NTNU

Kunnskap for en bedre verden

MPC-based trajectory planning and COLREGs-aware collision avoidance

Author:

Erlend Hestvik

TTK4551 - Engineering Cybernetics, Specialization Project

December 2021

Preface

This project report is the culmination of the research and work done in preparation of my master thesis during the autumn of 2021. This report is nothing groundbreaking in and of itself, but rather an exploration of ideas. I hope that by combining my ideas with the work of prior researchers, some new progress can be made.

I would like to thank Morten Breivik and Emil Thyri for their great counseling and guidance throughout the duration of the project. Their assistance has been invaluable.

I would also like to thank Olex AS for letting me use one of their computers and software to record two days worth of AIS data.

Erlend Hestvik, 20.12.2021

Abstract

Combining trajectory planning and collision avoidance methodology is no mean feat. This study dives into the world of trajectory planning and collision avoidance algorithms developed for Autonomous Surface Vehicles in recent years and offers a new algorithm that combines both problems into one COLREGs-aware Model Predictive Control-based algorithm. To overcome some of the normal shortcomings of MPC I also employ an assumption that I am able to utilize historical traffic data sourced from AIS to generate a nominal reference path for the Own-Ship, which is the vessel that we control, to follow. In the end the algorithm ends up as a hybrid MPC and Line of Sight guidance algorithm. The proposed trajecotry planning algorithm is tested in six simulated scenarios to examine path finding performance and COLREGs compliance. The results show that the algorithm is functional, but with room for improvements. With modifications to the placement of constraints in NLP formulation better COLREGs performance could be achieved. Proposals for future work to improve the performance of the trajectory planning algorithm are also made.

Contents

List of Figures

1 Introduction

This chapter details my motivations for this project, and explains the problem at hand. It also includes an overview of relevant previous research and work done in the field, and lists the contributions of my own work. Finally, the chapter includes an outline of the rest of the paper.

1.1 Motivation

I find autonomous technology fascinating, instructing a machine to not only operate independently, but also make its own decisions based on the information available. In my head there is no reason why a machine can't do the same thing a human can do, for what is the brain if not a hyper complicated I/O machine? My brain is capable of gathering information, analyzing it, and responding based on prior experiences, all without any form of formal "programming" to understand what anything is. I believe that a sufficiently complicated machine will one day be able to do the same, and I will happily do my small part to contribute to the progress that will hopefully get us there.

Autonomous vehicles have become a very hot topic in recent years, and for a good reason. In my opinion this is *the* frontier for machine intelligence. Currently cars are receiving most of the attention, and the progress for self-driving cars has been incredible. And while self-driving cars are great, I think self-driving boats is a criminally underrated area of interest. Not only because of how much we rely on ocean traffic for shipment of goods, but also because the sea is a volatile environment to operate in. I believe that significant development in autonomous surface and underwater vehicles will be instrumental for pushing the limits of autonomous systems.

One day I believe that machines will truly be able to perform any task just as well as humans, I can only hope that my contributions will spare me from the robot uprising Hollywood has warned us about, see Figure 1.

1.2 Previous Work

In this section, we present some relevant previous work in the field of trajectory planning and collision avoidance for autonomous surface vehicles (ASVs).

First, (Huang et al. 2020) covers a lot of material related to the ongoing develop-

Figure 1: The future might become interesting. Image taken from MARKETWATCH PHOTO ILLUSTRATION/EVERETT COLLECTION.

ment of collision avoidance for ASVs. The paper outlines multiple ways of modelling ship motion, the different methods for target ship (TS) tracking, conflict detection and risk assessment. The study also includes about some of the algorithms developed for automatically moving out of the way to avoid a collision. The paper includes both what has been done, and a review / discussion of the different methods to highlight strengths and differences between methods for each topic. Furthermore, the paper contains a high number of relevant reference.

In a similar vein, (Vagale et al. 2021) conducts a thorough study of the state of the art within trajectory planning for ASVs. This paper is more focused on the current state of ASVs. Providing details relating to their levels of autonomy, regulations, and safety concerns. The paper also examines how trajectory planning and collision avoidance can be combined into one path planning package.

A comprehensive study of different methods for trajectory planning and collision avoidance is presented in the works of (Loe 2007). In this work, Loe evaluates multiple different trajectory planning algorithms and simulates them to compare their performance and behaviour. This report is a great read for getting an overview of the methods that can be employed when developing a trajectory planning algorithm, and the simulated scenarios gives an indication of the strengths and weaknesses of several commonly applied methods.

In (Eriksen and Breivik 2017), an approach to trajectory planing for ASVs through a nonlinear MPC approach is presented. The work describes in detail the mathematics behind MPC than I do in my study, and also compares two different objective functions and does a much more thorough evaluation of the feasibility of using MPC in real time.

In terms of COLREGs-aware algorithms, the work of (Woerner 2016) and (Tam and Bucknall 2010) are highly relevant. The work presented by (Woerner 2016) is very substantial, however its Chapter 4.5 "Rule-Specific Algorithms and Considerations" that is relevant for the work presented in this report. The chapter present mathematical approaches to evaluation criteria for the COLREGs situation assessment. In (Tam and Bucknall 2010), a more practical method for COLREGs situation assessment is described in natural language. Together they complement each other to create a great intuition for how to assess the entry criteria for COLREGs.

A paper by (Schöller et al. 2021) explores the prediction of TS trajectories. Through a method combining historical AIS heatmaps and a long short-term memory network the model is able to correctly anticipate the future trajectory of a vessel. The work shows that the future trajectory of TSs can be anticipated more accurately than the more commonly used method of assuming that TSs will maintain their current speed and course.

1.3 Problem Description

There are multiