}$ are based on the closest point to the current position of the robot $\mathbf{x}_{\text{rob}}$. This means the reference trajectory is consistently shifted forward, unless the robot reverses course (for example during obstacle avoidance maneuvers), as constraints take priority to achieving a lower objective value. Lastly, note that while $\boldsymbol{\alpha}_2$ is constant, $\boldsymbol{\alpha}_1$ is biased towards the earlier timesteps. This is because the reference path coordinates in the far future are less likely to be optimal than the earlier ones.

3) Nominal MPC design: Despite the linearized system dynamics, the MPC is still nonlinear because of the Euclidean norm term present in (9) and some constraints. From the constraints the zoning (10e), static obstacle (10f) and dynamic obstacle (10g) avoidance constraints, use the Euclidean norm making them non-linear. The state (10b), input (10c) and smoothness (10d) constraints on the other hand are linear and expanded to include the future states.

$$\begin{aligned}\underline{\mathbf{x}}_{\text{rob}}(1, H_p) = f(A_{\text{big}}(\kappa), B_{\text{big}}(\kappa), \mathbf{x}_{\text{rob}}(\kappa), \\ \underline{\mathbf{u}}_{\text{rob}}(0, H_c), \Delta t_{\text{ctrl}})\end{aligned} \tag{10a}$$

$$x_{\text{min}} \leq \underline{\mathbf{x}}_{\text{rob}}(1, H_p) \leq x_{\text{max}} \tag{10b}$$

$$u_{\text{min}} \leq \underline{\mathbf{u}}_{\text{rob}}(0, H_c) \leq u_{\text{max}} \tag{10c}$$

$$|\underline{\mathbf{u}}_{\text{rob}}(-1, H_c - 1) - \underline{\mathbf{u}}_{\text{rob}}(0, H_c)| \leq u_{\text{smooth}} \tag{10d}$$

$$\|\mathbf{x}_{\text{rob}}(i) - x_0\| \leq \rho_{\text{det}} - \rho_{\text{safe}}, \quad \forall i = \kappa+1, \ldots, \kappa+H_p \tag{10e}$$

$$\|\mathbf{x}_{\text{rob}}(i) - x_{\text{obs, static}}\| \geq \rho_{\text{obs}} + \rho_{\text{rob}} + \rho_{\text{w, state}}(i-1) \tag{10f}$$

$$\begin{aligned}\|\mathbf{x}_{\text{rob}}(i) - \mathbf{x}_{\text{obs, dyn}}(i-1)\| \geq \rho_{\text{obs}} + \rho_{\text{rob}} + \rho_{\text{w, dyn}}(i-1) \\ \|\mathbf{x}_{\text{rob}}(i) - \mathbf{x}_{\text{obs, dyn}}(i)\| \geq \rho_{\text{obs}} + \rho_{\text{rob}} + \rho_{\text{w, dyn}}(i-1) \\ \|\mathbf{x}_{\text{rob}}(i) - \mathbf{x}_{\text{obs, dyn}}(i+1)\| \geq \rho_{\text{obs}} + \rho_{\text{rob}} + \rho_{\text{w, dyn}}(i-1)\end{aligned} \tag{10g}$$

When $\kappa + H_c \leq i < \kappa + H_p$, then $\mathbf{u}_{\text{rob}}(i) = \mathbf{u}_{\text{rob}}(\kappa+H_c)$, following Figure 1. The robot state and control input vectors, $\mathbf{x}_{\text{rob}}$ and $\mathbf{u}_{\text{rob}}$ respectively, are defined using the linearized differential drive robot dynamics in (1) and the method of integration. They are enforced through the system dynamics constraints in (10a). Other constraints can enforce limitations of the vehicle, models, or the environment (in the form of obstacles).

Constraint (10b) is a state constraint that limits the maximum and minimum values of the robot x, y and θ, and works as a bound for the simulation. Constraint (10c) is a control input constraint that limits the maximum and minimum values of the robot v and ω, while constraint (10d) limits the rate of change thereof, representing mechanical or dynamic limits of the robot. Here $\mathbf{u}(\kappa-1)$ represents the past control input. Constraint (10e) forces the current and predicted robot states to stay within a detected zone defined by the planning radius ρ_{det}. A safety radius ρ_{safe} is subtracted from ρ_{det} to account for possible obstacles just outside of the detected zone, which could cause conflicts. Constraint (10f) constrain the robot away from the detected static obstacles based on a sum of the obstacle ρ_{obs}, robot ρ_{rob}, and state noise bound $\rho_{\text{w, state}}$. The dynamic obstacle avoidance constraints in (10g) instead use dynamic noise $\rho_{\text{w, dyn}}$ radius that includes the noise acting on the dynamic obstacle. The state and dynamic obstacle noise vary over time, because tube robust control will lead to a limited accumulated error over time [20]. The dynamic obstacle state $\mathbf{x}_{\text{obs, dyn}}$ is predicted using the model in (5). So the obstacle avoidance constraints use the predicted positions, not velocities or orientations. At this point, if the reference trajectory is considered infeasible by the MPC despite a (user-defined) margin, it calls back to the heuristic planner to find a new reference trajectory.

4) Noise robust design: To reduce the accumulation of linearization error, every timestep the dynamics are re-linearized. Also, the smaller the timestep used, the smaller the error and more responsive the MPC becomes. Despite these changes, the nominal MPC still has sizable inaccuracies and is also not robust to noise. To this end, the linear MPC approach is further developed and made robust to noise. There are several ways to do this, based on how the noise is characterized and

the desired results [20]. First, the noise is assumed to affect the robot state and the dynamic obstacle position. Second, the noise is assumed to be a bounded uncorrelated additive noise, meaning no chaos or mixture models need to be used. Therefore, third, a tube robust approach is used to account for this noise. Adding a bounded uncorrelated additive noise (w) to the state equations, results in (11).

$$\begin{aligned}\Delta x &= f(v_0\cos\theta_0 + \Delta v\cos\theta_0 - \Delta\theta v_0\sin\theta_0, \\ &\quad \Delta t_{\text{ctrl}}) + w_x \end{aligned} \tag{11a}$$

$$\begin{aligned}\Delta y &= f(v_0\sin\theta_0 + \Delta v\sin\theta_0 + \Delta\theta v_0\cos\theta_0, \\ &\quad \Delta t_{\text{ctrl}}) + w_y \end{aligned} \tag{11b}$$

$$\Delta\theta = f(\omega_0 + \Delta\omega, \Delta t_{\text{ctrl}}) + w_\theta \tag{11c}$$

To use tube robust MPC, the nominal response is limited by looking at the possible results of the control inputs. Ensuring that the noise can be accounted for by a feedback term that uses up a margin of the control input range, the remainder of the input can be used to control the robot. With the state noise feedback, the state and input matrices A and B of the system dynamics are used. Though this is straightforward for the state noise, the problem is that the dynamic obstacle noise affects the state indirectly through the future states. Thus to address that, a method is used that combines constraint tightening with feedback control.

5) Pole placement and feedback: Although the noise is preferred to fade away as soon as possible, setting the poles at 0 can be overly conservative in reducing the range of control inputs. The pole placement for the state noise is done using $A(\kappa)$ and $B(\kappa)$, placing the poles at [0.5,0.5,1], obtaining a feedback matrix $K_{\text{state}}(\kappa)$. For the dynamic obstacle the poles are placed at [0.7,0.7,1], splitting 30% of the noise towards noise feedback, and 70% towards constraint tightening, resulting in feedback matrix $K_{\text{dyn}}(\kappa)$. This leads to a maximum possible bound of 2 times for the state noise, and 3.333 times the dynamic obstacle noise. The maximum possible noise values are found using the geometric sequence in (12).

$$\rho_{\text{w, state}}(\kappa+n) = w_{\max,\text{state}}\sum_{i=0}^{n}(1-\xi_{\mathbf{x}})^i \tag{12a}$$

$$\rho_{\text{dyn}}(\kappa+n) = w_{\max,\text{dyn}}\sum_{i=0}^{n}(1-\xi_{\text{dyn}})^i \tag{12b}$$

Where ξ_x is the damping value for the state noise (50%), ξ_{dyn} damping value for dynamic obstacle noise (30%), and n is the number of future time steps, ranging between 1 and H_p. The noise bounds are defined by $w_{\max,\text{dyn}}$ for the dynamic obstacle noise bound, and $w_{\max,\text{state}}$ for the state noise bound. Constraint (10g) refers to $\rho_{w,\text{dyn}}$, which in this case is simply a sum of the constraint tightening due to the dynamic obstacle noise, ρ_{dyn}, and state noise, $\rho_{w,\text{state}}$. Based on these values constraint tightening can be done to limit the control inputs.

6) Constraint tightening: For the state noise, the state constraint is tightened by $[1, 1.5, \ldots, \sum_{n=0}^{H_p-1} 0.5^n]$ times the maximum state noise for each time step. For the dynamic obstacles, the safety radius is expanded by $[1, 1.7, \ldots, \sum_{n=0}^{H_p-1} 0.7^n]$ times their maximum noise bound for each of the n future time steps, respectively. With regards to dynamic obstacles, the future obstacle positions would therefore be increasing in radius up to their maximum value, as shown in Figure 10.

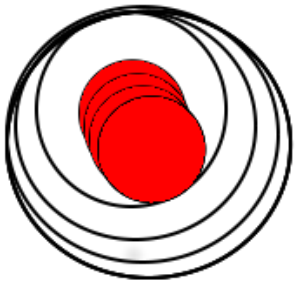


Fig. 10. Dynamic obstacle with predicted future states. The further in the future it is, the more the dynamic noise bound expands, limiting the configuration space.

7) Feedback control: The state noise feedback determines the control input μ_1 using the nominal control input u, where $\mu_1(\kappa+i) = \underline{\mathbf{u}}(\kappa+i) + K_{\text{state}}(\kappa)w_{\text{state}}$, for all $i = 0, 1, ..., H_c$. With regards to the dynamic obstacle noise, a limit has been put in place, so that only obstacles that are on the trajectory predicted by the MPC are accounted for. The noise shifts the obstacle away from its nominal trajectory, so the feedback must drive the robot towards the new position of the obstacle. This results in a sign change, such that $\mu_2(\kappa+i) = \mu_1(\kappa+i) - K_{\text{dyn}}(\kappa)w_{\text{dyn}}$, for all $i = 0, 1, ..., H_c$.

However, if two obstacles with perfectly opposite noise spectra would intersect the trajectory at different time points the robot would not change from its nominal trajectory and simply risk collision. To account for this case, the trajectory tracking MPC can request a re-planning from the heuristic motion planner for a new reference trajectory. However, since that can take several time steps, the immediate action is to only perform noise feedback on the first dynamic obstacle. With this logic in mind, the shift in dynamic obstacle position is directly translated to a shift in predicted robot positions. Ideally, the further in the future the dynamic obstacle is, the more spread out the noise correction is. This would then only lead to a minor course correction in the reference trajectory, thereby incorporating the long distance noise feedback into the reference trajectory motion planner, while ensuring the short-term reference tracker is locally robust.

IV. Comparison Algorithms

For a comparison of performance, two state-of-the-art approaches were taken, based on different algorithms that solve the same dynamic obstacle avoiding problem. The first method is Horizon-based Lazy RRT (HL-RRT)* based on the sampling approach RRT*. The second method is an APF approach developed for naval collision avoidance. Both papers are modified versions of their original algorithms, developed for a dynamic environment. Following is a summary on their working with some comments regarding their presentation and implementation.

A. Sampling-based: HL-RRT*

There are many possible types of sampling methods for the purposes of motion planning, from probabilistic roadmaps (PRM) to RRT, see [62]. Since the asymptotically optimal versions of PRM (PRM*), and RRT (RRT*) published in [54], further improvements based on these are available, such as kinodynamic RRT or RRTx. From the methods designed for dynamic environments, a recent work that seems similar to the MPC is HL-RRT* proposed in [19].

There are two main differences between HL-RRT* and RRT*. The first is the *lazy* collision checking, meaning the HL-RRT* only tests branches within a collision-checking horizon. While following the optimal path found by the algorithm regular updates occur, which shift the horizon forward. In these events, the collision checking horizon is re-centered on the current position of the robot and a new tree is generated. The second difference is a learned sampling approach for future growth of the tree.

The original RRT* samples the free space uniformly, and finds a nearby vertex to "grow" a new branch from. The sampled point might not be reached, since the steering function might be constrained, so instead the closest reachable point is used for the branch. Any nearby nodes are then tested if they can connect to the new branch at a lower cost, and then the tree gets rewired. On the other hand, the HL-RRT* samples the free space, goal, or a learned set randomly, taken from [63]. The learned set is defined by a robust expectation-maximization algorithm, taken from [64]. It then grows a branch by steering to the new node the same way as RRT*.

A possible error could be present in their implementation, however unlikely. One thing to note is that one of the variables does not follow the definition as given in [54] or [53]. In RRT*, η is defined as a tunable parameter set as the *supremum* of the segment length between two points in the free space, reachable by the steering function and subject to no obstacle intersections. On the other hand, [19] defines η as a tunable parameter often set as the *infimum* of a reachable set breadth. This variable is used to limit the steering function, and serves as the "radius" in the parenting and rewiring sub-processes. Based on the metric function it might not be a radius, but regardless of that, overestimating it makes the RRT* result in a lower path cost at the expense of slower calculations (negligible when considering the performance of the HL-RRT*). Also, several follow-up equations (such as: $r \geq 4\eta$) are thus different compared to the other RRT implementations (where $r = \min([...], \eta)$).

Only when everything can be justified by connecting the underlying theory or results, can the implementation be seen as correct. Regarding the other sub-functions or variables these are either newly defined in [19] or a reference is provided to another paper. As a result, whenever a reference is made to another work, all variables, equations, and algorithms are double-checked with regards to their definitions and use. In case of a possible conflict such as with η (note that this is also the only possible misinterpretation with a substantial effect on the performance), the basic RRT* algorithm takes precedence over the unclear HL-RRT*. This decision is made because: "Finally, RRT* is selected as the underlying algorithm in HL-RRT* ..." [19].

B. Artificial potential field

For the other comparison algorithm an APF was selected. There are many types, though an interesting one was found in the form of a COLREGS compliant design [15]. COLREGS refers to international regulations for preventing collisions at sea, and there are other APF algorithms designed for this. The reason why this one is interesting is three-fold, because the simulations used in the paper match the ones planned here, it is computationally inexpensive, and has a qualitative comparison that included an MPC approach. Regarding the simulation, there is a mix of dynamic and static obstacles with a circular shape and random motion in a 2D environment with clearly defined goals.

The algorithm is described quite clearly and completely, and no additional clarifications or interpretations need to be made besides the ones written in the paper. Regarding the similarity of the algorithm with the MPC approach implemented here, there is a horizon in the form of a collision risk range. The only difference (besides not being able to explicitly define constraints) is that the algorithm in [15] is used to control the orientation of the robot, but it does not control its speed. Instead, the speed is assumed constant, which is a valid assumption in naval environments. The source paper [65] suggested possibly reducing the speed as the proximity to an obstacle increases, but this was not tested. As given in [15], by summing a set of clearly defined attractive and repulsive forces a resulting force defines the orientation. While in the negotiation zone COLREGS is hardcoded, so for dynamic obstacles the passing side is limited to starboard. A possible problem that could arise here comes from testing the algorithm for obstacles with limited changes in orientation. As a consequence, it might struggle with highly dynamic obstacles, or obstacles that do not follow COLREGS.

Finally, regarding the qualitative comparison, the authors of [15] claim MPC has a "high?" computational time. Specifically, [15] presents a table where a qualitative comparison was made between several algorithms (among which an MPC approach) and an artificial potential function. This did not inform the reader sufficiently on why they should select their algorithm, and the qualitative comparison could end up being misleading.

V. Case study setup

The first step towards proving the research objectives set forth is to create a simulation environment in which the tests can be held. To realize this, the input and output variables of interest can be used to help evaluate the simulations. First, by setting several performance metrics, the same criteria can be used to evaluate the control designs. Second, using the same parameters where applicable ensures the simulations are the same. To ensure this, these parameters are analyzed in greater detail and some supporting theory is used where needed. Finally all assumptions and their effects are summarized, such that the scope of the simulations is clear.

A. Performance metrics

When running the simulations a variety of information is available, in a variety of states. As such, selecting the performance metrics and the way they are measured is no trivial task. From [15], [19] and several others, success rate, path length and mission time (the time to reach the goal) are the three main dependent variables used to measure their performance. In fact, these are explicitly suggested in [9], and therefore used to evaluate the performance. A combination of a high success rate, low path length and little time is the ideal case.

Regarding success rate, the result to consider is whether the algorithm succeeds in a specific criterion. This is to reach the goal. However, there are two ways the goal might not be reached, be it either a livelock scenario (for example, circling in place) or due to a collision. It is decided that any collision instantly fails the simulation, since at that point the robust control has failed. In either case, the success rate is clearly defined, but when analyzing the failures, both scenarios are considered individually.

Path length is the second variable, and while success rate is a binary choice (fail/pass), path length has an associated value. Here either the ideal planned path length, the ideal motion planned length or the actually taken path length can be used. Considering the use of noise in the environment, the actually taken path length is considered, and those values are compared among the algorithms. Since the path length is a numeric value, the average and standard deviation are used to define the results. However, this is only done for the successful results, meaning some tabulated values might need additional explanation or are left empty.

Lastly, the simulated time in this case refers to the mission time. Though it is interesting to compare the algorithms in their computational time, this does not necessarily give a good insight into whether an algorithm is better suited for this environment. Instead more simulations are done but with different conditions. The first type limits the iteration budget of fmincon and limits the time allotted to sub-functions, to attempt to realize real-time performance. This also ensures the algorithms are tested with the same computational performance limitations in the form of a maximum control update step, akin to a time-marched simulation. Note that real-time simulations are not always based on a real-time operating system, but sometimes refer to near real-time, for example [55], [19]. To prevent misunderstandings, in this work these words are referred to as computationally constrained, or it is referred to a computational budget. The second type is allotted an unconstrained amount of time for each calculation, permitting the ideal path and motion planning approach. This way the mission time is considered from a mission optimization perspective, and is completely separate from computational time. The computational time is instead implemented as a time-marching condition that affects the success rate, with an influence on the path length and mission time.

B. Simulation scenarios

There are two types of scenarios that are interesting to consider. The first type contains several randomly placed obstacles that obstruct the direct path to the goal, but are not intended to form an infeasible problem. This is often used when evaluating the performance of path and motion planners, as is the case for [19] and [15]. In this scenario 6 static and 5 dynamic obstacles are randomly placed and used to evaluate the performance. This number is based on a visual inspection of the environment after initialization, ensuring that dynamic obstacles are in the way of the shortest path, but also ensuring the problem is always feasible. The positions, velocities, accelerations and points of attraction of the obstacles for each simulation are shown in Appendix A.

The intent is to ensure there would always be a feasible path. On the other hand, the second type contains obstacles that are intended to create a temporarily infeasible problem. Here an environment of 8 static and 8 dynamic obstacles was generated where the goal region would be placed behind a layer of dynamic obstacles. A robot can only pass if it considers the time-varying *trajectory* but not the time-invariant *path*. Though an infeasible problem most of the time, there are small time windows where the robot could potentially squeeze in between dynamic obstacles. Due to the cyclical motion of the dynamic obstacles a path is cyclically feasible, which is intended to assist the biased sampling of HL-RRT*. These environments are designed to assess not only the performance, but rather also the difficulties and limitations in exploring motion planning in high complexity environments.

One of the final concerns regarding the simulation scenarios is the noise. Obstacle noise and state noise are assumed additive and uncorrelated, affecting only the positions. The noise is randomly generated, but it is generated before testing and supplied to the different algorithms such that they are tested with the same noise profiles. However, another issue that affects reproducibility is related to the solver. The number of iterations in a solver might not be enough to find the best possible constraint-adhering set of inputs in the control scheme with a low iteration budget, which could terminate the optimization prematurely. Performance with a low computational budget is thus difficult to guarantee.

To show that the performance falls within some bounds, results are shown after repeating the tests 10 times for the complex case, and once for each of the simple environments, displaying the average and standard deviation in the process. Therefore, the results are shown for different simulation environments, repeated 10 times with different noise profiles. Besides that, two more aspects are varied, namely the type of algorithm (heuristic and MPC, HL-RRT*, APF) and whether the simulations are done with a low or high computational budget.

C. Parameter values

Having addressed the dependent and independent variables, the software and hardware configuration could also affect the simulation. To limit that influence, the same parameters are used wherever possible. With regard to the hardware variables,

the CPU is a 4 core 2.5 GHz Intel® Core™ i7-4710MQ with 8GB of RAM memory and an integrated GPU that is Intel® HD Graphics 4600. The operating system is Ubuntu 18.04.6 LTS, a 64-bit OS, with open-source drivers where applicable. The simulations are done on MATLAB R2020b. Though not needed, for fear of the influence of performance throttling, all background processes in the OS were turned off when running simulations, and MATLAB was limited to a single core.

Having addressed the simulation hardware, the remaining parameters were all used to defined values for the simulation. However, even with general values there are differences among the algorithms that lead to different results. One example is the APF approach, which has no velocity constraint since it is designed for constant speed. So, in as far as it is possible, all simulations on the case studies are done using the variables presented in Table I.

TABLE I
SIMULATION VARIABLES USED IN THE RANDOM ENVIRONMENTS, WHERE APPLICABLE.

Variable	Value
Control time step, Δt_{ctrl}	0.2
State noise per Δt_{ctrl}	$\leq 0.2\, u_{\max}$
Dynamic obstacle noise per Δt_{ctrl}	$\leq 0.5\, u_{\max}$
Environment upper bound, $x_{\max}$	[12,12]
Environment lower bound, $x_{\min}$	[-2,-2]
Minimum-maximum speed, $u_{\max,1}$	[-0.1,1]
Minimum-maximum turn rate, $u_{\max,2}$	[-1,1]
Smoothness limits, u_{smooth}	[0.4,1]
Robot radius, ρ_{rob}	0.5
Obstacle radius, ρ_{obs}	0.5
Sensor radius, ρ_{det}	5
Goal radius, ρ_{goal}	0.5
Initial position, x_0	[0,0]
Goal position, x_g	[10,10]

The control time step size is sourced from the original heuristic algorithm described in [18]. The values describing the vehicle dynamics are inspired by the TurtleBot3 (1.82 rad/s, 0.26 m/s), but the difficulty of the control problem is increased by lowering the turn rate and increasing the maximum possible speed.

The other parameters are tuned to ensure the robot has the solution space for ad-hoc motion planning. For example, the dynamic and static obstacle values listed in Appendix A are tuned with the environment bounds and obstacle radii to ensure a a sufficiently simple or cluttered environment. The MPC specific parameters are shown in Table II.

TABLE II
PARAMETERS USED FOR THE MPC CONTROLLER.

Variable	Value(s)
Low budget fmincon max iterations	10^3
High budget fmincon max iterations	10^5
Prediction horizon, H_p	5
Control horizon, H_c	3
Reference trajectory weights, Q	[4,4,1]
Control input weights, R	[1,0]
Goal weights, P	[10,10,0]

The low computational budget limits the iterations in fmincon, such that the combination of these parameters and hardware means a solution is generated within 0.18 seconds per control time step. However, the results are not guaranteed to be local minima. By contrast, the high computational budget is designed to use more iterations to find a local minimum, but can take up to 12 seconds per control time step. The other variables in Table II are tuned using [66] and testing to improve the computational speed, lower costs and improve constraint satisfaction. Combining Table II with Table I and Appendix A, enables reproducing the results unconstrained by the iteration budget. The low budget results, however, could be influenced by the premature stop in the solver.

Second, HL-RRT* is considered. Most HL-RRT* specific parameters used in the controller are shown in Table III, and are sourced from test cases used in [19]. Anything that is not listed as a variable is hard-coded, as a function based on underlying theory or papers. For example, the volume of a unit n-dimensional sphere $\zeta_{\text{dim}=2} = \pi$ is used to find the steering radius [54]. Despite combining Table III with Table I and Appendix A, due to the probabilistic sampling-based nature of the algorithm, the same results cannot be guaranteed. Note that the short time step is designed to be the same length as the control time step for fair comparison.

TABLE III
PARAMETERS USED FOR THE HL-RRT* CONTROLLER.

Variable	Value(s)	units
Maximum number of nodes, n_{samp}	1500	-
Short-term steering step size, t_s	0.2	s
Lazy steering step size, t_l	0.5	s
Predictive horizon, TH	10	s
Goal sampling bias, ϵ	0.01	-
Learned-set sampling bias, ζ	0.4	-
Bounded disturbance (robust), δ	$0.1t_s \cdot u_{\max}$	m
Maximum steer (and/or) rewiring radius, η	$t_s \cdot u_{\max}$	m
Heuristic learning parameter, k	1.3	-
Convergence threshold, ϵ_{EM}	10^{-3}	-
Regularization value, γ_{EM}	10^{-4}	-
EM maximum iterations	100	-

Compared to the original algorithm in [19], the low computational budget approach splits the sampling after pruning over several time steps. This way, if a sizable number of nodes need to be regrown, obstacle avoidance can be performed despite the time-marched control update step. The high computational budget approach regrows all the nodes before moving, pausing the environment simulations in the process.

Finally, most APF specific parameters used in the controller are shown in Table IV. Combined with Table I and Appendix A, this allows reproducing the results, barring the noise profile. The APF is not computationally constrained by the time-marched simulation, and thus no changes are implemented to make it compatible with a low budget. Note that the influence range is defined to have the same length as the detection range in the MPC approach.

VI. SIMULATION RESULTS

In this section results are presented for the MPC, HL-RRT* and COLREGS compliant APF methods based on a sample from the case study. Note that any visualized results are

TABLE IV
PARAMETERS USED FOR THE APF CONTROLLER.

Variable	Value(s)	units
Attraction weight, ϵ	3000	-
Dynamic obstacle scaling factor, η_d	2000	-
Static obstacle scaling factor, η_s	300000	-
Emergency maneuver scaling factor, η_e	2000	-
Artificial safety margin, τ	0.3	m
Influence range, ρ_o	$\rho_{\text{det}} - \rho_{\text{rob}} - \rho_{\text{obs}}$	m
Safety radius, ρ_{safe}	0.5	m

samples from a set of generated paths, from a set of generated problems.

A. Model predictive control

Samples of the results for the simulations with the simple dynamic environment are shown in Figures 11-13 and samples of the complex dynamic environment results in Figures 14-16.

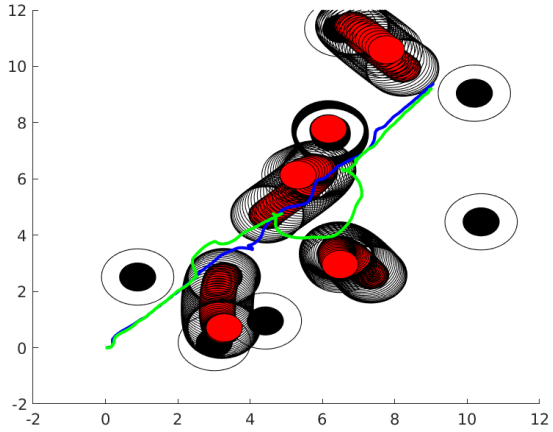


Fig. 11. Sample robot paths generated by MPC, using high (blue) and low (green) computational budget, in a simple noisy environment, with static (black) and dynamic obstacles (red).

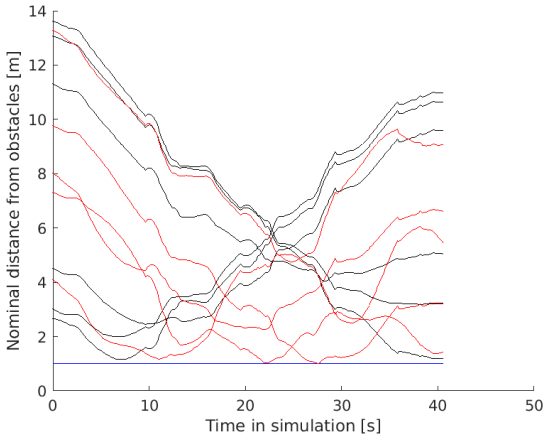


Fig. 12. Nominal (real) distance from static (black) and dynamic obstacles (red) using MPC with a high iteration budget in a simple environment, with the minimum safety radius (blue).

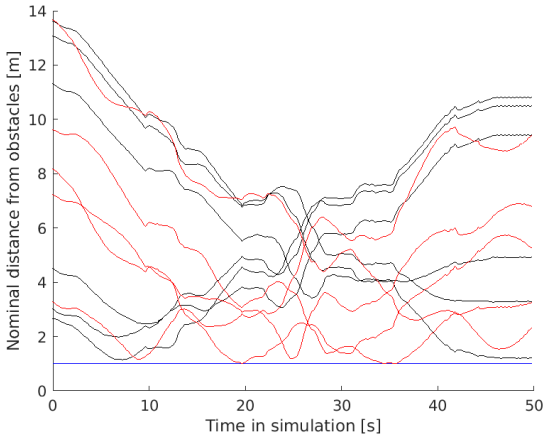


Fig. 13. Nominal (real) distance from static (black) and dynamic obstacles (red) using MPC with a low iteration budget in a simple environment, with the minimum safety radius (blue).

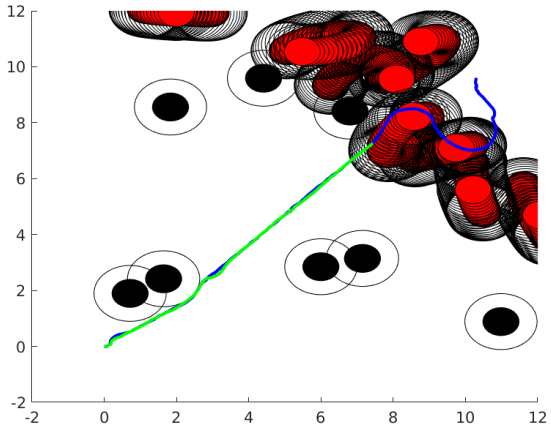


Fig. 14. Sample robot paths generated by MPC, using high (blue) and low (green) computational budget, in a cluttered noisy environment, with static (black) and dynamic obstacles (red).

B. Horizon-based lazy RRT*

Similar to the MPC approach, the simple dynamic environment results are shown in Figures 17-19 and the complex dynamic environment results in Figures 20-22. Only one modification has been made to support the HL-RRT* in case of a low computational budget, which was to spread out the "regrowing" of the RRT post pruning over several time steps. Otherwise, the HL-RRT* would halt in place for too long, causing unnecessary crashes with dynamic obstacles.

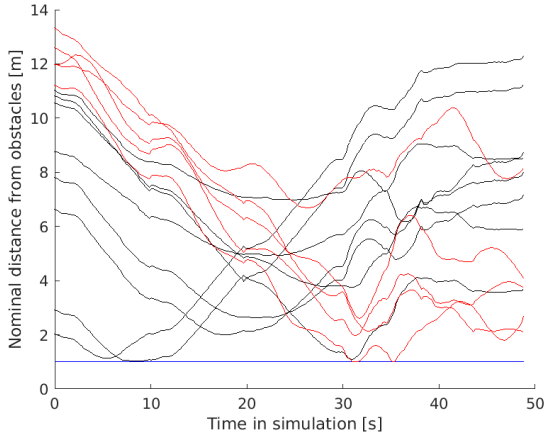


Fig. 15. Nominal (real) distance from static (black) and dynamic obstacles (red) using MPC with a high iteration budget in a cluttered environment, with the minimum safety radius (blue).

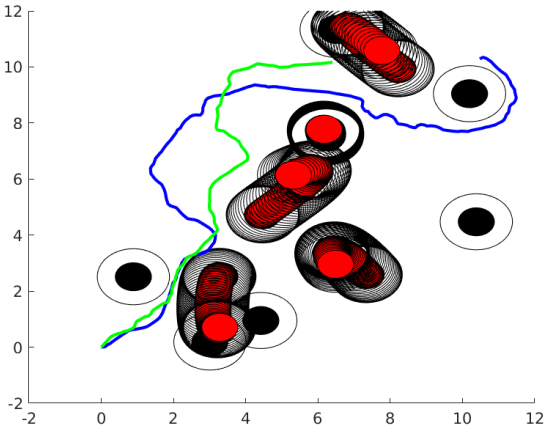


Fig. 17. Sample robot paths generated by HL-RRT*, using high (blue) and low (green) computational budget, in a simple noisy environment, with static (black) and dynamic obstacles (red).

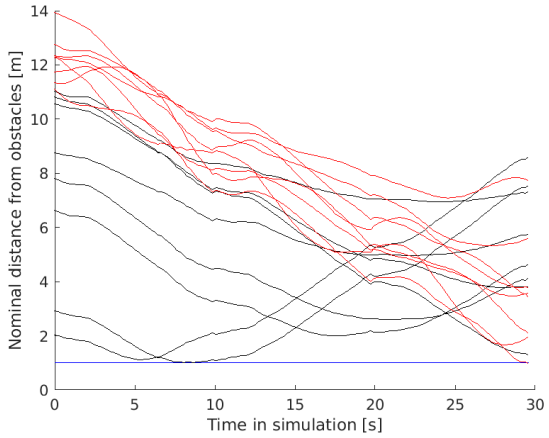


Fig. 16. Nominal (real) distance from static (black) and dynamic obstacles (red) using MPC with a low iteration budget in a cluttered environment, with the minimum safety radius (blue).

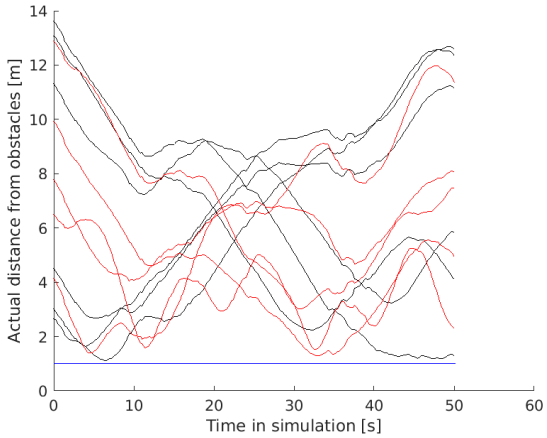


Fig. 18. Nominal (real) distance from static (black) and dynamic obstacles (red) using HL-RRT* with a high computational budget in a simple environment, with the minimum safety radius (blue).

C. Artificial potential function [15]

In contrast to the other two approaches, the APF is so quick that the performance is not impacted by the computational budget. In other words, it is never bottle-necked by computational limitations in these simulations. Unfortunately, this does not mean the results are good. Therefore only one result is plotted for the simple case in Figures 23-24 and the cluttered case in Figures 25-26.

D. Summarized results

Considering the size the images take up, the remaining results are tabulated. The first set of results is summarized in Table V for a variety of simple environments. The second set are summarized in Table VI using the same environment as in Figure 14 with the only difference being the obstacle and state noise.

TABLE V
SIMPLE ENVIRONMENT RESULTS USING DIFFERENT COMPUTATIONAL BUDGETS, OBTAINED FROM 10 DIFFERENT ENVIRONMENTS.

High budget	MPC		APF COLREGS		HL-RRT*	
	μ	σ	μ	σ	μ	σ
Success rate	100%		10%		100%	
Path length (m)	17.4	1.78	15.4	-	20.5	3.18
Time taken (s)	41.4	6.13	30.8	-	40.7	7.74
Low budget	**MPC**		**APF COLREGS**		**HL-RRT***	
	μ	σ	μ	σ	μ	σ
Success rate	100%		10%		70%	
Path length (m)	19.2	2.40	15.4	-	20.0	2.99
Time taken (s)	48.6	8.50	30.8	-	44.5	8.94

VII. DISCUSSION

Although all methods were designed for dynamic environments, a large variety of results is noted. In this section,

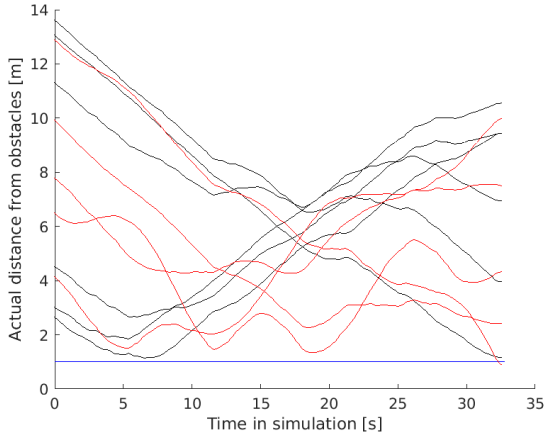


Fig. 19. Nominal (real) distance from static (black) and dynamic obstacles (red) using HL-RRT* with a low computational budget in a simple environment, with the minimum safety radius (blue).

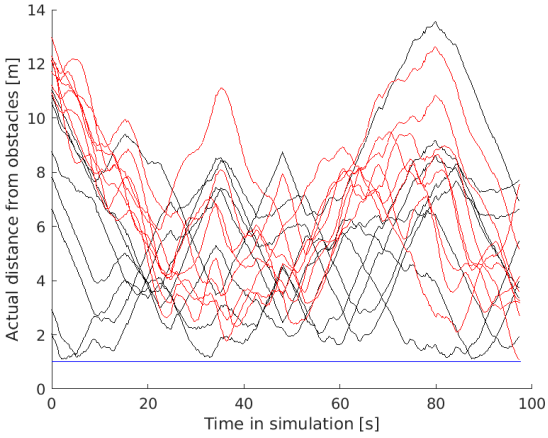


Fig. 21. Nominal (real) distance from static (black) and dynamic obstacles (red) using HL-RRT* with a high computational budget in a cluttered environment, with the minimum safety radius (blue).

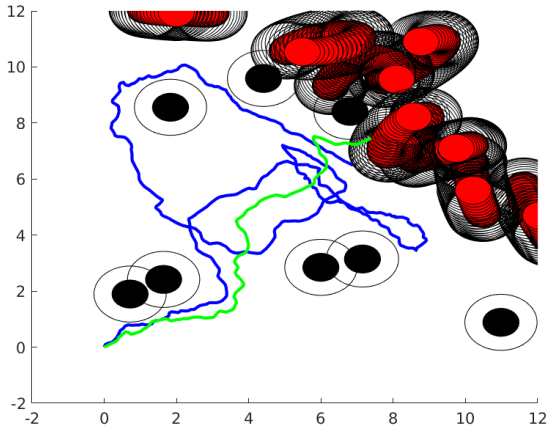


Fig. 20. Sample robot paths generated by RRT, using high (blue) and low (green) computational budget, in a cluttered noisy environment, with static (black) and dynamic obstacles (red).

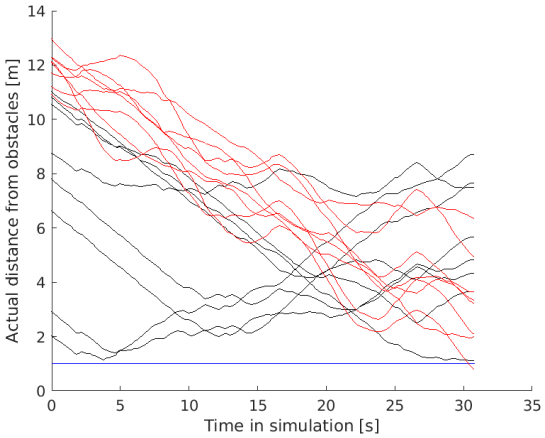


Fig. 22. Nominal (real) distance from static (black) and dynamic obstacles (red) using HL-RRT* with a low computational budget in a cluttered environment, with the minimum safety radius (blue).

TABLE VI
CLUTTERED ENVIRONMENT RESULTS USING DIFFERENT COMPUTATIONAL BUDGETS, OBTAINED BY CHANGING THE NOISE PROFILE 10 TIMES.

High budget	MPC		APF COLREGS		HL-RRT*	
	μ	σ	μ	σ	μ	σ
Success rate	70%		0%		0%	
Path length (m)	18.0	1.16	-	-	-	-
Time taken (s)	46.8	4.01	-	-	-	-
Low budget	**MPC**		**APF COLREGS**		**HL-RRT***	
	μ	σ	μ	σ	μ	σ
Success rate	20%		0%		0%	
Path length (m)	20.0	1.26	-	-	-	-
Time taken (s)	49.4	2.26	-	-	-	-

the results are presented alongside possible reasons behind the differences, exploring the limitations and strengths of the algorithms in the process. This way possible recommendations can be identified that could be implemented in future work.

First the APF is addressed in detail due to its lacking performance in these simulations. Following, both case studies are addressed separately, where a general overview of the results and performance is described. An in-depth analysis is done that attempts to illustrate the reasons behind the various success rates, path lengths and time taken.

A. APF

First off, the APF only has a single success from the 10 simple environment tests. This case had a wide swath of free space across the diagonal between the origin and goal points. Besides that success, failures occurred due to the planner getting stuck in a figure eight path, as visible in Figure 25, which could be related to the tuning of the weights with respect to the sensor range and velocity. Another possibility is that the tests in [15] were for a relatively open space, with the dynamic obstacles moving in straight lines excepting

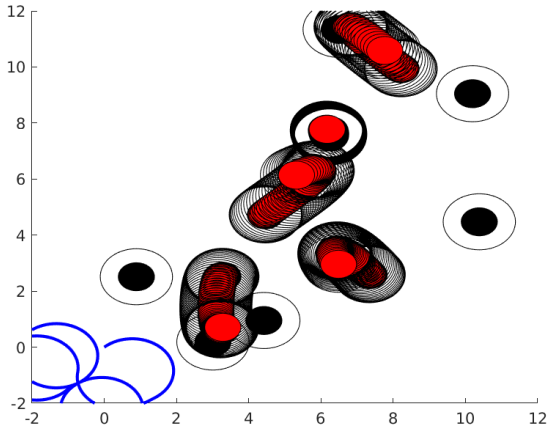


Fig. 23. Sample robot path (blue) using APF in a simple noisy environment, with static (black) and dynamic obstacles (red).

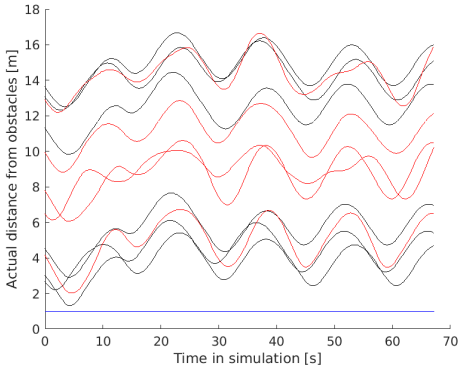


Fig. 24. Nominal (real) distance from static (black) and dynamic obstacles (red) using APF in a simple environment, with the minimum safety radius (blue).

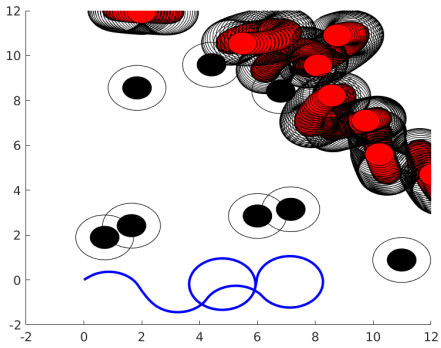


Fig. 25. Sample robot path (blue) using APF in a cluttered noisy environment, with static (black) and dynamic obstacles (red).

an instantaneous course correction. In other words, the APF algorithm is designed and tested for obstacles moving in a straight line, not the curved trajectories used in this work.

However, since the APF struggles to even pass the static obstacles in Figure 25 and Figure 23, it is much more likely

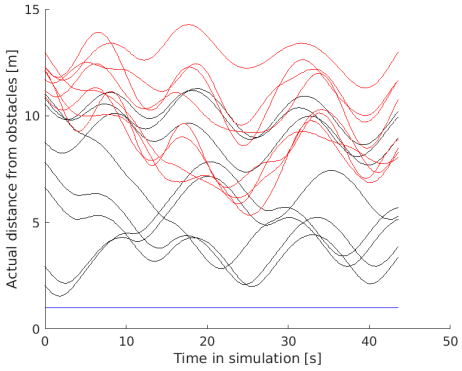


Fig. 26. Nominal (real) distance from static (black) and dynamic obstacles (red) using APF in a cluttered environment, with the minimum safety radius (blue).

that every different environment requires a re-tuning. Initial testing and results from other implementations, such as [67], confirms that this is more likely the case. To test the first idea, the factors are re-tuned for the current conditions, with the static obstacle scaling factor changed to $\eta_s = 3,000$, and the emergency maneuver scaling factor changed to $\eta_e = 200,000$, ceteris paribus. When done so, the figure eight livelock ceases to exist, and instead a trajectory is found, see Figure 27.

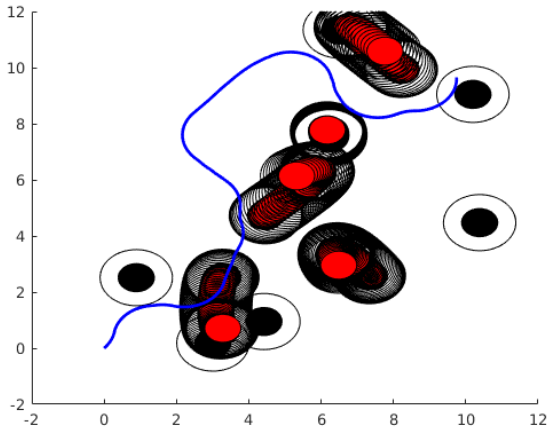


Fig. 27. Trajectory generation (blue) of APF following sub-optimal re-tuning of scaling factors ($\eta_s = 3,000, \eta_e = 200,000$, ceteris paribus).

This confirms that when the sensor parameters or velocities change, the other weighted parameters must change as well. There are limits to this approach, especially in uSaR where such knowledge and pre-tuning cannot be guaranteed. With this in mind a supporting algorithm, perhaps in the form of a neural network, can be used to modify the weights online for the dynamic environment. However, without such a supporting function, the APF approach is not as useful for uSaR or another dynamic environment that has limited information.

On the other hand, the MPC and RRT tuning is not as correlated to the parameters from the environment. This means that they do not require substantial re-tuning whenever they are tested for a new problem. Regardless of the situation, the

original tuning did ensure safe motion as no simulation in Table V or Table VI was due to an obstacle crash.

B. Simple environment

Following the results summarized in Table V the MPC approach is shown to outperform the HL-RRT* when computationally constrained. All HL-RRT* failures were obstacle crashes, and can be traced back to a specific "crushing" motion scenario. In this scenario the robot ends up between two dynamic obstacles or a dynamic and static obstacle, and does not find a new lower cost path in time that would move it to safety. The crushing motion scenario is illustrated in Figure 28.

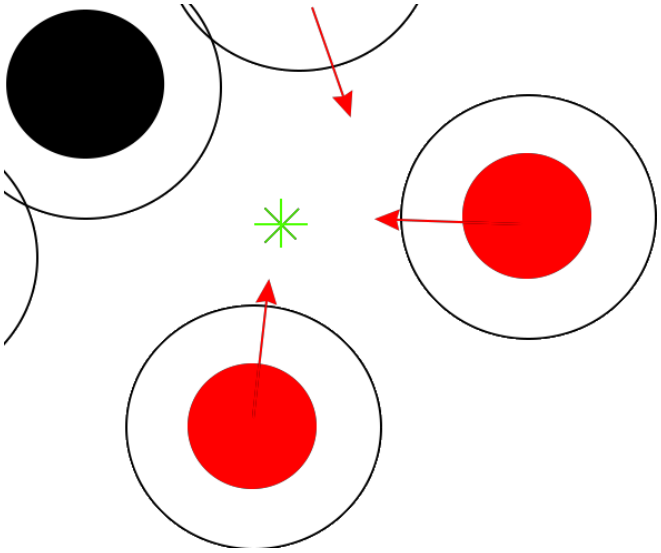


Fig. 28. Crushing motion scenario. Three dynamic obstacles (red) move towards the robot (green star).

This is a challenging scenario for any motion planner, and the MPC therefore also struggles here. The difference in their performance can thus better be explained by the proclivity of the motion planner in the MPC to avoid this scenario. By considering obstacle positions at multiple possible locations, the trajectory is pre-tested to an extent. Thus this scenario either creates a dynamic obstacle belt to be avoided, or the reference trajectory is set to update outside the "crushing" region.

Returning to the HL-RRT*, it can handle this scenario in with a high computational budget, but the robot cannot always manage to find a better path with a low budget. So it follows that a computational bottleneck is the cause of this problem, and not a fundamental concept. When analyzing the algorithm, a robot would only move on a path that leads to a lower cost, which in this case means closer to the goal. So unless a node is found that is closer to the goal than its current position, a robot controlled by HL-RRT* will simply not move. In the tests using a high computational budget such a node is always found, which is why there is a 100% success rate.

What is concerning is that one of the contributions in the HL-RRT* is introducing the Gaussian mixed elite set using [64], which caused a substantial decrease in performance when computationally constrained. According to the tuning tool available in MATLAB, the eliteset calculation and node re-growth takes over 80% of the computational time. Though [64] shows good performance compared to its peers, the expectation-maximization function is called whenever replanning is required and has a sizable "elite" set that can grow larger than the tree. Due to their exclusion of this step, the RRT* and RRTx algorithms have a higher iteration speed. Higher iteration speed leads to a larger chance that a better node, enabling collision avoidance by the time the dynamic obstacles would otherwise collide with the robot.

When comparing path lengths, the HL-RRT* seems to generally take a longer path compared to the MPC controller, while the path is usually shorter when generated with a higher computational budget. Table V does display a shorter average path length for the HL-RRT* when using a lower computational budget, but this is because only the successful tests are considered, which brought the average down. In a few cases, however, this also happened because of the random nature of HL-RRT*. This is somewhat visualized in the path planned by the HL-RRT* in Figure 17, where there is a wide berth made to avoid the middle obstacles. The reason behind this wide berth is partly due to the sampling and partly because of the local cost propagation in the tree, a limitation of the RRT*. The RRTx fixes this through a global cost update. This is confirmed when considering the tree that is used at the time of the berth, shown in Figure 29.

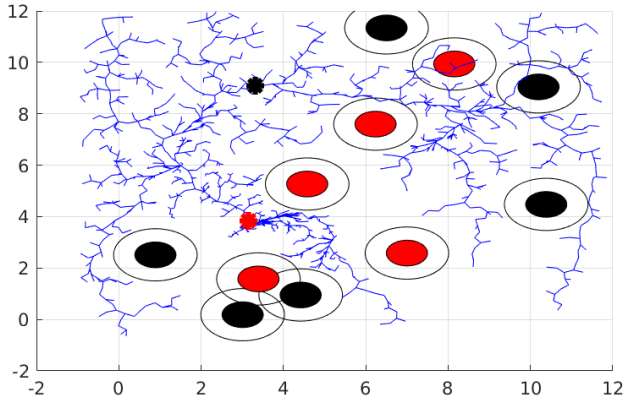


Fig. 29. Tree structure (blue) of the HL-RRT* with a high computational budget, at the time of the berth in Figure 17, with dynamic (red), and static (black) obstacles. Collision checking is done from the current position (red star) to the horizon (black star).

To clarify, some path lines seem to pass through obstacles in Figure 29, but this is how HL-RRT* works. By only collision checking the lines within a time horizon from the current position it saves calculation time by reducing the use of the checking function, which is usually the slowest part in RRT [19]. This is an acceptable simplification because the dynamic nature of the environment implies that any branches in the future (past the horizon) would need to be re-tested in case of an unexpected change.

By comparison, Table V, shows a shorter average path length for the MPC compared to the HL-RRT*. When com-

pared with itself, the MPC takes an approximately 10% longer path length on average when computationally constrained. This can be explained using the behavior of the MPC in Figure 11, which shows a small berth in the green line (low iteration budget), which is avoided in the blue line (high iteration budget).

The reason behind the overcompensation in Figure 11 is the limited knowledge of the obstacle dynamics coupled with a limited computational window to find an optimal path. Since the MPC assumes obstacles move in straight lines of constant velocity, with a bounded noise, it sometimes overcompensates despite there being no reason for it. To successfully avoid the obstacles (see Figure 13), the computationally constrained MPC overcompensated and even brought the robot in proximity to another dynamic obstacle. In contrast, the MPC with a high iteration budget (blue line) found a cheaper path by only slightly changing course and speeding up. In the subsequent iterations the high budget MPC methodalso yields a lower cost as it almost always finds a local minimum, in contrast to the computationally constrained version which sometimes stops prematurely.

Regarding the mission time, the HL-RRT approaches are faster on average. The same cost function and constraints are used where possible, so the HL-RRT* approach simply performed better. The most effective change would be to redesign how the reference trajectory is generated and used in the MPC. Figures 12 and 13 show the distances of the MPC controlled robot to obstacles over time. Looking at the slopes with respect to the static obstacles (black lines), variations in speed can be noted. Here steeper curves are found at a time of 28 and 25 seconds, for the low and high budget controller respectively. There are other times where the slopes change, but also note that during regular movement, as is the case in the first 10 seconds, the controller usually sticks to lower speeds. Therefore a possible point of improvement is to optimize the reference trajectory based on parameters or the cost function.

Summarizing the simple environment analysis, the MPC approach has a higher success rate and produces a shorter path than the HL-RRT* and APF, with both a high and a low computational budget.

C. Complex environment

Having identified some limitations of the HL-RRT*, the reasons why the computationally constrained trajectories ended up crashing into obstacles become clearer. Unfortunately, the high budget HL-RRT* also failed to show a single success. Theoretically, multiple collision-free paths are possible at various times through the moving wall environment despite the constraints imposed by the bounded noise. In the case of using a high computational budget, none of the failures of the HL-RRT* were due to obstacle collision, but simply artificial time outs when no feasible path was found even after 2 minutes.

When looking at the blue (high budget) path found in Figure 20, the algorithm seems to explore a variety of possible entrances, but is quick to dismiss them. This follows from the lack of collision-checking beyond the time horizon. The path the HL-RRT* finds is also not based on the trajectory of obstacles or a velocity obstacle, but instead on their actual positions. The main reason behind the back-and-forth motion is that a generated path is infeasible, but is beyond the time horizon so the HL-RRT* planner believes it is feasible. By not checking obstacle trajectories, a path may be discarded despite it being feasible in the near future. Note that by not accounting for the obstacle motion, there are no successes but also no collisions.

The same cannot be said for the MPC approach. Successes have been noted, even some for the computationally constrained case. However, even when there is a high iteration budget, the algorithm still failed by collision in 3/10 simulations, see Table VI. This is a concerning result, since this means the safety of the robot is not robustly guaranteed. Another way to interpret the failures is that because noise is the only variable in these simulations, noise can lead to a collision despite the intent to design the MPC robust to noise. All three failures in the high computational budget case can be traced back to the same crushing failure mode of Figure 28, in the same location, where the robot entered a zone after which no control inputs could lead it back to safety.

On the other hand, the MPC with a low iteration budget crashed 7 times due to the crushing failure mode, and in all seven simulations the minimization function (fmincon) reached maximum iterations without finding a safe trajectory. It also failed once by exceeding the velocity constraint instead. To test whether the scenario can be salvaged if more iterations were allotted, the problem is re-solved using a larger number of iterations (same as the high budget) upon passing point (8,8) in Figure 14. Three simulations still failed, despite these conditions.

To understand why the MPC failed in these scenarios, the states and knowledge of the robot near the time of collision were looked at. First, before the crashes, the path planner created a feasible, collision-free trajectory. Second, the crash and entrapment scenario happened somewhere in the middle of the robot position at the planning stage and temporary goal. Third, when increasing the MPC prediction horizon to 11, and control input horizon to 9, the robot avoids the crash.

While the short-term noise is successfully accounted for in the tube approach of the MPC, the limited update rate of the reference trajectory results in a dangerous accumulation of errors. Concerns regarding the accumulation of errors were already identified in the motion planner design, which resulted in later points not being weighted as much. The problem is, no alternative reference trajectory is provided for the MPC for the weights to matter. To fix this, two approaches are considered:

1) Increasing the MPC prediction and control window such that the robot manages to avoid these traps. The downside is the additional computation which will make the computationally constrained controller perform even worse for the current control update rate.
2) Increasing the update rate of the heuristic motion planner. One downside is that the heuristic planner minimizes the path length for motion planning, and does not consider other variables. In other words, since only the path length is considered, turning 180 degrees in its place has no influence in its decision on which path is shorter, despite

this possibly causing sizable inefficiencies or infeasible scenarios. Another possible problem is the additional computational effort. Though it finds a trajectory within a second, doing so frequently would require a design that accounts for this delay.

For these two cases, it must be said that the heuristic path and MPC motion planner combination controller performs best when compared to the other algorithms with respect to success rate as well as path length. This is especially true when the computationally limited performance is considered, as the controller is shown to work under less than ideal conditions, and is shown to be feasible for this specific hardware and software configuration. Additional testing must be done with hardware-in-the-loop to ensure that it is feasible outside of the simulated environment. Though the mission time is higher on average compared with the HL-RRT*, the controller does ensure a lower path length. By modifying the way the reference trajectory is used, this controller can possibly have better results in all performance metrics. Moreover, with some additional development invested into the heuristic path planner this combination could even tackle more complex environments.

VIII. Conclusion

In this study a local controller was developed that can perform path and motion planning in dynamic environments with limited knowledge of moving obstacles. Sourced from [18], the greedy heuristic path planner was successfully modified for dynamic obstacle environments, while a nonlinear MPC approach performs trajectory tracking. The controller is made robust to noise through tuning, robust control theory and limited integration between the heuristic motion planner and MPC. The noise is modelled to be additive and uncorrelated, and affects the robot and dynamic obstacle positions. Assuming the linear velocity of the obstacles is measured from timestep to timestep, a noise bound on an obstacle position of up to 50% of the input envelope per time step was tested to succeed.

Tested against two state-of-the-art approaches (based on APF and RRT), it is shown to have the same or higher success rate in cases with a low and a high computational budget. It also has shorter path lengths on average, despite the HL-RRT* reaching the goal quicker. The RRT and MPC based controllers have proven to be able to produce collision-free trajectories despite having multiple dynamic obstacles. The APF approach however seemed to be tuned for a specific problem type, meaning the algorithm failed to succeed in a different, generalized case. When considering a highly complex scenario the MPC approach has a number of successful trajectories, with a 70% success rate using a high computational budget, and 20% using a low computational budget, showing that through some improvements in computational performance, the results of the control algorithm can be improved. This is an improvement over the 0% success rate of the comparison methods.

Focus was placed on developing the controller, modifying the path planner, and designing a case study in which the three approaches are compared, such that results can be evaluated quantitatively. This work proves that MPC with a guiding path planner is a viable motion planner for the jogger's problem in a time-varying environment. On top of this, several sources of possible improvements are found. Note that the controller has not been optimized in terms of computational performance, which leaves more opportunities for performance improvements. Especially in the case of highly complex environments such as those present in uSaR.

A. Future work

1) Reflection: During the course of the research, several possible improvements have been identified. Planning-wise, the problem is initially defined quite generally, where an uSaR environment was meant to be tackled. The aim was to solve an uSaR scenario such as in the RoboCup Rescue simulation, with a newly developed motion and path planner combination. Though not infeasible in the time constraint of a master thesis, when developing the controller architecture, literature in this field usually takes one of three paths.

- One path is solving the global problem or making a simulation for it, using environmental variables (such as pre-existing human motion or fire spread models). Often limited to a scope defined by the sourced models, further development could then be done by refining the model by including responses, task allocation, or improving the resolution with other models.
- Another path is the development of a combination of a global and local controller, which often requires a number of simplifying assumptions to obtain results. Further development, if done, is suggested by abolishing each assumption step-by-step, bringing it closer to hardware.
- The last path is employed here, where a detailed subproblem is solved, before moving on to a larger problem. Further development is done to increase the scope of the problem, combine it with other solutions, or to try to implement it in hardware.

2) Recommendations: One recommendation is to create a simple set of unified model-in-the-loop test cases, and start benchmarking various available controllers. This way a dataset for researchers and engineers using ad-hoc simulators can be used to quantify the results of their developments. This will not only help future research when intending to choose a model to expand on, but it will help put other work into context in this field of research.

Although the controller is tested with a noise affecting the obstacle position of up to 50% of the control input range, no analytical limit was determined. A future study could be made in determining the limits of the noise such a controller can be made robust to. There are other technical recommendations for the comparison papers. By comparing various RRT approaches such as RRTx, RRT-rope and such, a valuable insight into the strengths of the RRT approach was noted, namely the *rapid* sampling and growth of the random tree, and thus the weakness of HL-RRT* (the repeated expectation-maximization). To best improve the HL-RRT*, an increase in computational performance can be done by decreasing the magnitude of pruning and regrowing steps. In

the case with a high computational budget, the complex case study suggests that the HL-RRT* could benefit from sampling space-time (thus considering predicted obstacle motion) rather than simply sampling space. Trying to find an optimal path in a heavily cluttered environment was not feasible in the case study, despite there being feasible trajectories. As far as future research for the COLREGS APF, or even APF in general, the correlation between environmental and robot parameters with the scaling factors would be interesting to find. With that in mind, to make the APF more robust to limited knowledge environments, a possible future development is to combine the APF with an MPC (as done in [67]), but this would require some additional development and tuning.

The short-term goal with respect to the heuristic motion planner would be to develop and test the controller with limited perception and make it work with other vehicle models. The ideal multi-robot team for an uSaR scenario is heterogeneous after all. The motion planner, as implemented, detects all obstacles within the sensor range, despite the line of sight possibly being broken. Addressing blind spots and suddenly appearing obstacles would ensure robust obstacle avoidance under the line-of-sight condition, enabling higher order simulations such as uSARSim. Additional online support would likely be needed, for example through a global localization algorithm or an additional local swarming condition.

In order to avoid any possible tech debt, it is suggested to develop a (multi robot) global controller only after the local controller is tested for a variety of system models. Much like how the COLREGS APF method is tuned for a specific problem and vehicle model, it is possible that the MPC approach suffers the same problem. Following that, the long-term development goal would be to translate the controller into Robot Operating System (ROS) 2.0 and test it by possibly combining it with the TurtleBot software and testing the global and local controller in uSARSim or RoboCup Rescue simulations.

REFERENCES

[1] R. R. Murphy, *Disaster robotics*, ser. Intelligent Robotics and Autonomous Agents series. Cambridge, MA: MIT press, 2014.

[2] M. Statheropoulos, A. Agapiou, G. Pallis, K. Mikedi, S. Karma, P. J. Vamvakari, M. Dandoulaki, F. Andritsos, and C. Thomas, "Factors that affect rescue time in urban search and rescue (usar) operations," *Natural Hazards*, vol. 75, 01 2015.

[3] J. Qi, D. Song, H. Shang, N. Wang, C. Hua, C. Wu, X. Qi, and J. Han, "Search and rescue rotary-wing uav and its application to the lushan ms 7.0 earthquake," *Journal of Field Robotics*, vol. 33, no. 3, pp. 290–321, 2016.

[4] A. Rom and I. Kelman, "Search without rescue? evaluating the international search and rescue response to earthquake disasters," *BMJ Global Health*, vol. 5, no. 12, 2020. [Online]. Available: https://gh.bmj.com/content/5/12/e002398

[5] Y. Liu and G. Nejat, "Robotic urban search and rescue: A survey from the control perspective," *J. Intell. Robotics Syst.*, vol. 72, no. 2, p. 147–165, Nov. 2013.

[6] S. Doocy, A. Daniels, C. Packer, A. Dick, and T. D. Kirsch, "The human impact of earthquakes: a historical review of events 1980-2009 and systematic literature review," *PLoS currents*, vol. 5, 2013.

[7] J. Rajan, S. Shriwastav, A. Kashyap, A. Ratnoo, and D. Ghose, "Disaster management using unmanned aerial vehicles," in *Unmanned Aerial Systems*. Elsevier, 2021, pp. 129–155.

[8] S. Grogan, R. Pellerin, and M. Gamache, "The use of unmanned aerial vehicles and drones in search and rescue operations–a survey," *Proceedings of the PROLOG*, 2018.

[9] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous uav guidance," *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1, pp. 65–100, 2010.

[10] M. Mohanan and A. Salgoankar, "A survey of robotic motion planning in dynamic environments," *Robotics and Autonomous Systems*, vol. 100, pp. 171–185, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889017300313

[11] C. Katrakazas, M. Quddus, W.-H. Chen, and L. Deka, "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions," *Transportation Research Part C: Emerging Technologies*, vol. 60, pp. 416–442, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X15003447

[12] M. Hoy, A. S. Matveev, and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey," *Robotica*, vol. 33, no. 3, pp. 463–497, 2015.

[13] M. A. Mousavi, Z. Heshmati, and B. Moshiri, "Ltv-mpc based path planning of an autonomous vehicle via convex optimization," in *2013 21st Iranian Conference on Electrical Engineering (ICEE)*. IEEE, 2013, pp. 1–7.

[14] J. Marzat, S. Bertrand, A. Eudes, M. Sanfourche, and J. Moras, "Reactive mpc for autonomous mav navigation in indoor cluttered environments: Flight experiments," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 15 996–16 002, 2017, 20th IFAC World Congress. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896317325375

[15] H. Lyu and Y. Yin, "Colregs-constrained real-time path planning for autonomous ships using modified artificial potential fields," *The Journal of navigation*, vol. 72, no. 3, pp. 588–608, 2019.

[16] Y. Mualla, A. Najjar, A. Daoud, S. Galland, C. Nicolle, A.-U.-H. Yasar, and E. Shakshuki, "Agent-based simulation of unmanned aerial vehicles in civilian applications: A systematic literature review and research directions," *Future Generation Computer Systems*, vol. 100, pp. 344–364, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18328462

[17] F. Rosique, P. J. Navarro, C. Fernández, and A. Padilla, "A systematic review of perception system and simulators for autonomous vehicles research," *Sensors*, vol. 19, no. 3, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/3/648

[18] A. Jamshidnejad and E. Frazzoli, "Adaptive optimal receding-horizon robot navigation via short-term policy development," in *15th International Conference on Control, Automation, Robotics and Vision, ICARCV 2018, Singapore, November 18-21, 2018*. IEEE, 2018, pp. 21–28. [Online]. Available: https://doi.org/10.1109/ICARCV.2018.8581157

[19] Y. Chen, Z. He, and S. Li, "Horizon-based lazy optimal rrt for fast, efficient replanning in dynamic environment," *Autonomous Robots*, vol. 43, no. 8, pp. 2271–2292, 2019.

[20] A. Bemporad and M. Morari, "Robust model predictive control: A survey," in *Robustness in identification and control*. London: Springer London, 1999, pp. 207–226.

[21] S. Dixit, U. Montanaro, S. Fallah, M. Dianati, D. Oxtoby, T. Mizutani, and A. Mouzakitis, "Trajectory planning for autonomous high-speed overtaking using mpc with terminal set constraints," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 1061–1068.

[22] T. Schoels, P. Rutquist, L. Palmieri, A. Zanelli, K. O. Arras, and M. Diehl, "Ciao∗: Mpc-based safe motion planning in predictable dynamic environments," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6555–6562, 2020.

[23] Z. Liu and O. Stursberg, "Recursive feasibility and stability of mpc with time-varying and uncertain state constraints," in *2019 18th European Control Conference (ECC)*. IEEE, 2019, pp. 1766–1771.

[24] M. Kamel, M. Burri, and R. Siegwart, "Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 3463–3469, 2017, 20th IFAC World Congress. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896317313083

[25] R. Gonzalez, M. Fiacchini, T. Alamo, J. L. Guzmán, and F. Rodríguez, "Online robust tube-based mpc for time-varying systems: A practical approach," *International Journal of Control*, vol. 84, no. 6, pp. 1157–1170, 2011.

[26] P. F. Lima, J. Mårtensson, and B. Wahlberg, "Stability conditions for linear time-varying model predictive control in autonomous driving," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 2775–2782.

[27] L. Magni and R. Scattolini, "Stabilizing model predictive control of nonlinear continuous time systems," *Annual Reviews in Control*, vol. 28, no. 1, pp. 1–11, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1367578804000021

[28] A. Völz and K. Graichen, "Stochastic model predictive control of nonlinear continuous-time systems using the unscented transformation," in *2015 European Control Conference (ECC)*. IEEE, 2015, pp. 3365–3370.

[29] J. Berberich, J. Köhler, M. A. Müller, and F. Allgöwer, "Linear tracking mpc for nonlinear systems part i: The model-based case," *arXiv preprint arXiv:2105.08560*, 2021.

[30] M. Basescu and J. Moore, "Direct NMPC for post-stall motion planning with fixed-wing uavs," *CoRR*, vol. abs/2001.11478, 2020. [Online]. Available: https://arxiv.org/abs/2001.11478

[31] A. Mesbah, "Stochastic model predictive control: An overview and perspectives for future research," *IEEE Control Systems Magazine*, vol. 36, no. 6, pp. 30–44, 2016.

[32] N. Ozaki, S. Campagnola, and R. Funase, "Tube stochastic optimal control for nonlinear constrained trajectory optimization problems," *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 4, pp. 645–655, 2020.

[33] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[34] A. Bemporad, C. A. Pascucci, and C. Rocchi, "Hierarchical and hybrid model predictive control of quadcopter air vehicles," *IFAC Proceedings Volumes*, vol. 42, no. 17, pp. 14–19, 2009.

[35] D. Q. Mayne, E. C. Kerrigan, E. Van Wyk, and P. Falugi, "Tube-based robust nonlinear model predictive control," *International journal of robust and nonlinear control*, vol. 21, no. 11, pp. 1341–1353, 2011.

[36] S. V. Rakovic, A. R. Teel, D. Q. Mayne, and A. Astolfi, "Simple robust control invariant tubes for some classes of nonlinear discrete time systems," in *Proceedings of the 45th IEEE Conference on Decision and Control*. IEEE, 2006, pp. 6397–6402.

[37] S. M. LaValle, *Planning algorithms*. Cambridge: Cambridge university press, 2006.

[38] C. Robin and S. Lacroix, "Multi-robot target detection and tracking: taxonomy and survey," *Autonomous Robots*, vol. 40, 04 2016.

[39] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2016.

[40] M. E. Celebi, F. Celiker, and H. A. Kingravi, "On euclidean norm approximations," *Pattern Recognition*, vol. 44, no. 2, pp. 278–283, 2011.

[41] J.-T. Camino, C. Artigues, L. Houssin, and S. Mourgues, "Linearization of euclidean norm dependent inequalities applied to multibeam satellites design," *Computational Optimization and Applications*, vol. 73, pp. 679–705, 2019.

[42] M. Spahn, B. Brito, and J. Alonso-Mora, "Coupled mobile manipulation via trajectory optimization with free space decomposition," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 12 759–12 765.

[43] K. S. Narkhede, A. M. Kulkarni, D. A. Thanki, and I. Poulakakis, "A sequential mpc approach to reactive planning for bipedal robots using safe corridors in highly cluttered environments," *IEEE Robotics and Automation Letters*, 2022.

[44] D. Drew, "Multi-agent systems for search and rescue applications," *Current Robotics Reports*, vol. 2, 06 2021.

[45] C. T. Recchiuto and A. Sgorbissa, "Post-disaster assessment with unmanned aerial vehicles: A survey on practical implementations and research approaches," *Journal of Field Robotics*, vol. 35, no. 4, pp. 459–490, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21756

[46] Z. Wang and H. Gu, "A review of locomotion mechanisms of urban search and rescue robot," *Industrial Robot: An International Journal*, 2007.

[47] R. Dhaouadi and A. A. Hatab, "Dynamic modelling of differential-drive mobile robots using lagrange and newton-euler methodologies: A unified framework," *Advances in Robotics & Automation*, vol. 2, no. 2, pp. 1–7, 2013.

[48] S. Burion, "Human detection for robotic urban search and rescue," Ph.D. dissertation, École Polytechnique Fédérale de Lausanne, 2004.

[49] J. P. Queralta, J. Taipalmaa, B. C. Pullinen, V. K. Sarker, T. N. Gia, H. Tenhunen, M. Gabbouj, J. Raitoharju, and T. Westerlund, "Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision," *IEEE Access*, vol. 8, pp. 191 617–191 643, 2020.

[50] J. Fayyad, M. A. Jaradat, D. Gruyer, and H. Najjaran, "Deep learning sensor fusion for autonomous vehicle perception and localization: A review," *Sensors*, vol. 20, no. 15, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/15/4220

[51] R. C. Luo and C.-C. Chang, "Multisensor fusion and integration: A review on approaches and its applications in mechatronics," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 1, pp. 49–60, 2012.

[52] F. Castanedo, "A review of data fusion techniques," *The scientific world journal*, vol. 2013, 2013.

[53] M. Otte and E. Frazzoli, "RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *The International Journal of Robotics Research*, vol. 35, no. 7, pp. 797–822, 2016.

[54] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[55] P. Yao, H. Wang, and Z. Su, "Real-time path planning of unmanned aerial vehicle for target tracking and obstacle avoidance in complex dynamic environment," *Aerospace Science and Technology*, vol. 47, pp. 269–279, 2015.

[56] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," in *2017 ieee international conference on robotics and automation (icra)*. IEEE, 2017, pp. 1527–1533.

[57] S. Lee, R. Davidson, N. Ohnishi, and C. Scawthorn, "Fire following earthquake—reviewing the state-of-the-art of modeling," *Earthquake spectra*, vol. 24, no. 4, pp. 933–967, 2008.

[58] T. Korhonen and S. Hostikka, "Fire dynamics simulator with evacuation: Fds+ evac," *Technical Reference and User's Guide. VTT Technical Research Centre of Finland*, 2009.

[59] G. Solmaz and D. Turgut, "A survey of human mobility models," *IEEE Access*, vol. 7, pp. 125 711–125 731, 2019.

[60] T. A. Nüssle, A. Kleiner, and M. Brenner, "Approaching urban disaster reality: The resq firesimulator," in *RoboCup 2004: Robot Soccer World Cup VIII 8*. Springer, 2005, pp. 474–482.

[61] R. Scattolini, "Architectures for distributed and hierarchical model predictive control - a review," *Journal of Process Control*, vol. 19, pp. 723–731, 2009.

[62] J. D. Gammell and M. P. Strub, "Asymptotically optimal sampling-based motion planning methods," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, pp. 295–318, 2021.

[63] J. Bruce and M. M. Veloso, "Real-time randomized path planning for robot navigation," in *Robot soccer world cup*. Springer, 2002, pp. 288–295.

[64] M.-S. Yang, C.-Y. Lai, and C.-Y. Lin, "A robust em clustering algorithm for gaussian mixture models," *Pattern Recognition*, vol. 45, no. 11, pp. 3950–3961, 2012.

[65] S. S. Ge and Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method," *Autonomous robots*, vol. 13, pp. 207–222, 2002.

[66] J. L. Garriga and M. Soroush, "Model predictive control tuning methods: A review," *Industrial & Engineering Chemistry Research*, vol. 49, no. 8, pp. 3505–3515, 2010.

[67] X. Wang, J. Liu, H. Peng, X. Qie, X. Zhao, and C. Lu, "A simultaneous planning and control method integrating apf and mpc to solve autonomous navigation for usvs in unknown environments," *Journal of Intelligent & Robotic Systems*, vol. 105, no. 2, p. 36, 2022.

Part II

Preliminary Analysis

*This part has been assessed for the course AE4020 Literature Study.

2

Literature Review

2.1. Introduction

Disasters are devastating to both communities and the areas they live in. Earthquakes alone claimed over a million of lives in the 20th century, while the 2010 earthquake in Haiti totalled around 222,500 deaths [1]. The number of people injured or forcefully displaced due to disasters is higher. Disaster management is the response to these events, where search and rescue (SaR) operations are performed to mitigate any further life losses or injuries. Many lives have been saved through the tireless efforts of SaR teams. However, past events show that the time taken for arrival and starting the operations is one of the main limiting factors for further contribution [2]. Improving the search speed is therefore one of the ways the contribution can be further improved.

In the past 2 decades SaR has become increasingly augmented with robotics [3]. There are many tasks that robots can be developed to undertake, from search and mapping, to the actual rescue itself. One of the key reasons robots are used, is the ability to traverse hazardous environments, and the ability to search for victims and map disaster areas which could otherwise be hard to reach. This not only increases the search speed and search area, but also improves the efficiency of further rescue operations. Many shortcomings and challenges still exist, as the widespread use of search robots has not yet been adopted (see chapter 2 in Murphy's book [3]). For this thesis, two problems will be tackled, namely obstacle avoidance in the dynamic environment, and the autonomous motion planning and control accompanying the search mission.

2.1.1. Project scope

This project's scope encompasses the design of multi-agent software using a hierarchical bi-level model predictive control (MPC), and the possible hardware implementation for a single robot. The simulations are to be done on a multi-agent system under various levels of control, communication, prior environmental knowledge, sensor quality, and number of robots. The mission is victim detection in a SaR scenario.

The research focus for the thesis is the controller design. The sensor, communication, and locomotion subsystems are all of importance to the controller design but they are not the focus. They shall only be developed to the extent that is required to specify the effect and the requirements for the controller.

2.1.2. Project objectives

SaR missions aim to rescue as many people in distress as possible and to limit any further effects of disasters in a limited time frame. Robots are used to expedite this process. Different designs and controllers have been developed for different missions but all serving this purpose. As such, whenever a new controller has been developed, it would be interesting to compare its performance to the established solutions and evaluate possible future developments. The research objective follows this logic:

> The research objective is to develop a distributed hierarchically structured MPC with obstacle avoidance to optimize the search time for victims in an urban search and rescue (uSaR) scenario using a multi-agent SaR system, and to evaluate its performance by comparing it to other solutions.

The main goal of this research is to develop a working multi-agent obstacle avoiding distributed hierarchical MPC-based controller that is compatible with various conditions. The second goal is to

test and compare its performance with other solutions in terms of search efficiency (time use), obstacle avoidance, reliability, and cost. When done with simulations, an optional further goal would be to test the hierarchical MPC using hardware, such that the controller can be compared to both simulated and practical implementations.

First of all, the testing and simulation environment needs to be accurately defined through the mission's description. The first step is thus to understand the environment, challenges, and the system goals and constraints that stem from these definitions. Once that is done, the desired direction of development and suggestions from a SaR team's perspective in conjunction with other works on the topic can be used to establish the simulation and testing of the controller. That is the first sub-goal.

Secondly, the system's robustness to variable conditions and an unsafe environment is of vital importance. Failing to succeed is, at worst, the same as failing to save a life. Designing the robustness for a mission, system, robot, and subsystem level, within the scope of the project, is another sub-goal.

From a design perspective, the multi-agent distributed system is a sub-goal by itself. Regarding this, all system design considerations from task division to perception need to be described, and the parts that fall within the scope of this research must be addressed. Any possible recommendations that stem from this part would impact future work and describe possible extensions to a hierarchical MPC implementation for the uSaR mission.

From a single robot's perspective, there are a multitude of navigation and path-planning algorithms. These methods must be compared to each other, and after a selection, the optimal bi-level MPC combination can be made for victim search efficiency. The other possible algorithms can serve as test-cases for comparing performance. Selecting and implementing the navigation and hierarchical MPC is another sub-goal.

Recognizing all the different sub-goals, a modular design approach has to be taken, where each part can be tested and modified individually. Creating a clear modular structure for ease of testing and prototyping is the final sub-goal.

2.1.3. Research questions

The main objective of this project is to evaluate the performance of a hierarchical MPC for SaR applications and provide future design suggestions for a multi-robot system. From the objective and sub-objectives several research questions follow, which form the direction for both the literature study and the thesis.

1. What are the main considerations for developing a distributed autonomous control system for a multi-robot team in a SaR scenario?
    (a) What are the most important characteristics, requirements, and tasks for the robot team in case of a victim search mission, and how do they affect the design of the control system?
    (b) To what extent do the sensor, communication, and other sub-systems affect the robot motion planning?
    (c) How does a controller achieve a desired performance considering non-homogeneous sensors, levels of communication, and (prior) knowledge for the robots?
    (d) How is robustness of the control system with respect to both mission and sensor uncertainties defined and how can it be guaranteed?
2. How effective is a hierarchical bi-level MPC control system for a multi-robot SaR team in an urban environment?
    (a) What is the effect of real-time control on the controller, and what methods exist for improving computational efficiency to this end?
    (b) How is dynamic obstacle avoidance best implemented in terms of power usage, robustness to obstacle behaviours, and robustness to uncertainties?
    (c) How does the bi-level MPC compare to other controllers in terms of power usage, robustness, and real-time performance, for both the mission and obstacle avoidance?
3. How can the controller best be implemented and evaluated, and with what methods and accuracy can it be verified or validated?

(a) What approaches for model and controller simulation exist, and to what extent do they need to be done?

(b) What are the metrics and scenarios to evaluate the performance of the controller and its implementation?

(c) What is a hardware test required for and how can it be used to improve the controller?

2.1.4. Literature study structure

The literature study finds research to support a thesis which can answer the research questions and fulfil the research objectives. Firstly, in Chapter 2.2 the mission itself is further defined through SaR literature specific for robots and the environment. Subsequently, in Chapter 2.3 various perception and motion planning designs are listed, and evaluated to an extent. Further design choices such as communication and fleet design are also considered, such that the general controller architecture can be formulated. Chapter 2.4 focuses on MPC literature to define the theory and variations for the specific implementation of the hierarchical MPC. Furthermore, several tuning methods for MPC are presented alongside a development plan for the thesis.

2.2. Search and rescue

In the event of a disaster, the first step is to organize the response. When people could be in distress, search and rescue (SaR) operations will take place with a search for victims. Since the 9/11 disaster, robots started to be used for SaR operations [3]. Throughout the years search and rescue has been increasingly augmented with robotics to improve the search speed, to map inaccessible or dangerous areas, or even to perform direct intervention such as opening doors or blocking oil pipes [3].

Whatever the environment (be it collapsed structures or the wilderness) there are hostile elements and limited sources of information. Secondly, a unanimous truth is that the quicker the response, the greater the chance for survival, and the quicker the recovery, see Figure 2.1.

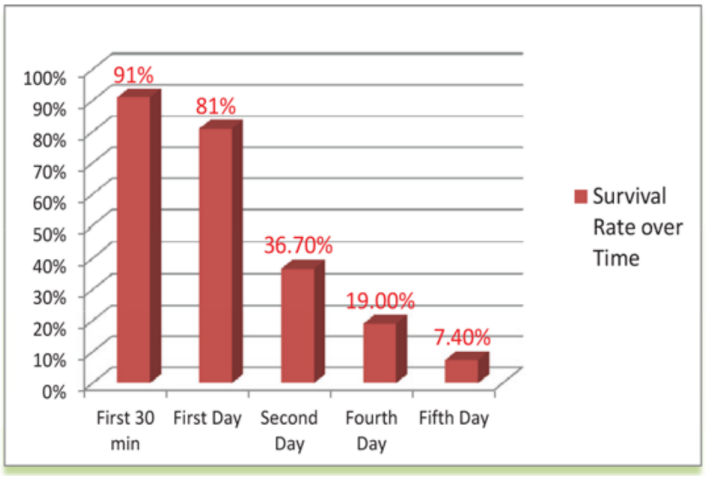


Figure 2.1: Survival rate over time for victims who survived the initial effects of an earthquake. [4]

In order to design for the multi-faceted nature of search missions, each aspect will be addressed separately. Firstly, Part 2.2.1 reviews some examples and literature on how robotics and SaR have cooperated so far, to obtain an overview of how SaR robotics were developed, and to assist in defining the mission. Having an initial overview, Part 2.2.2 uses the environment to define several SaR sub-types. Part 2.2.3 then uses the mission taxonomy, and combines it with the environmental definitions to specify the scope of the mission.

2.2.1. Previous use-cases

Reviewing the possible missions for SaR robotics, applications range from direct intervention (such as the case of inserting a tube and regulators in the Deepwater Horizon spill of 2010), to a search mission (to

reach areas rescue workers cannot easily reach, or to expedite the process) [3]. As far as the thesis is concerned, the controller design and literature will be performed for the search mission, which so far has mostly been done to locate miners in collapsed caves [3].

Past uses of unmanned aerial vehicles (UAVs) include assessing the damage and assisting reconstruction efforts. Examples include FINDER with UAVs in the 2015 earthquake in Nepal [5], and the post-disaster assessment after the 7.0 earthquake in Lushan as done by Qi et al [4]. Qi et al also summarized the previous work of UAV support teams designed for various environments, such as the case with Pratt et al [6] who analysed the success of a post-disaster assessment of hurricane Katrina using a UAV.

Most literature on this topic pointed towards the use of UAVs as scouts, as they have the greatest flexibility in motion, and can provide a birds-eye view to the operator [7]. Another point of interest is that there is some success with both simple map-less algorithms (such as bug algorithms or pre-trained deep-learned networks), and complex sensor-fused map-using algorithms (such as decentralized visual simultaneous localization and mapping (SLAM), or SLAM with ML-RANSAC) algorithms ([8], [9], [10]).

2.2.2. Search and rescue environment

The robot applications in SaR depend on the environment. The type of disaster and the environment define the mission and constraints at both an agent and system level, changing the complexity of the search mission, and the subsequent agent design.

Grogan, Gamache, and Pellerin [8] described the variety of environments as uSaR, wilderness SaR, air-sea rescue, and combat SaR. These are SaR missions in an urban, wilderness (including mountains and caves), maritime, and combat environments, respectively. USaR occurs in urban areas where the search mission is to find an unknown number of stationary victims, in a somewhat localized environment. Wilderness SaR and air-sea rescue, on the other hand, deal with a known number of mobile victims in an open-ended search location. [8]

Two examples will be given to illustrate the impact of the environment:

1. Air-sea rescue usually requires a UAV to sweep over a large area of water, find men overboard, and rescue them with either seafaring drone support or through human intervention. An autonomous solution can exist in the form of E.M.I.L.Y., with an overhead robot (UAV) as a spotter [11]. There are other solutions, but that study is beyond the scope of this project.
2. On the other hand, uSaR has dynamic obstacles, danger zones such as rooms at risk of fire or further collapse, and a constrained flight space. It is even possible for the robot to get lost before being able to return and communicate its findings. A successful semi-autonomous example is the quadcopter UAV solution by Qi et al developed for the post-disaster assessment of the 2013 Lushan earthquake [4].

Compared to air-sea rescue, uSaR requires higher mobility, but lower endurance on its flight hardware. Secondly, uSaR requires greater situational and spatial awareness, but lower resolution sensors (among other differences such as with its control logic, see Part 2.2.3). Every environment has its own set of peculiarities. Moreover, the SaR environment is further defined by the disaster affecting it.

Another very important aspect is that due to all the rubble and obstructions to the line-of-sight in indoor environments, uSaR applications tend to be global navigation satellite systems (GNSS) denied environments [3]. The rubble also tends to affect remote control and data streams, suggesting the need for non-centralized control. This will have consequences for the perception system in Part 2.3.1, communication in Part 2.3.3, and several topics beyond the scope of the project (such as the human-robot interaction, and possible formation constraints due to tasks such as daisy-chaining).

Characteristics and risks

Any type of disaster, be it of natural (flood, hurricane, or earthquake) or technical origins (explosion, gas leak, sinkhole, etc.), has its own set of latent and active risks and size of impact. These change the optimal system's design and control. For the purpose of this work a disaster encompassing collapsed buildings is selected, with risks of fires and explosions, and a multitude of both dynamic and static obstacles.

Unfortunately there is no exhaustive list of environmental characteristics and risks made for a robotic system in an uSaR environment. Rosique et al. did mention the distinction of the environment either influencing the sensors or the physics while discussing simulators [12].

For a collapsed building disaster various environmental variables and risks must be modeled and designed for, such as dust, rubble, and possible fires. These can be categorized as influences on the sensors, motion, or the mission:

- Sensor-related influences: There are two types, those who affect the ability to measure, and those which can lead to false positives/negatives. From the first type examples include fog, illumination, dust, fire, and obstacles. From the second type, examples include ambient noise, cluttered environment, and warm objects. These depend on the features detected by the sensors.
- Motion-related influences: Examples include rain, wind, loose sediment, and obstacles. There are types which must be avoided (obstacles) and types which cannot (rain/wind).
- Mission-related influences: Examples include fire, possible air quality measurements, and obstacles. These must be separately addressed through the robot's control logic with prioritization. In the event of the loss of an agent, the fleet's subsequent search operations are affected.

For example, consider an uSaR mission with a mission-related influence of fire. The fire is present in a stairway connecting floors, while victims are possibly stuck in rooms. The robot search team must perform the search for victims, despite possible obstacles such as closed doors. The robots must therefore quickly traverse the stairway and check for a possible victim in the rooms (despite the possibility of not being able to enter all rooms), knowing some robots could possibly become inoperable.

Will losing a robot at the start of the search mission lead to more possible casualties, or more specifically what are the consequences of agent-loss? Secondly, which areas should the controller prioritize? These are some of the concerns for fleet and controller design. Although the example is both complex and highly time-constrained, the controller must be designed to account for it.

2.2.3. Mission taxonomy

As mentioned in the work of Robin and Lacroix [13], the taxonomy clearly defines the scope of the mission. By clarifying the problem, the best method to obtain the solution becomes apparent.

For uSaR, the mission defined by Grogan, Gamache, and Pellerin [8] concerns finding immobile (or trapped) victims, with no knowledge on the number of victims. Due to the collapse of structures (roofs/walls/entrances) or the displacement/destruction of objects (fallen wardrobes), any given map cannot be assumed to be fully accurate. Sometimes a map can even be unavailable due to security or privacy concerns.

The robotic support team can assist/perform the search through one of three specific missions. The missions could be search for victims, mapping of a disaster-struck area, or tracking and surveillance [8]. The work in this thesis will focus on the search for victims. A search (or target detection) does not require mapping.

Classifying it as a search mission, Robin and Lacroix [13] define uSaR as a target detection problem with mobile search, the same as Grogan, Gamache, and Pellerin. A target is an entity (e.g. victim or object) that can be localised within the environment. There are three sub-types of missions then defined based on the information and resources that are available: *capture*, *probabilistic search*, and *hunting*.

Each sub-type has a variety of control algorithms made. The algorithms can be selected based on the system's degree of (de-)centralization, cooperation, stochasticity (considering uncertainties), and whether the pathing is done using optimization (solved on-the-go) or planning (solved off-line).

Target detection - Planning

If the map is known, and there are sufficient robots and time for full coverage then the problem becomes a *capture*, which simplifies to a path optimization or graph clearing task. As it stands, most literature defined in the work of Grogan et al. [8] uses a *capture* problem. Either graph-based, or set covering methods are used. In case of an outdated map (or new information), task allocation is mentioned as a remedy. Robin and Lacroix [13] also mentioned the scalability problems of using a centralized system and why decentralized cooperation algorithms perform better. In essence, with *capture* there is a clear worst-case scenario which can be optimized for before-hand.

When there is a lack of resources (robots or time), *probabilistic search* is more applicable. Here the "worst-case guarantee" cannot or is not fulfilled. *Probabilistic search* assumes SaR would have non-adversarial targets, drastically decreasing the complexity of the problem. Here, Markovian models (MDP)

(specifically partially observable Markov decision process (POMDP)) are mentioned as suitable, but as the targets in uSaR are mostly immobile, particle filters were also mentioned as suitable. Particle filters perform better regarding scalability and can subsequently be used for tracking if the mission requires it [13].

Target detection - Optimization
So far the target detection problem, or rather its solutions, were solved as a planning problem, where the path is pre-planned. Formulating it as an optimization problem means optimization is performed on-the-go, leading to a globally sub-optimal solution. Note that this is more compatible with the dynamic nature of SaR, where new information and developments could occur at any time.

By definition of having full information, a planning approach would always perform better than optimization. *Probabilistic search* on the other hand, can benefit from this. Gradient-based optimization, or a two-step approach using rough planning and a detailed optimizer can be used. Multiple criteria decision making (MCDM), in particular stands out as a good choice [13].

Lastly, in case of *hunting* there is not only a lack of resources but also a major lack of information. For *hunting*, both the map and number of targets are unknown. In the absence of a map, and/or due to lack of resources (bad sensors, small number of robots, limited time, etc.), a pathing problem degenerates into a hunting problem. The robots could collaborate through reactive (react to obstacles/environment) random search, map-less graph-clearing through local frontiers, or through emergent behaviour (e.g. from ”anti-”flocking), as summarized in [13].

Initial design choices
Map generation and localization are not necessarily required. Numerous solutions avoiding mapping were listed in [13] and [10]. Opting to avoid a map reduces the computational load on the controller, allowing for a cheaper single robot. This choice does incur additional design considerations which have to be addressed in the form of the system’s robustness, through testing and simulations.

As it stands, both MCDM and a two stage controller seem to be a good choice for uSaR, as well as a simple emergent agent-based hunting method. These will be further considered on how to be implemented as MPC in Chapter 2.3.

2.3. System design
As identified by Drew [7], even though publications on multi-agent systems has over tripled on a per-year basis since 2010, the number of papers on multi-agent systems for SaR has remained about the same. There are four challenge areas he identified: Perception and planning, communication, human-robot interaction, and cost/function trade-off.

In this chapter both the single and multi-agent aspects of the robot’s design will be looked at. Using Drew’s work as a reference, the chapter is structured similarly. First, in Part 2.3.1 the sensor subsystem is considered through victim detection sensors, navigation sensors, some perception algorithms, and sensor fusion algorithms. Part 2.3.2 discusses various path-planning and optimization based on the target detection mission. Communication can have a strong influence on motion planning, and is therefore covered to some extent in Part 2.3.3. Any other aspects beyond the scope of the project are listed in Part 2.3.4. All system-level and robot-level decisions which can be made at this point will be presented in Part 2.3.5. Lastly, simulator types and considerations for simulation will be addressed in Part 2.3.6.

2.3.1. Perception
Sensors are vital for positive identification of the target, path-finding, and obstacle avoidance. There are numerous surveys comparing sensors for each task, each having different rankings, but there is no clear ”optimal” choice. This is why a trade-off must be performed based on individual task performance, synergies with other subsystems and sensors, and resource usage, which will be done in Part 2.3.5.

It should be understood that the detailed design of the sensor subsystem falls beyond the scope of this project. However, the extent to which it is done is necessary to establish the possible means by which the navigation and victim detection will be done, such that some uncertainties can be bounded (by looking at the errors in navigational algorithms, or victim identification accuracy), and the cost of an agent (in

case of agent-loss) can be determined. Besides that, it will be required to select sensors for the hardware implementation.

Victim detection sensors
Victim detection is defined by how good and reliable the sensors (and interpretation) are. Sensors can be used to detect the heartbeat, voice, temperature, texture, motion, scent, color, oxygen levels, and the shape of the victim. The sensors can be visual, acoustic, thermal, ultrasonic, seismic, or chemical ([14], [15]). A selection were compared in Burion's work [14], see Table 2.1. The laser rangefinders mentioned

Table 2.1: Comparison of select features of different sensors used in victim detection [14].

	Technology	Feature detected	External size	Cost	Human distinction	Strengths	Weaknesses
Linear camera	CCD/CMOS EM 0.4-1.1 μm	vision	-	-	-	price	low resolution
USB camera	CCD/CMOS EM 0.4-1.1 μm	vision	+	+	++	cost / performance	low resolution
Stereo vision	CCD/CMOS EM 0.4-1.1 μm	vision/ distance	++	++	++	vision, distance info	computational cost
Laser rangefinder	ToF/ triangulation EM 620 - 820 nm	distance	++	+++	-	measurement precision	price
Radar	ToF EM 5 - 24 GHz	distance	+	+++	-	precision with big range	price
Ultrasonic sensor	Membrane SW 130 - 290 kHz	distance	-	-	-	price	echo sensitivity
Microphone	Membrane SW 100Hz - 16 kHz	sound	-	-	+	price	noise sensitivity
Infrared camera	CCD/CMOS EM 7-14 μm	heat	++	++	+++	human distinction	price
Pyroelectric	Crystalline sensor EM 7-14 μm	body heat	-	-	++	price, human distinction	only motion detection
Thermopile	Thermocouple EM 5.5 - 13 μm -25 - 100 °C	heat	-	-	+	price	noise sensitivity
CO_2 sensor	Electrochemical	gas	++	++	++	human distinction	too directional
SpO2	Light absorption (650nm, 805nm)	blood oxygen/ pulse rate	-	N/A	+++	human distinction	not available for robotics

EM = Electromagnetic waves; SW = Sound waves; ToF = Time of Flight

in Burion's work, are a family of sensors that contains the subset of Light detection and ranging (LiDAR). Rosique et al. [12] further compared a selection of sensors under various criteria such as resolution or durability. Additionally, ToF cameras were mentioned which perform ranging using the phase difference of reflected near-infrared light to the original light. However, neither Rosique et al. nor Burion mentioned the power use of the sensors, which would also be of interest in lightweight robot designs.

Stepanova et al. [16] specifically mentioned acoustic sensors to identify knocking behind closed doors or collapsed walls, and that a robot should "broadcast the message that it is there on the rescue mission to the potential survivors to encourage them to start signalling that they need help." They also suggest that the overview provided by drones with visual feeds speeds up the initial search process.

From Table 2.1, the best solution is a combination of sensors that detect different features (such as thermal and visual) such that positive victim detection is highly probable. Even under circumstances which could affect the feature detection of a sensor such as low illumination, a mixed-sensor approach could still reliably identify victims through other features. The main downside is the added complexity of sensor fusion (see Part 2.3.1).

Another, cheaper, less-energy consuming solution comes in the form of deep-learned neural networks. In the past few years, progress has been made with both real-time and relatively small-sized (54 megabytes [17]) neural networks for victim identification. These methods boast mean accuracy precisions of over 70% (a metric for quality of object detection algorithms), with the one in by Martinez-Alpiste et al. [17] at 95%. YOLOv3 is another very popular contender, often used due to its high quality real time classification [18].

One sensor that has not been mentioned in Table 2.1 are ground penetrating radars. Though mostly used in post-avalanche SaR scenarios, they can also be applied to uSaR [19]. This way buried victims can be detected, who might otherwise be missed. The drawbacks are the high price, power use (2-4 Watts for the sensor itself), and weight (about 1kg) [20]. There are two ways to use this system, however. The first is to use it directly on every agent, which is too costly. The other is to design a separate robot that performs the ground scan, collaboratively setting goals with the other agents.

To conclude, Stepanova et al. suggest both a visual and acoustic sensor, while Rosique et al. and Burion both suggest the infrared camera to be included in a sensor fused subsystem. The best for victim detection would therefore be a combination of these three, but when limited to only one sensor, a visual camera is the sensor of choice. If a swarm approach is intended, the lowest-cost solution would take priority in the form of a possibly (inference) sensor-fused pyroelectric sensor and cheap USB camera. Secondly, in case of a UAV, the external size, power use, and weight of the sensor subsystem is limiting the flight time and thereby influencing the robot design.

Navigation sensors

From the point of view of navigation, sensors are needed to detect the robot's position or path relative to a starting point, a reference signal, etc. Rosique et al. [12] separated sensors into two groups. Those that are used for environmental perception and those used in position estimation. For uSaR the sensors can be better distinguished as *physical*, and *signal tracking*.

Physical sensors are inertial measurement units (IMUs), cameras, magnetometers, and similar sensors, which do not necessarily require an external signal. These sensors measure the physical state of the world or the robot's motion through inertial means. The algorithms used in navigation are based on dead reckoning, normal or visual odometry, referencing a map or objects (can be self-made using SLAM), or referencing a set of tabulated values. Other sensors, such as proximity sensors (which work by measuring reflected infrared signals), only measure if there is an object, without providing any state estimation. Very cheap, and useful for detecting proximate objects, but nothing else.

On the other hand, *signal tracking* sensors include GNSS receivers and other antennas. These rely on the capability to communicate with a foreign signal (not made by the robot itself), or only to receive the reference signal. The methods used to establish position and heading are usually based on either the signal's strength, time-of-arrival, frequency difference-of-arrival, carrier phase, or angle-of-arrival [21].

The benefit of a signal tracking sensor is a comparatively low-energy, high-accuracy, low-cost sensor subsystem that can define the robot position with respect to a coordinate system. Moreover, it does not run into closed-loop problems such as positional drift (accumulated error over time). The drawback is that it relies on the availability of the reference signal. In practice, a combination is often used, such as an IMU in combination with a GNSS receiver in car navigation systems.

A simple GNSS signal, though tempting, is not necessarily sufficient. As mentioned by Breßler et al. [22], in dense urban and indoor environments, the non-line-of-sight GNSS rarely produces sub-meter-level positioning. Possible solutions exist in the form of sensor fusion (for example, GNSS/IMU), a high-accuracy (expensive) base station, or using additional signals.

Most surveys regarding sensor use in uSaR (including [22], [8], [16], and [15]) show interest in a wireless sensor which aims to detect WiFi, Bluetooth, and cellular signals for improving positioning and victim detection performance. It is low-cost, lightweight, and possibly highly accurate. In uSaR scenarios this is due to the many different possible reference signals, despite the obstructions of rubble and buildings. It does run into the same problem as the other signal tracking sensors regarding signal availability, where the availability could be affected by scenarios such as an empty battery, or disabled power lines. If used for victim detection it assumes that a signal implies a victim, and no signal implies no victim. This could be a false assumption. Navigation-wise, for a swarm-based approach, combining the GNSS/IMU with the wireless signal sniffer would be a cheap, yet sufficiently accurate (sub-meter) solution.

One last note specific for the navigation aspect of the perception system is that the perception range limits the speed of the agent. Consider the following: if an agent requires 2 seconds to reach a full stop and the sensor range is 20 meters, then the theoretical maximum allowable speed of the agent is $20/2 = 10$m/s. If it is any quicker, the agent would bump or crash into an obstacle before it can perform the avoidance.

Sensor fusion
Burion [14] reported that the sensor fused microphone, visual camera, and thermal camera has the best results in his constructed uSaR environment. It was an indoor environment with mirrors, windows, plastic sheets, noise, knocking, computer screens, heaters, as well as various levels of illumination. It simulated a possible gas leak or post-earthquake scenario. He also concluded that data fusion in general reduces the amount of false positives. There are several ways to perform sensor fusion, however.

Sensor fusion methods are all rooted in statistics. There are many different ways to classify them and criteria to measure them by. If interested in the possible classifications and design strategies thereof see Castanedo's work [23]. Sensor fusion can be done low-level (directly combining the sensor data for more accurate data), medium-level (combine results to establish features for classification), or high-level (combine symbolic representations for a more accurate decision). A classification of various algorithms is presented in [24], see Table 2.2.

Table 2.2: Classification of common fusion algorithms (adapted from [24]).

Low level fusion		Medium level fusion	High level fusion
Estimation methods		Classification methods	Inference methods
Recursive: • Kalman filter • Extended Kalman filter Non-recursive: • Weighted average • Maximum likelihood	Covariance-based: • Cross covariance • Covariance intersection • Covariance union	• Parametric templates • Cluster analysis • K-means clustering • Learning vector quantization • Kohonen feature map • Artificial neural network • Support vector machines	• Bayesian inference • Particle filters • Dempster-Shafer theory • Expert system • Fuzzy logic

Out of these methods, state estimation techniques are of interest to reduce navigation or victim identification errors. The higher-level fusion methods could be used for the goal searching and pathing, but that is beyond the scope of the sensor subsystem. But the low-level fusion methods are designed with different applications in mind.

For a swarm approach, the state estimation and victim detection can use a *maximum likelihood* or *weighted average* approach due to the computational constraints. A swarm can gather many signals (through co-localization), for which the function $\tilde{x}(k) = \arg\max_p (z|x)$ can be used to determine the estimated state ($\tilde{x}$), using a probability density function dependent on k previous observations ($z = (z(1), ..., z(k))$). The other methods require a bit more of an explanation.

For a multi-agent approach with less restrictive computational limits, the unscented or extended Kalman filters or particle filters are of interest. The covariance methods are only useful in case of a distributed or centralized system, explained in Part 2.4.3, where hierarchies are reliant on communication. The Kalman and particle filters can be implemented decentralized (without explicit communication), which is why they will be focused on.

A Kalman filter estimates the state x based on a measured and predicted state, essentially: $x_{\text{estimated}} = x_{\text{predicted}} + K(z_{\text{measured}} - z_{\text{predicted}})$. The Kalman filter uses the Kalman gain K, which is a measure of trust between the prediction and measured state: The lower the measurement uncertainty is compared to the prediction uncertainty, the higher the gain K becomes. It is determined (Equation 2.1c) using the prediction covariance matrix P, observation matrix H, and the sensor noise covariance matrix R.

See Equation 2.1 for a single iteration of a linear Kalman filter.

$$\hat{x}_{k+1,k} = \Phi_{k+1,k}\hat{x}_{k,k} + \Psi_{k+1,k}u_k \tag{2.1a}$$

$$P_{k+1,k} = \Phi_{k+1,k}P_{k,k}\Phi_{k+1,k}^T + \Gamma_{k+1,k}Q_{d,k}\Gamma_{k+1,k}^T \tag{2.1b}$$

$$K_{k+1} = P_{k+1,k}H_{k+1}^T\left(H_{k+1}P_{k+1,k}H_{k+1}^T + R_{k+1}\right)^{-1} \tag{2.1c}$$

$$\hat{x}_{k+1,k+1} = \hat{x}_{k+1,k} + K_{k+1}\left(z_{k+1} - H_{k+1}\hat{x}_{k+1,k}\right) \tag{2.1d}$$

$$P_{k+1,k+1} = \left(I - K_{k+1}H_{k+1}\right)P_{k+1,k} \tag{2.1e}$$

Where Φ is the state transition matrix, Ψ is the input distribution matrix, and Γ is the noise input matrix. These are obtained by discretizing the continuous F (state transition) and B (input) matrices, and the continuous process noise input matrix G. u is the input, and w (process noise) and v (sensor noise) are two Gaussian random noises with zero mean and covariance matrices Q and R, respectively. In case of non-linear models, there are two common solutions.

To deal with non-linearities, the unscented Kalman filter is based on sampling points around the mean, and propagating these through non-linear models such that the covariance of the estimations can be obtained, while the extended Kalman filter uses the Jacobian (computationally expensive). Kalman filters assume full knowledge of the model (Φ, Ψ, H), and noise matrices (Γ, Q, R).

Particle filters, similar to the Kalman filters, have a prediction and update step. They essentially work by first randomly sampling a set of N points, which contain the state information. Each sampled point (or particle) gets a future state prediction based on their previous state, a state transition matrix F, and a noise distribution. Each particle has its weight calculated by predicting their observation z, and computing the likelihood based on that. At that point, the points with the lowest weights get removed, and the state is adjusted using the mean of the remaining particles. Particle filters perform better than Kalman filters when there are non-linear dependencies and non-Gaussian noise, but a large number of particles are required for a precise estimator [23].

Knowing the theory behind sensor fusion approaches makes it possible to use either in case of merging different signals from different sensors. The most common solution used for uSaR UAV search teams is a simple RGB camera, not considering sensor fusion with inertial or GNSS units [25]. Each combination of sensors, perception algorithms, and possible sensor fusion has its own strengths and weaknesses, so it ends up being a trade-off. A greater level of sensor fusion can lead to higher accuracy at a higher cost. A more accurate perception algorithm is often traded off against a higher computational burden.

If heavily constrained by financial cost the choice is a cheap camera and IMU using visual odometry or SLAM for navigation, and a neural network for victim detection. When heavily limited by computational cost, a pyroelectric sensor with a camera should be used for victim detection, while navigation can be performed primarily through GNSS/IMU with proximity sensors. For swarms the choice can be co-localization and wireless sniffing for localization, using proximity sensors for additional obstacle avoidance, and a pyroelectric sensor with a microphone for victim detection. For the most sophisticated multi-agent team that is only concerned with the accuracy, the choice rests on a sensor fused GNSS/IMU using map- or feature-based matching when GNSS is denied, using a visual camera, infrared camera, and a microphone for additional victim detection accuracy.

The uSaR application in mind falls between the extreme cases. To address possible GNSS-denied environments, a wireless sniffer could be used, supplementing an IMU/visual camera combination for better localization using visual odometry, SLAM, or feature matching. If a high-level of localization is not required then the sensor subsystem would primarily be used for obstacle avoidance and victim detection. Optional improvements follow: Using a LiDAR for computational reduction (no depth estimation required) and improved navigational accuracy. An infrared camera could improve feature detection and obstacle avoidance in low-light environments, while also improving victim identification accuracy. A microphone can be added to further improve victim detection accuracy, and finding potential visually obstructed victims.

2.3.2. Vehicle motion planning

Vehicle motion planning is done by acquiring the environmental data (such as obstacles), the vehicle's trajectory, and direction, and combining those to decide on a path towards the goal. As stated in Part 2.2.3, from a mission perspective (target detection), the algorithms that fit the mission the best are MCDM,

a two-stage optimizing controller, or a reactive swarm method. Though the algorithms identified in Part 2.2.3 can be used directly, they are not necessarily optimal or sufficient for the dynamic environment. This part will evaluate the motion planning and obstacle avoidance algorithms in more detail using literature to that end. Specifically, a review on motion planning algorithms in dynamic environments [26], real-time applications [27], and specific for UAVs [28] are used for further reading.

Clear terminology exists within the field of motion planning, which is used to unambiguously describe an algorithm's functioning. The following definitions are mostly taken from Goerzen, Kong, and Mettler [28]. The vehicle (UAV) exists in a physical space known as the *world space*. The vehicle has a certain position and orientation, which would be its *configuration*. The set of all possible configurations in the world space is the *configuration space*. In the case of dynamic objects, the dynamics are also of high interest. The *state* includes the rates of change of the configuration, and possible further derivatives. In much the same way the set of all possible states is the *state space*.

Regarding the world there is the *free space* where a vehicle can exist, and the *obstacle space* where it cannot. A *path* is a curve traced by the vehicle in the configuration space, and a *trajectory* is a path including the time along the path. *Motion planning* is either path or trajectory planning between an *initial* state or configuration to a *goal* state or configuration ([28], [27], [26]). See Figure 2.2 for some clarification, if needed.

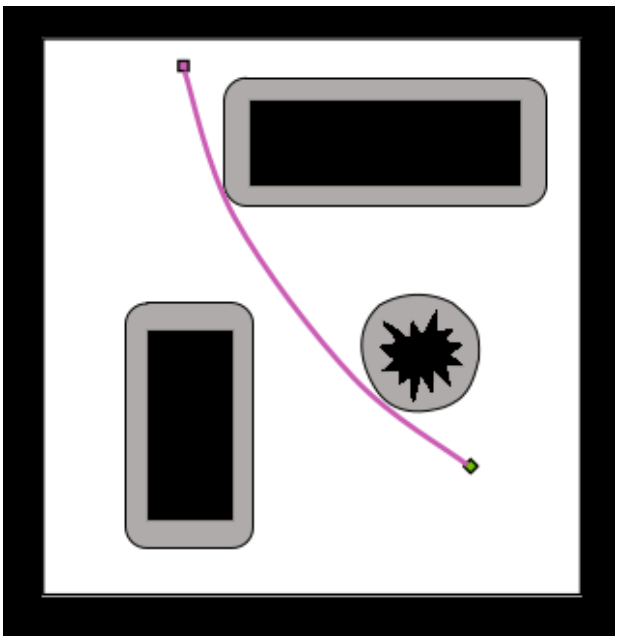


Figure 2.2: Point-vehicle representation [28]. Obstacle space in black, configuration space in white, free space in white + grey.

Motion planning algorithms are *complete* if they find a path when one exists, and return no path if no collision-free path exists. A planner is *optimal* when it returns a path that is optimal according to some criteria. Lastly, a *sound* planner guarantees that despite the uncertainties in sensing and control, the vehicle will enter the goal region without colliding with an obstacle. Planners that are *resolution complete/optimal*, only find a solution if the problem is sufficiently discretized. Insufficient discretization might paint a region of free space as occupied, and exclude that from the configuration space. For *probabilistically* complete/optimal if a solution exists, it will be found with the right conditions or randomness, which given infinite time to a random number generator will succeed [28].

Path planning

There are many different solutions for both motion planning and dynamic obstacle avoidance. Thankfully the scope can be limited. Goerzen, Kong, and Mettler [28] summarized that excluding the equations of motion lead to worse performing optimization for pathing, especially in the case of a vehicle with many degrees of

freedom such as an aircraft. Secondly, considering the uncertainties and the dynamic environment, using an optimization algorithm is a better fit compared to planning. This also agrees with Robin and Lacroix's [13] summary.

The vehicle can either be modeled as a rigid vehicle, or it can be bounded by a radius. Bounding the vehicle by a ball allows it to be reduced to a point for obstacle avoidance. The configuration space is modified by expanding the obstacles by the radius of the bounding ball. It is a conservative simplification, reducing the calculation time, problem complexity, maximum optimization of the solution, and possibly the feasibility. Regarding the feasibility: a piece of A4 (297x210x0.1mm) paper can fit in more places than its bounded ball size (a diameter of $\sqrt{297^2 + 210^2} = 364$mm).

Uncertainties remain the same despite such simplifications. Motion planning in general struggles with uncertainties, as there are no sound or complete solutions when characterizing uncertainty with an infinite-tail [28]. It can be understood that bounded uncertainties can still be worked with (practical applications exist), though they may not be sound. A comparison of various solutions that are not differentially constrained are listed, with information taken from Goerzen, Kong, and Mettler [28] unless otherwise cited.

- Roadmap methods reduce the problem to a graph search on which Dijkstra's algorithm or A* can run. All methods assume a point vehicle. A *visibility graph* connects each vertex of the polygonal or polyhedral obstacles (2D or 3D) to other obstacle vertices. The optimal path is then found by following the shortest path that grazes the obstacles. It is complete and optimal (resolution optimal for 3D). The *Voronoi roadmap* builds a skeleton that is maximally distant from obstacles, which is sub-optimal (or optimal if maximum margins are intended), but complete.
- Exact cell decomposition, decomposes the configuration space into small convex polygons. *Trapezoidal decomposition* reduces the free space to many trapezoids, connecting the centre points, and then using graph search to find a path. *Critical curve based decomposition* (and *cylindrical algebraic decomposition* for 3D) can work with rigid vehicles by taking into account the turning motion, such that the path consists of piecewise polynomial curves. These solutions are complete, and most can work with rigid vehicles, but they are non-optimal.
- Approximate cell decomposition methods are all resolution complete, but non-optimal. *Rectanguloid cell decomposition* works similar to trapezoidal decomposition, but instead divides the configuration space in rectanguloid regions, labelling the cells as empty (free space only), full (obstacle only), or mixed (both). *Approximate and decompose* aims to reduce the volume of mixed cells, but maintain a sufficiently small number for computational speed. 2^m *tree (octree) decomposition* aims to reduce the number of points required to represent obstacles compared to full grid solutions; it has been demonstrated on a helicopter UAV [29].
- Potential field methods work through assigning the free space a potential function, giving the goal the lowest potential, while obstacles are given a maximum value. They have low computational complexity, and are sometimes implemented for swarms. *Virtual force field* uses distance-decaying functions which peak at the obstacle regions, such that the vehicle safely moves by 'descending' to a lower potential. It is neither optimal nor complete (due to possible local minima). *Potential field guided search* uses a probabilistically or resolution complete search, to avoid local minima (such as A*). *Harmonic potential functions* solve the local minimum by using smooth partial differential equations (one minimum), thus being complete, and has been proven to work real-time on a rotorcraft UAV [30]. Other methods include *wavefront expansions* which essentially run Dijkstra's algorithm for the navigation function.
- Probabilistic approaches are non-optimal, and, for instance, modify the potential field approach by using a random walk, avoiding a local minimum. The *probabilistic roadmap* (PRM) tests if random samples of the configuration space belong to the free space, and then uses graph search to find a probabilistically complete path. There is also a probabilistically optimal version called the PRM* [31].
- Weighted region approaches modify the configuration space with a weight function. Using *polygonal weighted regions* the approach can either be done using Snell's law of refraction and finding an optimal path, or an approximation thereof which is resolution optimal. To extend this in higher dimensions a *weighted grid approach* can be used, reducing it to a discrete search over a grid, which is both resolution complete and resolution optimal.

Most of these methods are not explicitly designed to take into account the limited sensor range or lack of a map, assuming a *capture* problem (see Part 2.2.3, or Robin and Lacroix [13]). They can be

adapted, as these are rather the theoretical baseline. Secondly, when defining path planning for UAVs and other vehicles the differential constraints are of interest, where even in a trivial case of an obstacle-free configuration space, connecting two states in 3D is not generally exact [28]. To address these differential constraint approaches, Goerzen, Kong, and Mettler [28] compared sampling-based, minimum distance, mathematical programming, and potential-based approaches.

- Sampling-based approaches are usually resolution complete, resolution optimal, and resolution sound, but tend to be too slow to be used in real-time due to the high dimensionality of the state space. *State space lattice search* discretizes the search by mapping the state space of the vehicle as a lattice, and searching for a time-optimal path therein. *State-space navigation with interpolation* approximates the time-optimal path by returning a navigation function, which can be optimized through value iteration or with a control policy, but it runs into the same problem with higher order dimensions. A *reachability graph* looks into possible states, which quickly runs into high combinatorial complexity. The only feasible real-time sampling method mentioned is the *rapidly-expanding random tree* (RRT), a probabilistically complete approach, which works by randomly sampling the configuration space for a feasible trajectory. There is an expansion called RRT* which is also probabilistically optimal [31].
- Decoupled trajectory planning methods are generally non-optimal, but resolution complete, and resolution sound. More importantly, these can usually be implemented real-time. They work by first finding a path through the discretized configuration space (for instance through a probabilistic roadmap), setting waypoints, and then generating a trajectory for a vehicle based on that path. The *two-step approach* is the simplest form. A modified form that is commonly used for UAV applications is the *hierarchical decoupled planning and control*, but both of these methods have no general proof of completeness, soundness, or optimality. Besides this there is the *discrete configuration space search*, which for instance produces a set of ordered waypoints through which a spline is fitted through (adhering to acceleration constraints). The spline method's trajectory can then be used for speed optimization for a minimum-time solution. Last of this type is using *2-D Voronoi solutions using multiple planes* differentiated by a single angle which include initial and goal point, which has also been proven to work in a multi-level controlled rotorcraft.
- Maneuver automaton or control quanta use a discrete set of motion primitives to find an optimal path. These primitives are composed of the trim trajectories (equilibrium motions), and maneuvers (non-equilibrium transitions, such as braking or a U-turn with a car, or a simple banked turn to a hammerhead for aircraft). Any states between these pre-defined motions are interpolated. As can be expected, the more agile and controllable a vehicle is, the more maneuvers and trim states need to be recorded. The difference between these methods is that whereas the maneuver automaton chooses from a set of motion primitives, the control quanta instead chooses from a set of control policies.
- Mathematical programming approaches treat trajectory planning as numerical optimization. Various methods exist such as a linear quadratic regulator (LQR), or using mixed integer linear programming (MILP), but the idea is the same where a certain, possibly heuristic, cost-function can be minimized. These methods can be resolution complete, resolution optimal, and resolution sound, depending on how the problem (local minima, obstacle constraints) and controller (robustness, stability) are defined. Applications have been developed for multi-vehicle path planning and methods with risk avoidance. A commonly used method for UAV applications is *MPC*, where optimality and completeness again heavily depend on the definitions of constraints and the goal. Using the vehicle model, a trajectory is optimized within a certain control horizon.
- Other methods include the *dynamic window approach*, where a set of translational and rotational velocities are periodically optimized based on a distance from the goal, and the position of obstacles. This method has been combined with a global path planner, hierarchically decoupling the problem of planning and control. The potential field approach can also work with a differentially constrained problem serving as part of a feedback controller, though care must be taken for stability. Another approach is *reactive planning* which assumes no prior knowledge of obstacle positions nor do they perform global planning. As such these methods are usually developed as a local, lower-level, obstacle avoidance controller using (among others) visual flow, fuzzy logic, bio-inspired AI, or learning-based methods, while a separate controller ensures global goal following.

This has almost all been sourced from Goerzen, Kong, and Mettler [28], and the logic agrees with the target detection perspective from Robin and Lacroix [13]. Certain methods perform better for a UAV

path-planning application. From those, there are even fewer that can take into account the real-time computation constraints, limited sensor range, and possible lack of a map, besides the problem of the UAV state space. From those, *the hierarchically decoupled planning and control, reactive planning, and the mathematical optimization (specifically the MPC) are chosen as a baseline design direction.*

González et al. [32] took a different approach to characterizing the methods in Part 2.3.2 (which will not be reiterated for brevity). Simply put the methods can be summarized as Graph-searching methods (A*, Dijkstra, state lattices, etc.), sampling based planners (such as rapidly-expanding random trees), interpolating curve planners (line and circle, splines, polynomials, Bézier curves), and numerical optimizers (such as MPC). Line and circle approaches combine straight lines with curves around obstacles to find a shortest path, it has low computational cost and ensures shortest path for car-like vehicles. To clarify, their work focuses on methods for planar motion, but even in the 2-dimensional case they identified that real-time dynamic obstacle avoidance is an unsolved challenge. Specifically, due to the real-time constraint in high-speed scenarios, and the main problem being time-consuming perception algorithms.

Dynamic obstacle avoidance

The motion planning algorithms introduced so far concern themselves with the problems of static environments, which is not necessarily the case in uSaR. Mohanan and Salgoankar [26] have summarized various robot motion planning approaches in dynamic environments in their work, from potential fields methods and visual motion planning, to probability based methods. As most methods were already addressed by Goerzen, Kong, and Mettler in [28], summarized further in Part 2.3.2, only a few methods will be addressed here and the modelling of the dynamic obstacles.

Most higher level information used in these methods, such as the velocity or predicted path of objects, cannot always be directly measured (such as the distance which can be measured with a LiDAR). Rather the vehicle often uses a costly perception algorithm that might assume knowledge of self-motion, and periodic sensor inputs. This takes its own inaccuracies which were mentioned but not discussed in detail.

Velocity-based motion planning algorithms use the velocity of dynamic obstacles to predict a collision-free path. One example is the time-varying dynamic window, see Figure 2.3, where a set of trajectories are calculated based on the predicted motion of the obstacle.

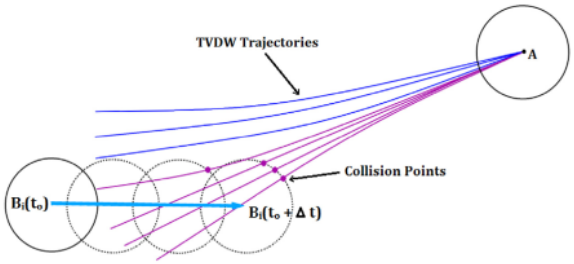


Figure 2.3: Time varying dynamic window [33].

However, most approaches assume erroneously that the obstacle moves in a straight line. So other methods assume a piecewise linear obstacle by using a smaller prediction window (partial motion planning). Another method exploits more detailed knowledge about the obstacle's trajectory, as is the case with inevitable collision state (ICS) and ICS-AVOID, which use non-linear velocity obstacles (i.e. obstacles that do not move in a straight line), see Figure 2.4.

Real-time adaptive motion planning (RAMP) performs simultaneous planning of path, trajectory and execution of motion, and can be used to minimize multiple criteria (such as energy, time, or flexibility). Whereas an MCDM generally requires plenty of precise information before making the best decision, RAMP is an optimization process that performs on-the-go.

Trajectory planning with differential constraints must have fast online performance, and as a result is slightly less strict on the accuracy and interpretation of the sensors. Several methods mentioned in Part 2.3.2 fall under this category, and to perform dynamic obstacle avoidance, most of these methods rely on some accuracy or continuity regarding sensor data and interpretation for the obstacles' trajectories.

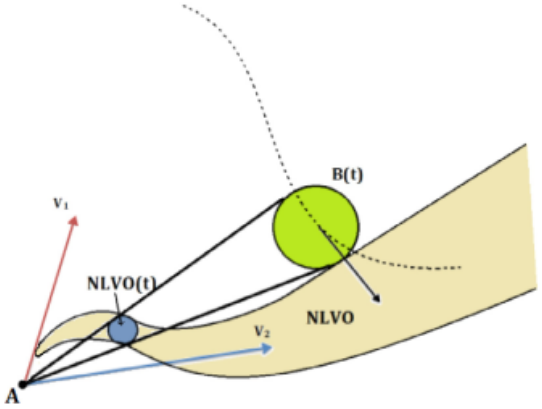


Figure 2.4: Non-linear velocity obstacles (NLVO) [26].

There are also other interesting algorithms. Probability based motion planning, for instance, assumes uncertainties (particularly state uncertainties) as part of the controller design. POMDP falls under this category. POMDP works by discretizing the system to a finite set of states. The agent, using probabilities, assumes a belief state based on observations affected by uncertainties. The belief state limits the choice of actions. The best action is selected to maximize a reward function. The current action and state lead to a future state. Future state optimization can be performed by including future state estimates and actions, leading to a decision tree, where the long-term reward can be maximized. The main drawback is the time complexity as the decision tree expands geometrically, due to the branching. There are methods to resolve or ameliorate this, such as branch-and-bound techniques.

It should be noted that it is possible to modify the other algorithms to include such uncertainties. For instance, the MPC can use stochastic control. Velocity obstacles can instead use probability velocity obstacles (PVO) which uses a probability grid to encompass possible obstacle states, but it is not efficient for velocity obstacles which do not move in straight lines [26].

Another set of algorithms concern themselves with multi-agent systems. In this case the obstacles (which can include other agents) are no longer passive, but reactive by design, because the agents can adapt their behaviour based on an underlying model or through communication (see Part 2.3.3). There are various models which include the agency of obstacles by design (rule-based, centralized, crowds), but other methods that were mentioned can be adapted to (possibly inefficiently) account for that possibility. Assuming non-linear velocity obstacles is sufficient for obstacle avoidance, but not optimal for the multi-agent system as a whole.

Regardless of what path-planning approach is taken, the performance of dynamic obstacle avoidance is as good as the dynamic obstacle model. Some methods are less reliant on sensor information than others. Following the requirements of real-time computation, UAV state space, limited sensor range, and possibly unreliable sensors, the complexity of the dynamic obstacle avoidance algorithm has to be decoupled from the global path-planning problem.

Selected path-planning method and further development

From the review so far, a hierarchically decoupled planning and control MPC approach with dynamic obstacle avoidance is one of the better controller combinations. This was indirectly obtained using the comparisons of several review papers ([26], [32], [28], and [13]). The details of the MPC are extensive enough to be discussed in chapter 2.4. There are two hierarchical MPC path-planning papers that will be used as inspiration: Jamshidnejad and Frazzoli's work [34] for 2D which will be expanded to 3D, and Bemporad and Rocchi's work [35] for 3D UAVs.

Bemporad, Pascucci, and Rocchi [35] developed a hierarchical MPC for a UAV, with a linear MPC for tracking generic position references, and a hybrid MPC for obstacle avoidance, subsequently applied for formation flight [36]. Jamshidnejad and Frazzoli developed a hierarchical MPC which is used to control

circular unmanned ground vehicle (UGV), using a shortest path planner which includes obstacle avoidance. Figure 2.5 shows the static obstacle avoidance by Bemporad and Rocchi [36], while Figure 2.6 shows the static obstacle avoidance by Jamshidnejad and Frazzoli [34].

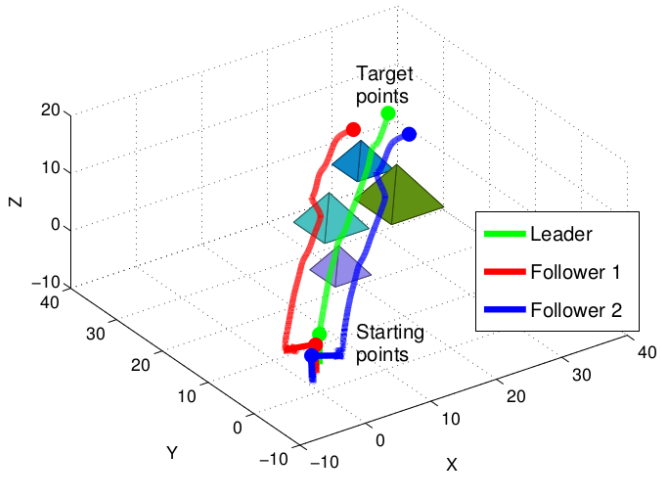


Figure 2.5: Obstacle avoidance performed by formation flying UAVs in the work of Bemporad and Rocchi [36].

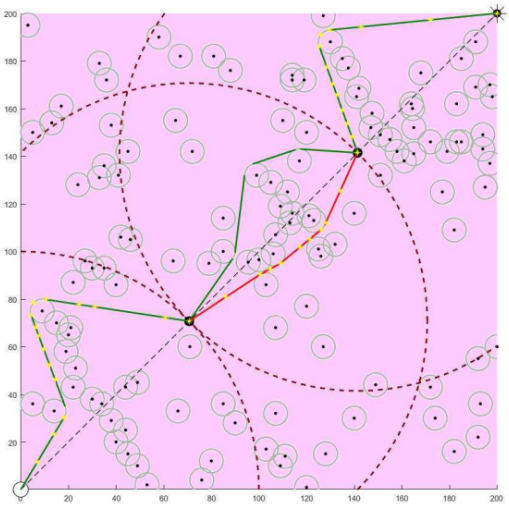


Figure 2.6: Obstacle avoidance performed by UGV in the work of Jamshidnejad and Frazzoli [34].

The obstacle-avoidance method in Jamshidnejad and Frazzoli's work [34], is a variation on the computationally inexpensive line and circle method. However, rather than interpolating path waypoints with straight lines and circular arcs, instead a heuristic algorithm is used to find the shortest path. Using straight lines, and circular arcs around obstacle boundaries impeding its path, the robot vehicle can safely reach the goal. If the goal is outside the sensor range, a temporary goal is generated between the robot's position and final goal, based on a customizable trigger. Finally, once the shortest path algorithm is done, it returns a set of waypoints which the waypoint following MPC (with a free space constraint) aims to follow by minimizing its cost function.

The obstacle avoidance method in Bemporad and Rocchi's work [36], models the obstacles as tetrahedra (for simplicity). The obstacle is avoided by setting binary 'obstacle avoidance' variables to 0, which return 1 if the linear inequality (position of UAV is inside the obstacle) is violated. They compared the method with a potential fields approach, and found that obstacle avoidance with the potential fields method is computed in the order of a few milliseconds, the decentralized approach took about 73ms, and the centralized 466ms. Though the communication is addressed through a hierarchical system, it does not take into account real-life factors such as communication delays and data transmission accuracy. This shows a relatively slower controller, but one that is sufficiently quick for real-time applications.

Interestingly, the MPC by Jamshidnejad and Frazzoli contains an autotuning approach for the prediction horizon (explained in Chapter 2.4), but it does not have hierarchically decoupled controllers. The MPC by Bemporad and Rocchi, on the other hand, has the opposite. Note that, though they have been identified as possible future developments, neither method includes sensor uncertainties, dynamic obstacles, effects of communication on the locomotion, or the effect of real-life practical implementation. These are the directions of development which will be considered.

2.3.3. Communication

Communication for the application of a multi-agent robotic uSaR team is either done between the human in the loop and the robots, or between the robots themselves. When looking at inter-agent communication the problem can be classified through its means or its goal. Rossi et al. [37] made a review and taxonomy for multi-agent algorithms with collective behaviour. Drew [7] performed a multi-agent study for motion planning and navigation. Regarding communication, he claims that (efficient) task allocation is required to address the scalability of multi-robot systems.

Methods deemed viable for multi-agent motion-planning, including task division, will therefore quickly be covered in Part 2.3.3. Human-robot interaction is beyond the scope of this project, so it is heavily summarized in Part 2.3.4. Control hierarchies within the selected controller's perspective will be discussed in Part 2.4.3. Means of communication (subsystem design), bandwidth allocation, and maintaining communication through, for instance, daisy-chaining or formations are deemed beyond the scope of this project. Instead the controller will be simply made *compatible* to various constraints from the communication aspect.

Considerations for path-planning

Does communication imply a black-box data transmission, does it determine or influence the goal, or does it define the details of the control strategy (such as through cooperative planning and task division)? It can be assumed to be unknown, so the controller must be made *compatible* with various communication strategies. In the definitions set forth by Rossi et al. [37], a range of algorithms are compared for motion planning, goal searching, and other missions. They defined motion planning as path-finding in cluttered environments (a bit more than obstacle avoidance). Goal searching is defined as target searching (optimotaxis source-search which works similar to a potential function, and global frontier techniques which are capture solutions [13]). From the global mission perspective the proposed algorithms following the taxonomy of Robin and Lacroix [13] are preferred.

The task allocation problem can be formulated and performed many ways. Task division for this research is mostly, if not only, the division of regions for search. For example, this can be done using auctions or swarm-based methods, but a detailed design and comparison of the effects of communication and task division algorithms is beyond the scope of the project. For more details, the reader is referred to Khamis et al. [38] and Rizk et al. [39].

Explicit communication is not necessary. It is also possible to ensure that the self-assigned task of the agent leads to a cooperative victim localization strategy (emergent strategies). Among the methods for decentralized control identified by Drew [7], some can be used to do this such as modifying the environment using supplementary networking devices (RFID tags), or mechanical means (such as sprayed chemicals). Other ways use implicit communication through stigmergy (the honeybee's waggle dance), or machine-learned approaches, possibly based on the motion or behaviour of the other agents.

The approaches determined in Part 2.3.2 (hierarchically decoupled planning and control, reactive planning, and the mathematical optimization in the form of an MPC), can directly address the addition of new constraints to their motion planning. When considering communication with motion planning, two of the methods suggested by Rossi et al. [37] are both hardware tested and MPC in nature: decentralized and distributed MPC. The difference between these two approaches lies in the extent of collaboration between robots, addressed in further detail in Part 2.4.3.

Communication could also be used to perform *co-localization*, using the data of other nearby rovers to improve the state estimate. A problem that could arise in this case is asynchronous data transfers, where the controller receives inputs from the different time steps from different robots [40]. This can be resolved to some extent by either idling during each communication update step, using a prediction-belief approach (for instance, with a Kalman filter), or to constrain the problem and subsequently relax certain constraints or costs based on the communicated data (for example, from robot A's perspective, robot B can move at least 10cm before hitting an obstacle, so after communication, robot B can move closer to the wall if so desired).

For now, considering the multi-agent nature of the simulations, two approaches can be programmed: either fully decentralized (with no explicit communication) or distributed/centralized (with explicit communication). The more robust solution would be one that is not reliant on data transmissions. On the other hand, using communication could lead to more globally optimal searches, at basically the same computational efficiency (using methods such as separating the MPC problem as in [40]). This is a trade-off that can be avoided by having a hybrid MPC that changes based on the reliability and availability of communication data, or to have event-driven communication. Event-based communication addresses both communication and its absence, by making the controller compatible with a set of flexibly-programmed behaviours (or constraints) or respective triggers ([41], [42]). State estimates can be updated using the event-based measurements, dealing with asynchronous data using synchronous updates within a space bounded using the event-based measurements.

When relying on communication, several constraints need to be set up such that the agents remain in communication range throughout the mission, to share the data they find or the actions they undertake. Secondly, task switching and goal setting needs to be made compatible with communication (for task allocation - the main benefit of communication). Creating a hybrid model (conditional communication) would avoid having to make that choice. Rossi et al. [37] identify the resilience to communication disruptions for a collective behavioral algorithm as a research area of interest. Formulating a distributed or decentralized controller in the proposed form is not necessarily a collective algorithm, but the underlying concept could be a solution for robust communication (namely, a controller compatible with communication, using event-driven communication, where communicated data supersedes/adapts local autonomy based on certain parameters).

2.3.4. Other

There are a few other considerations which fall beyond the scope of this project, but through research were found to be of interest for a final implementation. These are the fleet design (e.g. ground vs. air vehicle), system robustness (e.g. how to deal with agent loss), and human-robot interaction (e.g. adjustable autonomy). A detailed controller design flexible to these concerns is beyond the scope of this project, however, if possible, it should be attempted to make the controller compatible with them by virtue of a modular design approach.

Fleet design

UAVs have been mentioned as scouts, since UGVs could struggle with unknown terrain. However, UGVs could assist in obstacle manipulation to improve exploration or victim detection efforts [7]. Secondly, the usual flight time of 10 minutes for teleoperated UAVs, or (ideal) 30 minutes for micro UAVs, is too short for an extended SaR mission. Simply due to this, a heterogeneous system (with UGVs serving as potential charging platforms and data fusion centers) is recommended [7]. Besides the distinction between the method of locomotion of UGVs/UAVs, the agents can also be made heterogeneous by using different sensors.

When using heterogeneous fleets a different controller architecture must be designed that would take into account the differences between the agents. The communication architecture, motion planning, robustness, and global controller design must be designed to account for the heterogeneity.

Controller robustness to hardware states

Hardware robustness for the controller can be achieved at two stages: system-level and agent-level. Luckily, the methods for robustness are the same at both levels. Robustness can be achieved through redundant use of parts (e.g. multiple visual cameras) or through active fault-tolerant control (see [43] and [44]). In case of multi-agent systems, the redundancy comes in the form of the multi-agent aspect, while fault-tolerant control could be achieved through reactive communication in case of communication loss.

Human-robot interaction

Human-robot interaction is perceived as a major challenge for both researchers and practitioners ([16], [7]). As the number of agents in the system increases, so too does the cognitive burden on the human practitioner who controls or supervises the system. To combat this, interfaces with future state prediction and greater abstraction are used.

Adjustable autonomy is also provided as a strategy, where an agent's autonomy scales from fully autonomous to tele-operated based on situational context (e.g. for SaR: a dangerous environment or possible victim identification) [7]. So far, this is mostly done using machine-learned models.

2.3.5. Proposed robot design

"There is an inherent trade-off between the price, functionality, and agent count in the system. Given a static budget, the core effort is to balance the gains of parallelization with the loss of a robot's functionality" [7]. As far as the controller is concerned the main trade off is the computational cost of the perception and locomotion algorithms compared to the search speed and accuracy.

Following the logic of the hierarchically decoupled global planning and local control design suggested in Part 2.3.2, the problem can be decoupled even further into its constituent tasks, see the architecture in Figure 2.7. The global planner sets the direction of exploration, and optimizes the path for victim

identification based on communication, sensors, and other data. The local planner uses the waypoints or weights set by the global planner and uses victim detection and obstacle avoidance constraints to safely and successfully perform pathing.

Decoupling the problem, and thereby the controller, means trade offs can be performed on a controller level, while possibly simplifying the other subsystem selections. The hierarchically decoupled controllers could be coded using artificial potential functions, MPC variations, or some other path-planning approach mentioned in Part 2.3.2. However, for the purpose of this project MPC is chosen, where sub-choices regarding MPC design are further studied in Chapter 2.4.

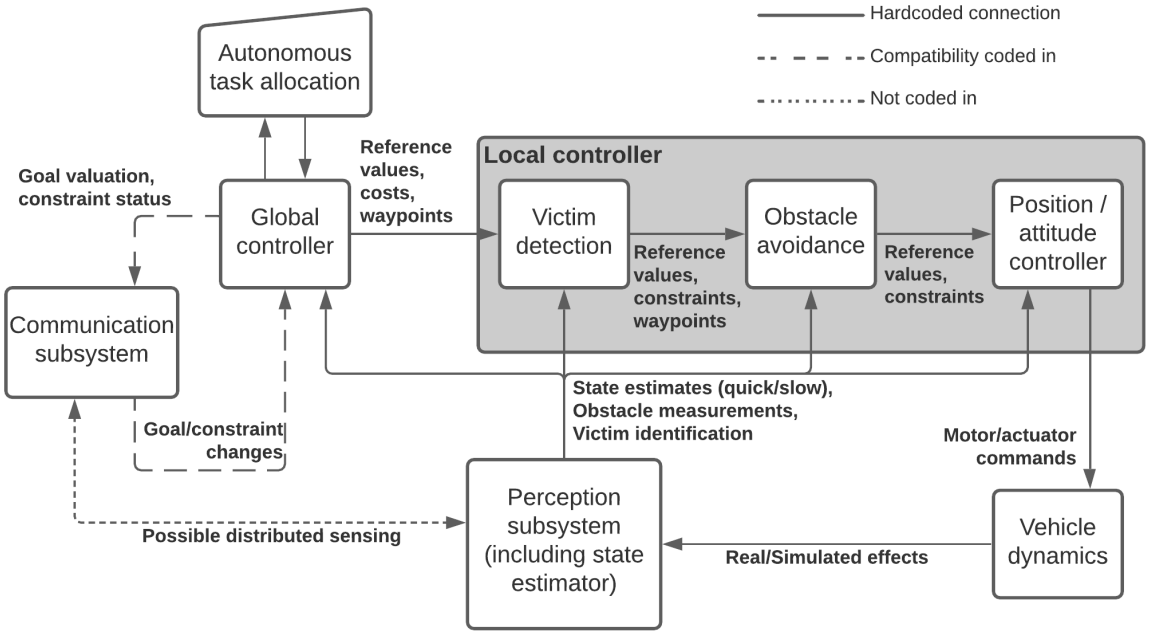


Figure 2.7: General decoupled control architecture for an agent.

General controller design

Considering the decoupled structure, each task (and thus controller) has different requirements. The position/attitude controller requires very quick updates cycles to accommodate the results of the other tasks. Requirements for sensors/perception is low latency data with just enough accuracy to be within an acceptable state estimate. This will follow from the local MPC (or possibly just a separate PID) controller with IMU feedback. Thus there are no other direct requirements besides the high update speed which is mostly controller dependent.

Furthermore, another task for the local controller is to accurately identify obstacles, and aim to avoid them. It modifies the waypoints set by the global controller through one of the methods listed in Part 2.3.2. It requires a quick update cycle, mostly dependent on the speed of the vehicle, and how constrained/cluttered the environment is. It could possibly be decoupled from the victim identification controller. At a high speed, the controller requires moderately accurate sensors, with low latency feedback from the perception subsystem. At lower speeds, the latency is not that important. The controller therefore is a limit to the operational speed of the system.

The victim identification controller has to modify the pathing to possibly include a few (2-3) different view angles of an area, possibly more if required to verify a potential victim (feedback from the perception subsystem). The perception needs to be as accurate as possible within a set time constraint, as this is the underlying mission. High accuracy perception is required, but low latency is not necessarily required.

The global path-planning controller uses possible information from the controller subsystem, internalized map, or weighted function to specify the goal for the local controller. It is formulated in such a way to reach

areas and fulfil the system level mission task. The autonomous task allocation could issue a ”return” task when the battery is nearing depletion, or some other tasks based on circumstances. It requires a slow but accurate positional estimation.

Perception selection
It is possible to select a LiDAR to address obstacle avoidance and further mapping. Though good for obstacle avoidance (not victim detection), it is costly, reducing the availability of the system to impoverished communities. An infrared camera unfortunately runs into the same cost problem. The solution is therefore to rely on several infrared proximity sensors to assist the short-range or possible out-of-frame obstacle avoidance with a visual camera. Victim identification, preferring the multi-feature detecting approach, could be done using the camera (YOLOv3 or [17]), pyroelectric sensor, and a microphone. Limiting any further financial cost increases, a GNSS/IMU/visual camera sensor fused approach is used for the global localization.

Fleet selection
Not only low in financial cost, the sensors are also low in power consumption. Therefore the main power use comes in the form of the controllers' optimization processes and the locomotion system of the robot. Both a UGV or UAV are possible. The ground vehicle is less power hungry due to the passively stable idle state and can carry more/larger batteries due to less strict weight requirements, while an air vehicle can reach areas the ground vehicle cannot. The proposed multi-agent system is therefore heterogeneous, using UAVs to reach places the UGV cannot reach or are out of its way. This way, the power storage features of the UGV can be used with the mobile exploration of UAVs. Multi-copters will be selected for the UAV, though it should be noted that traditional helicopter designs are equally viable for indoor exploration (a miniaturized version as in [45]).

2.3.6. Simulation platforms
Simulators can happen on multiple stages, namely model-, software-, or hardware-in-the-loop (MIL, SIL, HIL) [12]. The MIL approach uses high level of abstraction, useful to develop algorithms without involving dedicated hardware. SIL uses a model close to the real one, where code that is tested is written for that particular mechatronic system, within a modelling environment which would be useful to verify (or validate) that software. HIL tests the actual hardware (sensors, actuators, etc.) with the final software, connected to the simulator, providing feedback regarding possible signal/sensor smoothing, and might give rise to details that were not considered during the simulation design process.

With respect to MIL, an interpreted programming language such as Python or MATLAB is sufficient. For SIL, Mualla et al. [46] performed a literature review on the agent-based simulation of UAVs, presenting many different frameworks. Most researchers, however, tend to use their own simulation frameworks from scratch, possibly because existing simulation frameworks do not cover their area of interest (such as continuous-time simulation or specific environmental conditions), or because slightly modifying their MIL approach is sufficient. In case of the current scenario it would be of interest to also simulate possible mission-affecting levels by virtue of the environment, e.g. a roof collapse blocking the exit, leading to a possible loss of an agent.

Considering the intent of this thesis, MIL is sufficient to develop the general controller design. SIL is instead useful for a higher level verification which can be used for multi-agent testing in ”real” simulated conditions, which include sensor and environmental uncertainties. HIL is going to be used for the development as it is the usual course of action [12]. For HIL, if required, it can be tested at the TU Delft's MAVLab. Test-driven development will be used throughout.

2.4. Model predictive controller
MPC has been identified as the control theory that could be used to optimize both the global and local task. Compared to a simple feedback gain, there is more underlying theory which requires a bit of research into both the fundamentals and any further developments.

Part 2.4.1 lists the fundamental theory of how an MPC works. Part 2.4.2 on the other hand, introduces the concepts of stability, feasibility, and robustness, with several means to account for them in the controller design. At this stage a working MPC can be made, but considering the multi-agent aspect, the sensor and

state uncertainties, as well as the non-linearity of both the agent model and obstacles, several extensions to the original linear MPC are considered in Part 2.4.3. Part 2.4.4 contains several tuning methods for the various parameters in an MPC, as well as tuning for its robustness. Lastly, using this knowledge, a development plan is set forth for the hierarchically structured MPC in Part 2.4.5, which will be used in the thesis.

2.4.1. Fundamentals

MPC is a control method which uses the knowledge of the model to predict its future states and future inputs for "t" timesteps. In general control terms, there are two gains that require tuning, namely the throughput and the feedback gain that modify some error or a reference signal. The MPC is more complex than the usual controller, using an optimizer and a model of the controlled system. See Figure 2.8 for a visual representation of this process using discrete time variables.

The optimizer usually aims to minimize a cost function (for example $\sum (x_r - \hat{x})^2$) based on a reference signal x_r, and the predicted future states ($\hat{x}$) or outputs ($\hat{y}$). These are predicted using a model of the system. The controller (consisting of the optimizer and model), optimizes $\hat{x}$ within a number of future steps called the prediction horizon (N_m or H_p). It does this by changing the controller output ($\hat{u}$) over the control horizon (N_c or H_c), which drives the system. As the model of the system is used to predict $\hat{x}$ and $\hat{u}$, it can be understood that an accurate model is a cornerstone for the MPC method. It should also be noted that the state is not always fully observable (or badly observed due to sensor quality), and state estimation techniques can be employed. See Figure 2.9 for an illustration of the control and prediction horizon.

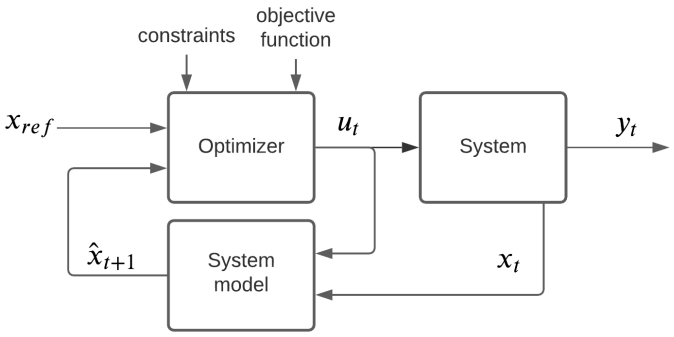


Figure 2.8: Feedback model predictive controller

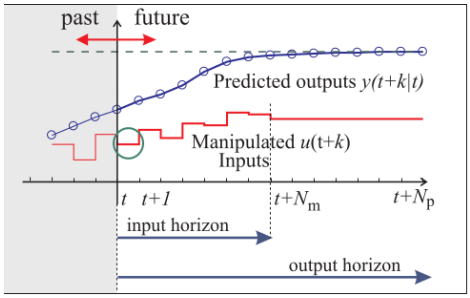


Figure 2.9: Schematic diagram of model predictive control horizons [47].

The optimization can be constrained by real-life limitations or design considerations, which are translated into mathematical constraints. Constraints can act on the input signal u by, such as in the case of an aircraft controller, limiting the maximum deflection of a control surface or the maximum power that can go to a motor. Besides enforcing input limitations, the system state can also be constrained. Continuing with the aircraft example, it is thus possible to set constraints to its state to avoid stalling during flight maneuvers.

Another way to define constraints is by looking at *how* they are enforced. A linear or linear quadratic solver (linear constraints, quadratic cost function) cannot resolve convex or concave constraints (such as variable multiplication). These definitions play a large role in how quickly (or even if!) an optimizer can find a solution to the problem. For more complex constraints (stochastic or quadratic) second-order cone programming can be used. A common way to represent the MPC mathematically is shown in Equation 2.2.

$$\min J = \left(\sum_{k=0}^{H_p-1} x_k' Q x_k + \sum_{k=0}^{H_c-1} u_k' R u_k + V_f(x_{H_p}) \right) \tag{2.2a}$$

subject to

$$F_1 u_k \leq G_1 \tag{2.2b}$$
$$E_2 x_k + F_2 u_k \leq G_2 \tag{2.2c}$$
$$\text{and "stability constraints"} \tag{2.2d}$$

Where Q and R are weight matrices for the state and input signal, respectively, and E, F, and G are matrices that set the input and state constraints [47]. x'_k is the transposed state vector at discrete time step k. It also includes a terminal cost V_f and/or stability constraints that ensure closed loop stability. Stability of an MPC, terminal cost, and stability constraints (Equation 2.2d) are explained in Part 2.4.2. Note that the constraints in Equation 2.2 can also include future (x_{k+1} or u_{k+1}) and past terms (x_{k-1} or u_{k-1}). Using the system model as a constraint is one way future states can get estimated.

2.4.2. Stability, feasibility, and robustness

To understand how to work with an MPC, the challenges surrounding an MPC must be understood. Stability implies the output in Figure 2.9 will converge to the reference value. Feasibility implies there is a solution to the problem (given the constraints). Robustness implies that a usually bounded uncertainty does not affect the stability or feasibility of the MPC.

Stability

By definition, Equation 2.2 is not necessarily stable nor feasible. There are various methods to ensure stability, each aiming to eventually set the system's state x to either 0 or to guide it to an linear quadratic (LQ) invariant set. As was summarized by Bemporad and Morari [47], there are 5 directions to ensure stability (1-4 are proven through Lyapunov stability theory, whereas 5 is norm-shrinking by definition):

1. It is possible to set a *terminal constraint*, where $x(t + H_p|t) = 0$.
2. For asymptotically stable systems the terminal constraint can be avoided by using an *infinite prediction horizon*, with a limited control horizon.
3. It is also possible to use a *weighted terminal cost* instead, based on the stabilizing solution from the algebraic Riccati equation. This is done by substituting the solution P, into Equation 2.2a, such that $V_f(x_{H_p}) = x'_{H_p} P x_{H_p}$.
4. Another way is to find a *terminal set*, which is LQ-invariant. Stability can be ensured if the state at time $t + H_p$ is contained within the set, as a feedback controller (usually linear, $u_{t+1} = K(x_t - x_{ref})$) can take over at that point.
5. A *contraction constraint* instead states that $\|x_{t+1} - x_{ref}\| < \|x_t - x_{ref}\|$, which naturally leads to convergence if the constraint is fulfilled.

All are valid methods, with their own considerations or drawbacks regarding computation or optimization. The terminal constraint formulations have a few problems: Setting the state to 0 within H_p time requires a large control cost, larger still when using a small prediction horizon [48]. Considering the input constraint in 2.2b, maintaining feasibility could limit the set of stabilizable states even further (possibly leading to a larger control cost). To some extent these problems can be mitigated by increasing the prediction horizon or by only stabilizing the unstable modes.

The infinite prediction horizon is a good solution, but only if the system is inherently stable will the optimization lead to a locally optimal solution [48]. The weighted terminal cost does not assume inherent stability, such as the infinite horizon [48], but it does not guarantee the optimal solution and can run into problems with feasibility. Terminal sets therefore more common [49]. There are variations on this based on scaling [50] or translating (moving) the terminal set based on the state.

Contraction constraints do not require long prediction horizons, and avoid the computation of the terminal set and enforcement of the constraints. This is mainly the reason why they have become popular for complex non-linear systems. Combinations of contraction constraints and terminal costs also perform well [49].

Feasibility
Compared to stability, feasibility can be better understood in real-world terms. Suppose an uSaR scenario, where an autonomously controlled robot enters a hallway, after which the only exit collapses. At that point the MPC can no longer find a solution to lead the robot back to the rescue team. The lack of feasibility can also happen due to the controller's inputs, using an aircraft as an example, by pitching the nose too high, there is flow separation on the wing, leading to a stall. This drastically reduces the effect of the control surfaces, thus possibly leading to infeasible cases (though there are controllers that can specifically resolve this [51]).

The reasons why feasibility can come into question is possibly due to a disturbance, a prediction window that is too short, or due to inflexible state constraints [49]. Resolving these can be done through robust design, increasing the prediction window or setting additional constraints for feasibility, or by 'softening' the state constraints, respectively.

State constraints can be modified to include slack variables, such that the violation of state constraints is done at the cost of a penalty to its cost function, yielding a solution which could otherwise not exist. However, this only applies to state constraints, as input constraints are always 'hard' (as they are usually based on real-life equipment which only operates within certain conditions). It is possible to use a "reference governor" ([47], [49]) that creates a feasible reference trajectory for the input signal (it essentially smoothes the reference signal when abrupt changes would lead to constraint violations).

Robustness
Another one of the main drawbacks of the basic MPC is that it does not take into account disturbances and uncertainties (such as wind gusts for an aircraft) in the input, or sensor noise in the output. For real-world applications these are vitally important, perhaps even more than the speed of solving the MPC. As put by Bemporad and Morari, robustness means that the system maintains stability and a certain performance for a specified range of model variations and a class of noise signals (uncertainty range) [47].

In other words, the first step to designing a robust MPC is to specify the uncertainty, the second is to account for them. There are methods, though they come at a cost of performance (in both computation and optimization) [47], [49]:

1. Uncertainty intervals through step- and impulse-response analysis: Not sufficient to guarantee robustness, as oscillating step responses could be allowed [49]. Still used in practical applications due to the very low computational cost [47].
2. Structured feedback: By modelling the uncertainty (as multiplicative, additive, or parametric) and feeding it back into the system. Based on the uncertainty (whether it can be modelled as a diagonal matrix, or full matrix), singular value decomposition or computational approximation can be used to evaluate whether robust stability is satisfied.
3. A set of possible models (or model states): Either a full set of finite possible models, or a set based on the extreme cases (and generating an 'envelope'). Then using the L_∞-norm, the worst case error is minimized (or by using the L_2-norm the least-squares errors can be minimized).

These methods can be applied to any MPC formulation. However, it is also possible to include robustness in design. One way is to define the cost function of the MPC to favor robustness, in a way such that the worst-case prediction must contract [49]. Another method is a stochastic MPC, where a stochastic dynamical model of the process is used to predict its possible future evolution [52]. Regardless of which method is chosen, the truth is that robustness is traded-off against optimization based on how bounded the uncertainties are.

2.4.3. Variations
So far, the fundamentals of MPC have been analyzed, with various possible methods to assert the stability of the whole system, and robustness of the controller. There are many variations of MPC based on the data, use-case, and type of system to be controlled. The way the MPC is defined is through the underlying model, constraints, objective function, and input signal. Depending on these factors various optimization algorithms can be used, from simple linear, to sequential quadratic programming, or even other non-linear methods such as genetic algorithms which define the computational speed, and to a much lesser extent the optimization.

The main distinction on the model side is whether it is described as linear or non-linear. Once it is known what the model is, it is possible to implement robustness with either the methods in Part 2.4.2, or through a dedicated robust MPC design. If stochastic variables or inputs are used stochastic MPC is a required design direction. Lastly the level of centralization can be developed through specifying a centralized, decentralized, or distributed MPC.

Linearity
MPC is linear or non-linear based on the model, constraints, and optimizer. It is even possible to have a hybrid MPC with discretized states when using switches. Considering the UAV use-case (a non-linear system), the way to use a linear MPC would be through piece-wise or repetitive linearization. When dealing with a linear system (and thus linear constraints), quadratic programming is the highest level of complexity, already setting the stage for a fast MPC.

As a matter of fact, **linear MPC** can use the (approximate) primal barrier method in order to solve the quadratic programming problem. As far as speed is concerned, linear or quadratic programming problems can be solved in milliseconds [53]. However, it is limited to only linear systems. This means for non-linear systems, the non-linear residuals can be used, but the system dynamics have to be linearized around the new point, as is done with [54]. There are several ways to perform linearization, see [55] for a comparison. On a hardware level, attempts to create a faster MPC led to a scalable embedded systems approach. Linear programming problems implemented in such a way can be solved in mere microseconds ([56],[57]).

However, not all controllers can be designed using successive linearization, but high speed solutions are still desired. In the words of Oberdieck and Pistikopoulos with **explicit MPC** "... the online computational requirements are reduced to a point location and a function evaluation, avoiding in principle the need for online optimization" [58]. To account for the hybrid system, the usual state-space model representation in Equation 2.3 is adjusted by using auxiliary variables seen in Equation 2.4a and constraints in 2.4b.

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k + Du_k \end{aligned} \tag{2.3}$$

$$\begin{aligned} x_{k+1} &= Ax_k + B_1u_k + B_2\delta_k + B_3z_k \\ y_k &= Cx_k + D_1u_k + D_2\delta_k + D_3z_k \end{aligned} \tag{2.4a}$$

$$E_2\delta_k + E_3z_k \leq E_1u_k + E_4x_k + E_5 \tag{2.4b}$$

Where A, B, C, and D are the system's state-space equations, x_k is the state at a discretized time step k, and z and δ are auxiliary control variables for real or binary control, respectively. Following the methods defined in [59], [60], and [58]; the state-space adjusted for the hybrid system is re-defined as a multi-parametric mixed integer (linear or quadratic) problem. Based on the parameters several critical regions are defined, each with a (linear or quadratic) optimal value function. Each critical region CR, has its own solution given by $x_{i,k}(\theta) = K_{i,k}\theta + r_{i,k}, \forall\theta \in CR_{i,k}$, with $k = 1, .., m, i = 1, ..., n$, where n is the upper bound for the number of critical regions, m is the number of critical regions created, θ are the parameters, with offset r and control gain K. Upon solving the multi-parametric problem in each critical region, the solutions and boundaries are stored in look-up tables which the online, i.e. real-time, controller will use.

Unfortunately one of the main drawbacks of the explicit MPC is that "the number of entries in the table can grow exponentially with the horizon, state, and input dimensions" [53], suggesting the use of more costly hardware, or online MPC approaches. With the interest for neural networks growing, neural networks with rectified linear basis functions were found ideal for approximating large solution spaces [49], thus reducing the downsides. Other drawbacks are that it cannot adequately respond to untrained scenarios, and that highly non-linear behaviour requires many critical regions.

When neither method can be used, due to the required control accuracy or adaptability to a changing environment, non-linear MPC is the other choice. Theoretically, the non-linear MPC is higher in accuracy as it can exploit state interdependencies and push the system's limits [61]. In case accuracy is of such concern, it is possible to model the system dynamics in continuous time to even get rid of the discretization error [62].

Kamel, Burri, and Siegwart [61], compared a **non-linear** and linearized MPC for their simulated hexacopter implementation, with the non-linear being more accurate. Interestingly, despite the theory claiming otherwise, the non-linear implementation was computationally quicker. This proves that at the practical stage it is not the theoretical limitations that dictate controller performance, but that it would mostly be on the way the controller is implemented, thus the researcher's or engineer's work. Though they did not discuss their results, it is possible that significant computational effort was devoted to re-linearizing the problem.

One work in particular dealt with an experimental implementation of a UAV exploring an underground environment with a nonlinear MPC for control and obstacle avoidance [63]. The experimental set up was rather slow (1.2 m/s was the quickest speed in the tests), but the autonomous exploration on the field trials was successful. It had a sampling speed of 50 ms, with the nonlinear MPC computation speed of 10 ms.

To summarize, the highest computational complexity is theoretically found in the non-linear case, followed by the explicit and then the linear. There can be cases such as with Kamel, Burri, and Siegwart [61], where other factors (such as linearization) might influence the controller's computational speed. The highest accuracy (minimum L_2 error term over the control period) in reference tracking or disturbance rejecting scenarios is found in the non-linear case which fully models the interdependent dynamics. Explicit MPC follow closely behind, with the linearized methods trailing last. Regarding adaptability, explicit MPC cannot resolve scenarios that were not considered off-line.

Stochasticity

As summarized by Mesbah [64], the main distinguishing factors for a stochastic MPC are the underlying model (linear vs non-linear), the type of the uncertainties (time-varying vs time-invariant, additive vs multiplicative), the propagation of the uncertainties (stochastic tubes vs simulation vs chaos/mixtures), chance constraints, and so forth. Regardless of how it is constructed, the aim of the stochastic MPC is to minimize an expected cost with respect to possible disturbances. By definition it has a high computational cost, especially when the system is non-linear [65]. The complexity comes in the form of a high number of control constraints due to parametrization of the input (when linear it is usually done as $u = Kx + v$, where K is chosen such that the state-space model is asymptotically stable, and v is the control input determined online). Mayne [65] therefore suggests to rather use conventional MPC with tightened constraints. Research is still actively being done on the subject for its ability to deal with stochastic variables, and to resolve the issues as mentioned in [64] and [65].

However, for completeness the subject will be quickly covered. In this case the system state can be and usually is described by $x_{t+1} = f(x_t, u_t, w_t)$, where w_t is a disturbance, usually a random process. At this stage a full state-feedback is assumed for simplicity (see Figure 2.8), where the state x is measured and fed back into the optimizer and model. The other choice is adding state estimation/reconstruction to an already noise-influenced output signal. The input signal u_t is modelled as a control policy π_t, which is based on the model type. Assuming a linear model (thus using $u = Kx + v$), either the nominal or expected cost is found (using the probability distribution).

At this point both the uncertainty model and propagation are of interest as it defines the probability distribution for the objective function. Modelling can be done using bounded additive or multiplicative uncertainties. It can either be done using a stochastic tube (mostly linear systems, but recently progress is being made for non-linear systems through constraint tightening [66], or unscented transform [67]), sampling, or uncertainty evolution theories(polynomial chaos, Gaussian mixtures, or Fokker-Planck, used mostly for non-linear systems) [64]. Once an approach for uncertainty propagation is chosen the only remaining aspect is the definition of the constraints.

As the variables can be distributed over a range of values, the input constraints are either 'hard', leading to a conservative estimate and limiting feasibility, or 'soft'. Either the full bounds, expectations, or probabilities can be used (for example $P(u_k \leq u_{\max} \,|x_k) > 0.95$). Though the input constraints remained largely the same, the state constraints are instead referred to as *chance constraints*. They can be joint, individual, or expectation type. Essentially following $P(g_j(x_k) \leq 0) \geq \beta, \forall j = 1, ..., s, k = 1, ..., H_p$, for a number of s different functions g, and a probability level β.

The knowledge for this section has mostly been taken from [65] and [64]. A suggested solution is in the form of explicit (off-line) stochastic MPC. So far, the aim of this work is not to develop a fully new MPC

approach, but rather to design a controller using MPC for the mission as described in Part 2.2.3, and made compatible the system design considerations in Chapter 2.3.

Multi-agency

Multi-agent systems can have indirect interaction through the environment or communicate directly, allowing cooperative control techniques. A summary of different architectures for decentralized, distributed and hierarchical MPC has been provided by Scattolini [68].

A **decentralized** system would have the different subsystems each with a separate controller interact through their states, see Figure 2.10. This means no explicit communication is present. A **distributed** system would have the controllers communicate with other controllers in communication range to find an optimum that includes each others' predicted states, see Figure 2.12. The **centralized** (also called **hierarchical**, but it could lead to confusion with Figure 2.13) MPC has a top-down approach where a coordinator gathers every local controller's information, optimizes the problem, and gives the local controllers set-points or costs, see Figure 2.11.

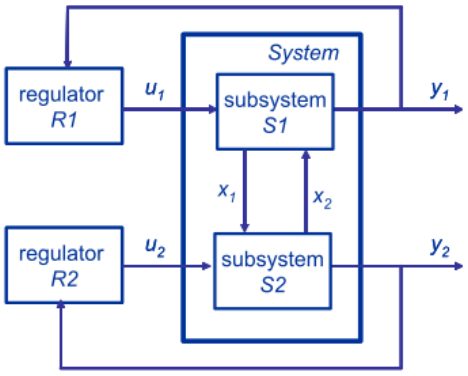


Figure 2.10: Decentralized MPC [68].

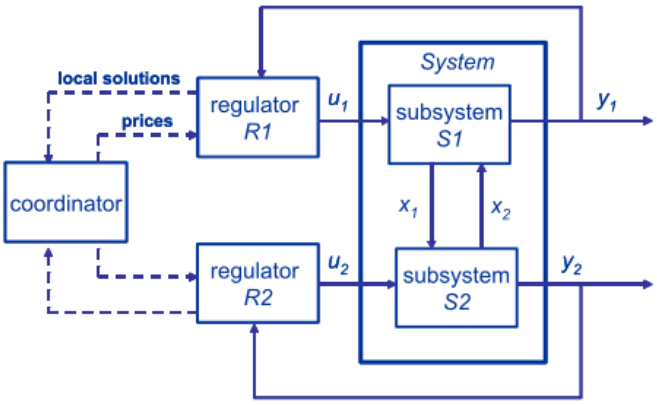


Figure 2.11: Centralized MPC [68].

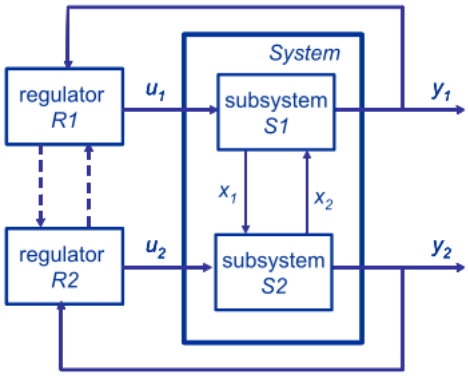


Figure 2.12: Distributed MPC [68].

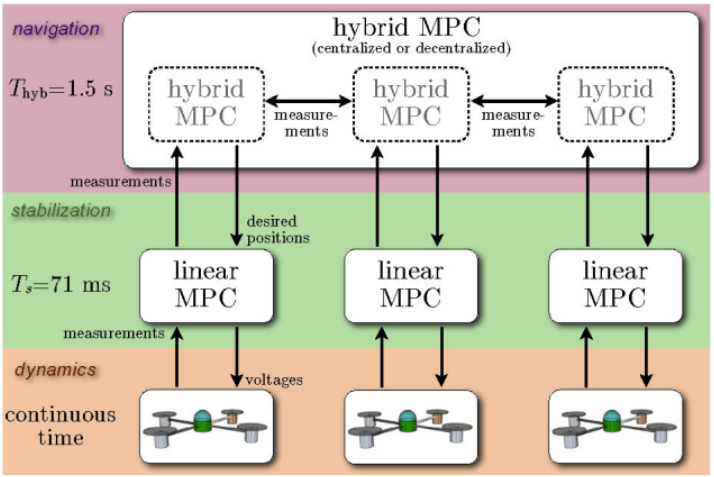


Figure 2.13: (De-)centralized hierarchical MPC for a single agent [36].

The main benefit of a decentralized MPC is the lack of required communication, as the subsystems only interact through the environment. The main demerit is its lack in global optimization: one single centralized MPC can optimize the entire system's future states, leading to better results compared to locally optimal states from using multiple decentralized MPC. Even though the analysis has been done on a system and

its subsystems, the same conclusions can be made for a multi-agent system, where the agents are the subsystems [68]. Scattolini referred to this as coordinated control.

Coordinated control can be online, by negotiating their final outcomes with their neighbours, requiring the auctioning and task division. Stabilized MPC approaches have been reported in these aspects [68], and there are many more successfully working designs without theoretical proofs on stability.

A distributed MPC is quicker than a centralized MPC [69], and does not rely as much on communication compared to a centralized system (by definition). Distributed systems do have better global performance compared to a decentralized architecture, due to their ability to coordinate. Camponogara et al. [40] suggests a framework that decomposes the MPC problem into smaller problems, solved separately by each agent. Zhihao et al. [42], and Rostami and Görges [41] employed distributed MPC with event-based communication, reducing computational burden that way.

Lastly, it is also possible to use the hierarchical MPC to represent a system that has both fast and slow dynamics (such as fast flight stabilization and slower victim of a UAV). In this sense, hierarchical means combining a fast MPC design for fast responses where needed, and a more accurate MPC for the slower dynamics, see Figure 2.13.

2.4.4. Tuning

Once the model has been decided on, the actual design of the MPC needs to be performed. This process is called tuning, where the variables of the MPC get modified for purposes of stability, robustness, or other optimization. There are two types of tuning, the first is for the model accuracy, the second is a trade-off between performance and robustness. Knowledge regarding tuning strategies has been summarized from Maciejowski's book [70] (chapters 7 and 8) and the review paper of Garriga and Soroush [71].

Model parameter tuning

There are many adjustable parameters that would define the MPC: the horizons, weights, disturbance models (covariance matrices), observer models (such as Kalman filter gains), reference trajectory parameters, and constraint parameters. Usually tuning happens a priori, but autotuning (self-tuning) methods can be of more interest in highly autonomous systems. There are some theoretical bounds or values, but most tuning is performed based on experience and "rules of thumb". Regarding model tuning, most methods suggest the use of state-space representations and performing stability analysis (looking at the root locus, Nyquist, or Bode plots, or using Routh-Hurwitz), modifying various parameters and gains until the system becomes stable.

Tuning can also be done on a per-parameter basis. For instance, the prediction horizon, H_p, can either be set to infinity for stability (see [47]), or finite, requiring tuning for closed-loop stability. The prediction window in this case has an upper bound due to real-world dynamics (dynamic environment and obstacles), sensor range and uncertainty, and the selected sampling time. Setting the value to 10 and tuning the other parameters, or using values based on heuristics (such as including up to 80 to 90% of the process steady-state) are suggested in [71]. For the control horizon H_c setting it to a default value of 1 and tuning the other parameters, or numbers based on certain criteria (e.g. the number of unstable poles), is suggested. An extensive list of tuning guidelines for each parameter has been listed in [71].

Autotuning has the benefit of updating the parameters as the system process is being optimized, thus always being set at an optimum value (based on some heuristic). The drawback is the extra computational effort. Some authors suggest methods using evolutionary algorithms (such as genetic algorithms or particle swarm optimization) [71]. The benefit of this method is that the tuning parameter optimization can run separate from the system's control optimization, and is thus easily implementable. The drawback is that several instances of the same control problem are (potentially iteratively) solved with different values.

Other methods for autotuning include modifying the MPC ("a redefinition of the objective function to include the optimization of weights"), using a sensitivity analysis ("a linear approximation of the relationship between the process output and the MPC tuning parameters"), or by using subspace identification and "minimizing the first N terms of the impulse response" [71].

Maciejowski [70] specifically addressed that tuning a feedback controller should be done including its feedback properties when constraints are active. This is very hard to do theoretically, and only a few tools can help, so simulation is suggested [70]. Much like Bemporad and Morari in [47], Maciejowski suggested set-point pre-filtering for tuning the input signal (possibly through the form of a reference governor).

Robust tuning
Assuming a sufficiently accurate model, robustness tuning can be performed, as fast closed-loop performance requires a high precision model [71]. Maciejowski suggests Lee and Yu's tuning, loop transfer recovery tuning, or using linear matrix inequalities.

Lee and Yu tuning is done offline. It requires an asymptotically stable system (the disturbance does not excite the plant state) in the form of state-space with white noise, following a disturbance model shown in Figure 2.14. State estimation and prediction is performed using Kalman filter theory.

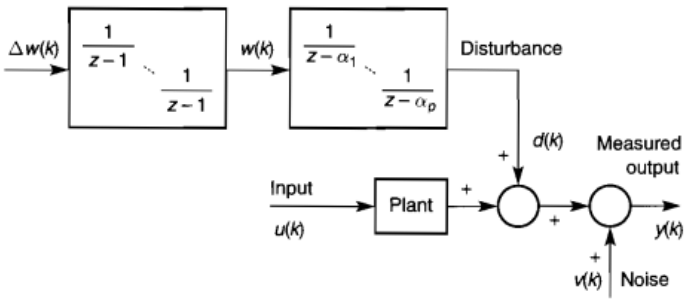


Figure 2.14: The disturbance model used by Lee and Yu [70].

Once the model is established first it is tuned for nominal stability, setting the control horizon "as large as possible within computational limits" (setting a control penalty matrix $R = 0$). Then it is 'de-tuned' for robustness by adjusting the disturbance and noise parameters (α, ρ the covariance of w, and σ the covariance of v, in Figure 2.14). If the uncertainty is well defined the control penalty matrix (R) can be increased to tune for input uncertainty robustness.

Loop transfer recovery can be done online, and it does not require a stable system. It also allows for correlated disturbances. It is developed for a linear model with a quadratic penalty cost, and disturbance and noise signals with a Gaussian probability distribution. The first step is to use a Kalman filter to find the optimal estimate, the second step is to determine the feedback gain matrix, setting up the controller. Subsequently, the disturbance and noise model (shown in Figure 2.15) is adjusted for a good return-ratio, where A, B, and C come from the state-space model for the system, A_w, B_w, C_w, and D_w are a state-space model of the disturbance process, v is the sensor noise, and Γ_w is the white noise input matrix of the disturbance state space model (taken from Kalman theory, Equation 2.1b).

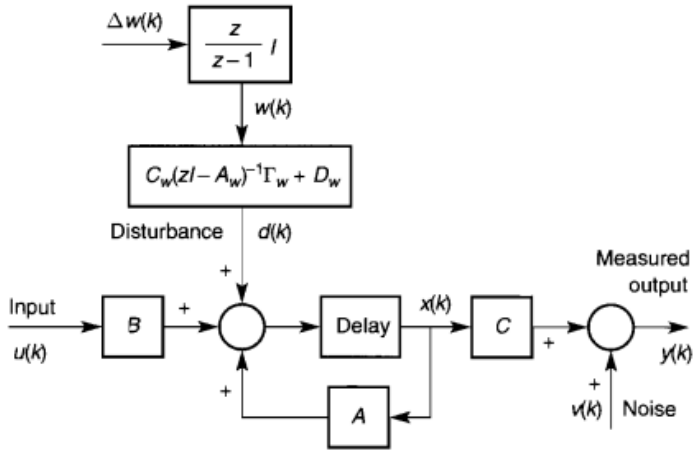


Figure 2.15: The disturbance model used for loop transfer recovery [70].

Tuning for robustness is done by first designing the disturbance model gains and adjusting the model design (the state-space). Once the return-ratio is good (based on the gains in a frequency response curve,

such as high gains at frequencies where disturbances are high), there is only one scalar parameter ρ which can be reduced.

Using a **linear matrix inequality (LMI)** approach increases the complexity of a typical quadratic programming problem to a convex LMI problem. It is possible to use with online optimization. The benefit is that it can address both robust constraint satisfaction and robust stability. There are drawbacks as the LMI optimization reduces the upper bounds, and the bounds on the uncertainties may be too conservative. For details the reader is referred to Maciejowski's book [70], particularly chapter 8: Robust tuning.

2.4.5. Final design

Considering the final controller selection in Part 2.3.5, the decoupled design (see Figure 2.7) allows each problem to be addressed separately. The multi-level MPC must account for global path-planning and goal tracking, and local victim detection, obstacle avoidance, and position/attitude determination. A stochastic MPC implementation is required for probabilistic non-linear dynamic obstacle avoidance. The position controller can be done using a linear MPC with the objective to minimize energy use or simply using a PID controller.

The position/attitude controller requires high speed, waypoint following, energy-minimizing control. This can be done through a linear or explicit MPC, as is done in most cases. However, it was found there are exceptions such as [61], where a non-linear model was used for state progression, keeping all other constraints linear, leading to a computationally outperforming non-linear formulation. So at this point, the design details of the position/attitude controller cannot be set.

The obstacle avoidance controller is dependent on the velocity. At high-velocity the controller must be made more robust by expanding the obstacle radii, limiting the configuration space. At lower velocities, the stopping distance lessens, so constraints can become less strict. Furthermore, in reality, there are stochastic features due to imperfect sensors and obstacles with unknown reactive behaviour. Though at simple levels (static obstacles) a linear or non-linear MPC are both sufficient, at highly complex levels (probabilistic dynamic obstacles) a robust stochastic (likely non-linear) MPC is required. The main complexity regarding the MPC design rests here.

The victim detection controller could set further waypoints which the obstacle avoidance controller aims to fulfil. This is mostly dependent on the requirements from the perception subsystem. Depending on the algorithm, the viewing angles could be optimized with respect to time in a newly discovered room. For that case a simple robust linear MPC that optimizes said viewing angles is sufficient. The main complexity stems from the perception subsystem's victim identification, which will likely bring slower speeds. However, neural networks such as YOLOv3 ([18], [25], and [17]) do manage to perform real-time.

Note that these controllers could all be merged into one larger MPC, but it is expected that this degree of problem decoupling will lead to time-saving improvements. The main reason is due to separation of computational choke points (victim perception, obstacle avoidance, and communicating goal changes), and exploiting this using several computational threads. Secondly, this method allows for quicker development and testing cycles, as each task's performance can be addressed, tested, and improved separately. The downside, however, is the lack of integrated multi-objective optimization as would be the case for a single larger MPC.

The global controller creates a path through some mapped, or unmapped regions, with probabilistic hotspots of victims, and possible information from a communication subsystem. A lot of conditional information, including the sensed environment, battery levels, and agent state, are used to determine the future course of action, and thereby the future path. It is a highly non-linear, hybrid problem (the model or only the cost should change based on communication or other states such as fire).

Lastly, it should also be noted that doing all the proposed measures is unlikely within the duration of the thesis. Therefore, the controllers are to be developed to different degrees, from a simple initial state, steadily progressing to a level sufficient for a somewhat robust autonomous system. Using the papers from Part 2.3.2 as a basis ([34], [36]), the following will be worked on (sorted by priority):

1. Path-planning: Initially the target is a static signal which the robot knows. The complexity increases when it becomes a search problem as defined in Part 2.2.3.
2. Communication: Two types of communication can be compared. Fully decentralized (no communication), and distributed (using event-driven communication).

3. Obstacle avoidance: Varies from static, dynamic, probabilistic, up to possibly reactive. Initial implementation will repeat the work of the two papers in Part 2.3.2, using static obstacle avoidance. More complex obstacle avoidance will come from the obstacle model definitions in Part 2.3.2. Multi-robot simulations can only follow after dynamic obstacle avoidance.
4. Stochasticity: Initially no randomness or sensor uncertainties are assumed. Once the basic model is completed these problems get addressed. See Part 2.4.3 for a short summary from the MPC perspective.
5. Hierarchical control: Way-point generation, path-following, victim detection, and dynamic obstacle avoidance are initially decoupled. This is done to ensure a modular testing and development approach can be taken. It does not mean they cannot be merged. Merging will be done to verify the time-saving claim.
6. Dimensionality: The design will be performed in 2D (terrestrial robot) before moving to 3D (UAV). At the 3D stage, all the previous steps are essentially repeated.
7. Hardware and simulations: The hierarchical controller will first be simulated using a model-in-the-loop approach. Once developed, higher-order simulations can be done before it can be tested at the TU Delft's MAVlab.
8. Fault-tolerant control and adjustable autonomy are future developments that fall far beyond the scope of the thesis. These topics will be worked on only if there is time.

2.5. Conclusion

Developing a multi-agent robotic search team for urban search and rescue (uSaR) operations is a multi-disciplinary task. This literature survey serves as a foundation to develop a controller to improve victim search efficiency. The controller must solve a victim search task alongside the local obstacle avoidance, victim identification, and path-following. For this, a hierarchically decoupled planning and control model predictive control (MPC) structure is used. Decoupling the global search problem from the local motion planning controller is expected to lead to similar levels of global optimization while improving the real-time performance required in an uSaR scenario.

First, the victim search mission was described using object detection taxonomy in Chapter 2.2. The distinction between sensor-, motion-, and mission-related influences was made providing a guideline for classifying uncertainties during simulation and testing. The indoor uSaR scenario is established to generally occur in GNSS-denied environments. The global mission itself was found to be an optimization problem, where the prior knowledge of the environment and possible victim distribution could specify it as a probabilistic search or hunting problem, as full coverage is not necessarily required.

Designing the multi-agent search controller requires some knowledge on perception, motion planning, and communication, which is performed in Chapter 2.3. The perception subsystem was selected minimizing financial costs, power usage, and weight. For victim detection, a visual camera using a neural network such as YOLOv3 is combined with a microphone and pyroelectric sensor. Navigation can be done by sensor fusing the visual camera with a GNSS receiver and IMU. Proximity sensors were identified as possible low-cost extensions for obstacle avoidance.

Motion planning algorithms were compared through various facets such as the robustness to uncertainties, compatibility with differential constraints, completeness, and calculation speed. Hierarchically decoupled planning and control, mathematical optimization (among which is the MPC), and reactive planning were identified as few of the better possible solutions. Dynamic obstacle avoidance was considered separately, with a focus on how the obstacles can be modelled.

Multi-agent controllers imply inter-agent interaction. Both explicit and implicit communication was compared to that end. For explicit, event-based communication stood out as a computationally cheap but robust solution, fitting the victim search mission. Other aspects important to a multi-agent system, such as fleet and robot design, failure tolerant design, and human-robot interaction were considered to the extent of finding their effects on the controller.

Using this knowledge, a general controller architecture was set up, where the global controller for the search mission is decoupled from the local controller which concerns itself with victim detection, obstacle

avoidance, and position/attitude control. The controller therefore has a modular design, and with event-based communication is flexible to possible inputs from the communication subsystem. Lastly, the design of the simulation was covered, where a model-in-the-loop approach was chosen for the development of the controller, as is the case for initial designs and prototyping.

Chapter 2.4 concerned itself with MPC, reviewing the background theory, fundamental notions of stability, feasibility, and robustness, and the many variations that can account for different problem and model types. Stability could be achieved through either contraction constraints or Lyapunov stability theory, while robustness is done using structured feedback or norm-minimization of the various possible model states. The variations concerned themselves with the linearity, stochasticity, and effect of multi-agent communication hierarchy on the MPC. Surprisingly, though a non-linear MPC is theoretically slower than a linear MPC due to the non-linear programming requirement, the successive linearization and the implementation can sometimes have a greater impact on the computational speed. Regarding stochastic MPC, non-linear stochastic tube MPC approaches were found as alternatives to sampling- or evolution-based approaches. The MPC still has many parameters, despite the form of the underlying problem. Offline and online tuning (manual vs auto-tuning) methods both exist, including robust tuning approaches which take into account disturbances and uncertainties.

With all the knowledge obtained a clear design plan for the controller could be set up. The literature also showed that the main contribution of the thesis is the hierarchical MPC with real-time dynamic obstacle avoidance. The final result of the literature study comes from decoupling the problem, leading to a clear development plan for a multi-agent motion-planning controller design.

References

[1] Shannon Doocy et al. “The human impact of earthquakes: a historical review of events 1980-2009 and systematic literature review”. In: *PLoS currents* 5 (2013).

[2] Anna Rom et al. “Search without rescue? Evaluating the international search and rescue response to earthquake disasters”. In: *BMJ global health* 5.12 (2020), e002398.

[3] Robin R Murphy. *Disaster robotics*. MIT press, 2014.

[4] Juntong Qi et al. “Search and Rescue Rotary-Wing UAV and Its Application to the Lushan Ms 7.0 Earthquake”. In: *Journal of Field Robotics* 33.3 (2016), pp. 290–321. DOI: https://doi.org/10.1002/rob.21615.

[5] David Lallemant et al. “Post-Disaster Damage Assessments as Catalysts for Recovery: A Look at Assessments Conducted in the Wake of the 2015 Gorkha, Nepal, Earthquake”. In: *Earthquake Spectra* 33.1_suppl (2017), pp. 435–451. DOI: 10.1193/120316eqs222m.

[6] Kevin S. Pratt et al. “CONOPS and autonomy recommendations for VTOL small unmanned aerial system based on Hurricane Katrina operations”. In: *Journal of Field Robotics* 26.8 (2009), pp. 636–650. DOI: https://doi.org/10.1002/rob.20304.

[7] Daniel Drew. “Multi-Agent Systems for Search and Rescue Applications”. In: *Current Robotics Reports* 2 (June 2021). DOI: 10.1007/s43154-021-00048-3.

[8] Sean Grogan et al. “The use of unmanned aerial vehicles and drones in search and rescue operations–a survey”. In: *Proceedings of the PROLOG* (2018).

[9] Yugang Liu et al. “Robotic Urban Search and Rescue: A Survey from the Control Perspective”. In: *J. Intell. Robotics Syst.* 72.2 (Nov. 2013), pp. 147–165. DOI: 10.1007/s10846-013-9822-x. URL: https://doi.org/10.1007/s10846-013-9822-x.

[10] Huihui Sun et al. “Motion Planning for Mobile Robots—Focusing on Deep Reinforcement Learning: A Systematic Review”. In: *IEEE Access* 9 (2021), pp. 69061–69081. DOI: 10.1109/ACCESS.2021.3076530.

[11] Xuesu Xiao et al. “UAV assisted USV visual navigation for marine mass casualty incident response”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 6105–6110. DOI: 10.1109/IROS.2017.8206510.

[12] Francisca Rosique et al. “A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research”. In: *Sensors* 19.3 (2019). DOI: 10.3390/s19030648. URL: https://www.mdpi.com/1424-8220/19/3/648.

[13] Cyril Robin et al. “Multi-robot target detection and tracking: taxonomy and survey”. In: *Autonomous Robots* 40 (Apr. 2016). DOI: https://doi.org/10.1007/s10514-015-9491-7.

[14] Steve Burion. “Human detection for robotic urban search and rescue”. PhD thesis. École Polytechnique Fédérale de Lausanne, 2004.

[15] P. Thanu Thavasi et al. “Sensors and Tracking Methods Used in Wireless Sensor Network Based Unmanned Search and Rescue System -A Review”. In: *Procedia Engineering* 38 (2012). International conference on modelling optimization and computing, pp. 1935–1945. DOI: https://doi.org/10.1016/j.proeng.2012.06.236. URL: https://www.sciencedirect.com/science/article/pii/S1877705812021492.

[16] Ekaterina R. Stepanova et al. “Gathering and Applying Guidelines for Mobile Robot Design for Urban Search and Rescue Application”. In: *Human-Computer Interaction. Interaction Contexts*. Ed. by Masaaki Kurosu. Cham: Springer International Publishing, 2017, pp. 562–581.

[17] Ignacio Martinez-Alpiste et al. “Search and rescue operation using UAVs: A case study”. In: *Expert Systems with Applications* 178 (2021), p. 114937. DOI: https://doi.org/10.1016/j.eswa.2021.114937. URL: https://www.sciencedirect.com/science/article/pii/S095741742100378X.

[18] Jamil Fayyad et al. “Deep Learning Sensor Fusion for Autonomous Vehicle Perception and Localization: A Review”. In: *Sensors* 20.15 (2020). URL: https://www.mdpi.com/1424-8220/20/15/4220.

[19] Da Hu et al. “Detecting, locating, and characterizing voids in disaster rubble for search and rescue”. In: *Advanced Engineering Informatics* 42 (2019), p. 100974.

[20] Danijel Šipoš et al. “A lightweight and low-power UAV-borne ground penetrating radar design for landmine detection”. In: *Sensors* 20.8 (2020), p. 2234.

[21] Ian Sharp et al. “Signal Strength Positioning”. In: *Wireless Positioning: Principles and Practice*. Singapore: Springer Singapore, 2019, pp. 475–504. DOI: 10.1007/978-981-10-8791-2_15. URL: https://doi.org/10.1007/978-981-10-8791-2_15.

[22] Julia Breßler et al. “GNSS positioning in non-line-of-sight context—A survey”. In: *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)*. IEEE. 2016, pp. 1147–1154.

[23] Federico Castanedo. “A review of data fusion techniques”. In: *The scientific world journal* 2013 (2013).

[24] Ren C. Luo et al. “Multisensor Fusion and Integration: A Review on Approaches and Its Applications in Mechatronics”. In: *IEEE Transactions on Industrial Informatics* 8.1 (2012), pp. 49–60. DOI: 10.1109/TII.2011.2173942.

[25] Jorge Pena Queralta et al. “Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision”. In: *IEEE Access* 8 (2020), pp. 191617–191643.

[26] M.G. Mohanan et al. “A survey of robotic motion planning in dynamic environments”. In: *Robotics and Autonomous Systems* 100 (2018), pp. 171–185. DOI: https://doi.org/10.1016/j.robot.2017.10.011. URL: https://www.sciencedirect.com/science/article/pii/S0921889017300313.

[27] Christos Katrakazas et al. “Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions”. In: *Transportation Research Part C: Emerging Technologies* 60 (2015), pp. 416–442. DOI: https://doi.org/10.1016/j.trc.2015.09.011. URL: https://www.sciencedirect.com/science/article/pii/S0968090X15003447.

[28] Chad Goerzen et al. “A survey of motion planning algorithms from the perspective of autonomous UAV guidance”. In: *Journal of Intelligent and Robotic Systems* 57.1 (2010), pp. 65–100.

[29] Peter Tsenkov et al. “A system for 3d autonomous rotorcraft navigation in urban environments”. In: *AIAA Guidance, Navigation and Control Conference and Exhibit*. 2008, p. 7412.

[30] Sebastian Scherer et al. “Flying fast and low among obstacles: Methodology and experiments”. In: *The International Journal of Robotics Research* 27.5 (2008), pp. 549–574.

[31] Sertac Karaman et al. “Sampling-based algorithms for optimal motion planning”. In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[32] David González et al. “A Review of Motion Planning Techniques for Automated Vehicles”. In: *IEEE Transactions on Intelligent Transportation Systems* 17.4 (2016), pp. 1135–1145. DOI: 10.1109/TITS.2015.2498841.

[33] Zvi Shiller et al. “The Nonlinear Velocity Obstacle Revisited: the Optimal Time Horizon”. In: *Guaranteeing Safe Navigation in Dynamic Environments Workshop*. Anchorage, United States, May 2010. URL: https://hal.inria.fr/inria-00562249.

[34] Anahita Jamshidnejad et al. “Adaptive Optimal Receding-Horizon Robot Navigation via Short-Term Policy Development”. In: *15th International Conference on Control, Automation, Robotics and Vision, ICARCV 2018, Singapore, November 18-21, 2018*. IEEE, 2018, pp. 21–28. DOI: 10.1109/ICARCV.2018.8581157. URL: https://doi.org/10.1109/ICARCV.2018.8581157.

[35] Alberto Bemporad et al. “Hierarchical and hybrid model predictive control of quadcopter air vehicles”. In: *IFAC Proceedings Volumes* 42.17 (2009), pp. 14–19.

[36] Alberto Bemporad et al. “Decentralized hybrid model predictive control of a formation of unmanned aerial vehicles”. In: *IFAC Proceedings Volumes* 44.1 (2011), pp. 11900–11906.

[37] Federico Rossi et al. “Review of multi-agent algorithms for collective behavior: a structural taxonomy”. In: *IFAC-PapersOnLine* 51.12 (2018), pp. 112–117.

[38] Alaa Khamis et al. “Multi-robot Task Allocation: A Review of the State-of-the-Art”. In: vol. 604. Springer International Publishing, May 2015, pp. 31–51. DOI: 10.1007/978-3-319-18299-5_2.

[39] Yara Rizk et al. “Cooperative heterogeneous multi-robot systems: A survey”. In: *ACM Computing Surveys (CSUR)* 52.2 (2019), pp. 1–31.

[40] Eduardo Camponogara et al. “Distributed model predictive control”. In: *IEEE control systems magazine* 22.1 (2002), pp. 44–52.

[41] Ramin Rostami et al. “Distributed model predictive control with event-based optimization”. In: *IFAC-PapersOnLine* 50.1 (2017), pp. 8933–8938.

[42] Cai Zhihao et al. “Virtual target guidance-based distributed model predictive control for formation control of multiple UAVs”. In: *Chinese Journal of Aeronautics* 33.3 (2020), pp. 1037–1056.

[43] Bara J Emran et al. “A review of quadrotor: An underactuated mechanical system”. In: *Annual Reviews in Control* 46 (2018), pp. 165–180.

[44] Alireza Abbaspour et al. “A survey on active fault-tolerant control systems”. In: *Electronics* 9.9 (2020), p. 1513.

[45] Cunjia Liu et al. “Hierarchical path planning and flight control of small autonomous helicopters using MPC techniques”. In: *2013 IEEE Intelligent Vehicles Symposium (IV)*. 2013, pp. 417–422. DOI: 10.1109/IVS.2013.6629504.

[46] Yazan Mualla et al. “Agent-based simulation of unmanned aerial vehicles in civilian applications: A systematic literature review and research directions”. In: *Future Generation Computer Systems* 100 (2019), pp. 344–364. DOI: https://doi.org/10.1016/j.future.2019.04.051. URL: https://www.sciencedirect.com/science/article/pii/S0167739X18328462.

[47] Alberto Bemporad et al. “Robust model predictive control: A survey”. In: *Robustness in identification and control*. Springer, 1999, pp. 207–226.

[48] L. Magni et al. “Stabilizing model predictive control of nonlinear continuous time systems”. In: *Annual Reviews in Control* 28.1 (2004), pp. 1–11. DOI: https://doi.org/10.1016/j.arcontrol.2004.01.001. URL: https://www.sciencedirect.com/science/article/pii/S1367578804000021.

[49] Max Schwenzer et al. “Review on model predictive control: an engineering perspective”. In: *The International Journal of Advanced Manufacturing Technology* (2021), pp. 1–23.

[50] Daniel Simon et al. “Reference tracking MPC using terminal set scaling”. In: *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE. 2012, pp. 4543–4548.

[51] Max Basescu et al. “Direct NMPC for Post-Stall Motion Planning with Fixed-Wing UAVs”. In: *CoRR* abs/2001.11478 (2020). arXiv: 2001.11478. URL: https://arxiv.org/abs/2001.11478.

[52] Gianluca Serale et al. “Model predictive control (MPC) for enhancing building and HVAC system energy efficiency: Problem formulation, applications and opportunities”. In: *Energies* 11.3 (2018), p. 631.

[53] Yang Wang et al. “Fast Model Predictive Control Using Online Optimization”. In: *IFAC Proceedings Volumes* 41.2 (2008). 17th IFAC World Congress, pp. 6974–6979. DOI: https://doi.org/10.3182/20080706-5-KR-1001.01182. URL: https://www.sciencedirect.com/science/article/pii/S1474667016400662.

[54] Julian Berberich et al. “Linear tracking MPC for nonlinear systems Part I: The model-based case”. In: *arXiv preprint arXiv:2105.08560* (2021).

[55] Yusuke Igarashi et al. “Mpc performances for nonlinear systems using several linearization models”. In: *2020 American Control Conference (ACC)*. IEEE. 2020, pp. 2426–2431.

[56] Terrence Skibik et al. “An Architecture for Analog VLSI Implementation of Embedded Model Predictive Control”. In: *2018 Annual American Control Conference (ACC)*. 2018, pp. 4676–4681. DOI: 10.23919/ACC.2018.8431320.

[57] Sergey Vichik et al. “Solving Linear and Quadratic Programs with an Analog Circuit”. In: *Computers & Chemical Engineering* 70 (Nov. 2014). DOI: 10.1016/j.compchemeng.2014.01.011.

[58] Richard Oberdieck et al. “Explicit hybrid model-predictive control: The exact solution”. In: *Automatica* 58 (2015), pp. 152–159. DOI: https://doi.org/10.1016/j.automatica.2015.05.021. URL: https://www.sciencedirect.com/science/article/pii/S0005109815002277.

[59] Alberto Bemporad et al. “The explicit linear quadratic regulator for constrained systems”. In: *Automatica* 38.1 (2002), pp. 3–20. DOI: https://doi.org/10.1016/S0005-1098(01)00174-1. URL: https://www.sciencedirect.com/science/article/pii/S0005109801001741.

[60] Thomas Besselmann et al. “Explicit MPC for LPV Systems: Stability and Optimality”. In: *IEEE Transactions on Automatic Control* 57.9 (2012), pp. 2322–2332. DOI: 10.1109/TAC.2012.2187400.

[61] Mina Kamel et al. “Linear vs Nonlinear MPC for Trajectory Tracking Applied to Rotary Wing Micro Aerial Vehicles”. In: *IFAC-PapersOnLine* 50.1 (2017). 20th IFAC World Congress, pp. 3463–3469. DOI: https://doi.org/10.1016/j.ifacol.2017.08.849. URL: https://www.sciencedirect.com/science/article/pii/S2405896317313083.

[62] Jun Yang et al. “Design of a prediction-accuracy-enhanced continuous-time MPC for disturbed systems via a disturbance observer”. In: *IEEE Transactions on Industrial Electronics* 62.9 (2015), pp. 5807–5816.

[63] Sina Sharif Mansouri et al. “Subterranean MAV navigation based on nonlinear MPC with collision avoidance constraints”. In: *arXiv preprint arXiv:2006.04227* (2020).

[64] Ali Mesbah. “Stochastic model predictive control: An overview and perspectives for future research”. In: *IEEE Control Systems Magazine* 36.6 (2016), pp. 30–44.

[65] David Mayne. “Robust and stochastic model predictive control: Are we going in the right direction?” In: *Annual Reviews in Control* 41 (2016), pp. 184–192.

[66] Angelo D Bonzanini et al. “Tube-based stochastic nonlinear model predictive control: A comparative study on constraint tightening”. In: *IFAC-PapersOnLine* 52.1 (2019), pp. 598–603.

[67] Naoya Ozaki et al. “Tube Stochastic Optimal Control for Nonlinear Constrained Trajectory Optimization Problems”. In: *Journal of Guidance, Control, and Dynamics* 43.4 (2020), pp. 645–655.

[68] Riccardo Scattolini. “Architectures for distributed and hierarchical Model Predictive Control - A review”. In: *Journal of Process Control* 19 (2009), pp. 723–731.

[69] Sina Sharif Mansouri et al. “Distributed model predictive control for unmanned aerial vehicles”. In: *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*. 2015, pp. 152–161. DOI: 10.1109/RED-UAS.2015.7441002.

[70] Jan Marian Maciejowski. *Predictive control: with constraints*. Pearson education, 2002.

[71] Jorge L Garriga et al. “Model predictive control tuning methods: A review”. In: *Industrial & Engineering Chemistry Research* 49.8 (2010), pp. 3505–3515.

Part III

Appendices for the scientific article

A

Initial obstacle conditions for simulations

This appendix contains the initial conditions of the static and dynamic obstacles, where PoA stands for Point of Attraction, see Figure 1.3. These were generated using

Table A.1: Obstacle initial conditions: simple simulation 1

Multiplier	Position 1		Velocity 10^{-2}		Acceleration 10^{-1}		PoA 1	
ID	x	y	x	y	x	y	x	y
1	6.50	11.34	-	-	-	-	-	-
2	3.01	0.18	-	-	-	-	-	-
3	4.42	0.95	-	-	-	-	-	-
4	10.20	9.05	-	-	-	-	-	-
5	10.38	4.47	-	-	-	-	-	-
6	0.89	2.51	-	-	-	-	-	-
7	8.22	9.93	-6.08	4.48	4.01	3.70	7.51	10.69
8	6.25	7.73	2.68	-8.06	3.83	5.29	6.21	7.64
9	7.36	2.59	-4.87	6.60	2.22	3.49	6.85	3.02
10	3.28	2.53	-6.90	3.77	2.25	2.21	3.24	1.57
11	4.46	4.72	-3.39	1.76	3.49	2.45	5.19	5.54

Table A.2: Obstacle initial conditions: simple simulation 2

Multiplier	Position 1		Velocity 10^{-2}		Acceleration 10^{-1}		PoA 1	
ID	x	y	x	y	x	y	x	y
1	3.71	11.17	-	-	-	-	-	-
2	7.05	11.54	-	-	-	-	-	-
3	4.30	3.09	-	-	-	-	-	-
4	2.76	3.47	-	-	-	-	-	-
5	7.71	7.25	-	-	-	-	-	-
6	9.87	8.22	-	-	-	-	-	-
7	7.75	8.05	-7.96	4.38	3.91	1.68	8.04	8.99
8	6.23	1.85	-1.12	2.70	4.99	4.15	6.03	2.70
9	3.35	3.23	-0.32	2.38	5.70	2.83	3.24	4.29
10	6.12	8.54	-6.01	3.50	1.82	3.10	5.04	7.91
11	4.20	9.71	-3.74	6.47	3.68	4.40	4.48	10.47

Table A.3: Obstacle initial conditions: simple simulation 3

Multiplier	Position 1		Velocity 10^{-2}		Acceleration 10^{-1}		PoA 1	
ID	x	y	x	y	x	y	x	y
1	8.22	7.58	-	-	-	-	-	-
2	5.22	2.74	-	-	-	-	-	-
3	2.74	3.60	-	-	-	-	-	-
4	10.25	5.73	-	-	-	-	-	-
5	9.67	5.02	-	-	-	-	-	-
6	2.62	2.41	-	-	-	-	-	-
7	5.06	10.90	9.72	-3.38	3.52	5.00	4.97	10.70
8	1.02	3.77	-2.29	-0.77	4.39	3.31	1.33	3.53
9	10.98	5.96	6.23	-9.11	3.81	2.52	11.60	5.99
10	8.59	8.97	-8.07	-6.45	4.85	1.18	8.97	8.13
11	10.75	3.42	-2.98	6.49	2.31	5.80	11.80	3.82

Table A.4: Obstacle initial conditions: simple simulation 4

Multiplier	Position 1		Velocity 10^{-2}		Acceleration 10^{-1}		PoA 1	
ID	x	y	x	y	x	y	x	y
1	10.00	7.26	-	-	-	-	-	-
2	9.31	1.19	-	-	-	-	-	-
3	1.81	7.97	-	-	-	-	-	-
4	11.74	4.41	-	-	-	-	-	-
5	6.82	8.65	-	-	-	-	-	-
6	1.99	5.02	-	-	-	-	-	-
7	1.27	4.77	8.14	3.04	5.47	5.85	1.01	4.46
8	9.51	8.73	1.91	3.80	2.12	2.11	9.05	9.05
9	11.09	5.00	0.54	10.83	4.82	6.38	11.00	5.04
10	4.50	3.05	1.04	8.27	4.02	3.51	4.58	3.91
11	10.55	3.63	7.29	2.74	3.57	2.24	11.20	3.23

Table A.5: Obstacle initial conditions: simple simulation 5

Multiplier	Position 1		Velocity 10^{-2}		Acceleration 10^{-1}		PoA 1	
ID	x	y	x	y	x	y	x	y
1	8.33	3.83	-	-	-	-	-	-
2	7.82	7.93	-	-	-	-	-	-
3	10.80	11.15	-	-	-	-	-	-
4	5.30	1.94	-	-	-	-	-	-
5	9.98	8.94	-	-	-	-	-	-
6	1.18	8.74	-	-	-	-	-	-
7	5.16	8.94	-5.86	-0.23	3.77	2.14	4.11	9.70
8	8.37	6.42	-1.80	-1.43	2.74	1.39	8.79	5.49
9	9.34	2.70	-1.36	-1.80	4.99	1.99	9.03	3.50
10	7.30	3.60	0.67	-4.16	5.22	5.38	7.22	3.99
11	1.17	3.86	-4.05	-0.73	4.46	4.59	1.01	4.51

Table A.6: Obstacle initial conditions: simple simulation 6

Multiplier	Position 1		Velocity 10^{-2}		Acceleration 10^{-1}		PoA 1	
ID	x	y	x	y	x	y	x	y
1	2.82	6.89	-	-	-	-	-	-
2	9.34	3.83	-	-	-	-	-	-
3	5.10	11.25	-	-	-	-	-	-
4	0.32	5.94	-	-	-	-	-	-
5	5.68	0.94	-	-	-	-	-	-
6	10.10	3.25	-	-	-	-	-	-
7	4.27	9.07	4.26	-3.23	2.56	2.14	3.81	8.49
8	9.92	5.42	2.07	-1.23	3.47	1.00	9.50	4.02
9	8.30	3.04	-0.63	1.80	0.96	1.09	9.13	3.61
10	6.17	2.64	-2.30	-3.16	2.85	6.11	6.72	1.98
11	4.01	9.59	0.67	4.06	1.16	3.40	5.22	10.26

Table A.7: Obstacle initial conditions: simple simulation 7

Multiplier	Position 1		Velocity 10^{-2}		Acceleration 10^{-1}		PoA 1	
ID	x	y	x	y	x	y	x	y
1	4.18	10.00	-	-	-	-	-	-
2	2.51	4.70	-	-	-	-	-	-
3	8.16	8.81	-	-	-	-	-	-
4	4.27	3.49	-	-	-	-	-	-
5	5.26	3.51	-	-	-	-	-	-
6	10.91	3.56	-	-	-	-	-	-
7	5.68	1.15	-9.39	0.23	10.05	7.29	5.55	1.13
8	10.52	1.32	-6.17	1.43	0.32	5.52	10.69	1.87
9	3.68	10.59	4.24	1.80	1.95	1.96	3.85	10.28
10	8.93	11.09	4.36	4.16	3.56	1.25	8.86	10.11
11	6.20	10.03	-0.56	0.73	7.89	4.35	5.68	9.06

Table A.8: Obstacle initial conditions: simple simulation 8

Multiplier	Position 1		Velocity 10^{-2}		Acceleration 10^{-1}		PoA 1	
ID	x	y	x	y	x	y	x	y
1	3.75	0.65	-	-	-	-	-	-
2	1.29	4.50	-	-	-	-	-	-
3	8.34	4.40	-	-	-	-	-	-
4	8.90	3.41	-	-	-	-	-	-
5	4.76	10.97	-	-	-	-	-	-
6	4.16	7.92	-	-	-	-	-	-
7	2.73	3.06	-5.46	-0.75	3.09	2.55	2.62	2.49
8	9.12	6.20	2.09	5.46	1.88	1.79	9.23	6.74
9	6.02	9.10	4.11	4.78	3.22	4.21	5.45	9.84
10	4.96	3.54	10.09	2.39	1.76	4.03	5.63	3.79
11	1.18	7.68	-7.30	-8.81	5.57	4.43	1.16	6.88

Table A.9: Obstacle initial conditions: simple simulation 9

Multiplier	Position 1		Velocity 10^{-2}		Acceleration 10^{-1}		PoA 1	
ID	x	y	x	y	x	y	x	y
1	4.44	5.65	-	-	-	-	-	-
2	9.13	11.27	-	-	-	-	-	-
3	5.12	4.53	-	-	-	-	-	-
4	1.56	10.41	-	-	-	-	-	-
5	10.22	10.47	-	-	-	-	-	-
6	0.26	10.49	-	-	-	-	-	-
7	2.78	1.16	2.94	1.54	3.53	3.32	2.62	1.01
8	10.54	8.90	5.94	4.09	3.80	4.17	11.59	9.87
9	1.61	2.57	-0.09	-3.83	5.49	7.21	1.01	2.32
10	9.36	1.80	1.69	7.09	2.37	2.60	8.83	1.62
11	5.78	3.48	-9.84	2.69	3.23	4.20	5.47	4.56

Table A.10: Obstacle initial conditions: simple simulation 10

Multiplier	Position 1		Velocity 10^{-2}		Acceleration 10^{-1}		PoA 1	
ID	x	y	x	y	x	y	x	y
1	6.90	8.48	-	-	-	-	-	-
2	11.36	10.66	-	-	-	-	-	-
3	8.66	11.28	-	-	-	-	-	-
4	7.62	7.50	-	-	-	-	-	-
5	2.38	3.78	-	-	-	-	-	-
6	6.73	1.03	-	-	-	-	-	-
7	7.24	9.51	5.97	0.07	2.52	3.50	6.61	10.55
8	5.08	5.25	2.64	-1.22	6.05	1.59	4.70	4.71
9	10.52	10.20	0.72	-5.97	5.81	4.96	11.09	9.48
10	7.62	1.24	7.00	4.43	1.87	3.17	7.71	1.69
11	7.30	4.70	5.46	-1.41	4.89	4.52	7.97	5.64

Table A.11: Obstacle initial conditions: cluttered scenario

Multiplier	Position 1		Velocity 10^{-2}		Acceleration 10^{-1}		PoA 1	
ID	x	y	x	y	x	y	x	y
1	6.00	2.85	-	-	-	-	-	-
2	1.64	2.42	-	-	-	-	-	-
3	6.80	8.42	-	-	-	-	-	-
4	4.41	9.59	-	-	-	-	-	-
5	10.98	0.89	-	-	-	-	-	-
6	1.83	8.56	-	-	-	-	-	-
7	0.72	1.90	-	-	-	-	-	-
8	7.15	3.15	-	-	-	-	-	-
9	7.71	9.57	-3.59	-1.09	3.38	2.65	8.54	10.28
10	2.96	11.94	1.78	-3.77	3.61	6.34	2.05	11.99
11	5.06	10.57	-2.40	10.95	4.72	6.06	6.05	10.85
12	6.27	9.30	-6.92	8.20	2.12	3.97	7.19	9.79
13	8.62	6.71	-8.49	-1.93	3.19	2.93	9.37	6.95
14	10.15	6.58	-7.70	-7.05	5.60	3.32	10.30	5.68
15	11.61	5.75	-4.77	-9.62	3.53	3.18	11.98	4.83
16	8.89	8.26	9.39	-8.80	4.71	5.40	8.34	7.58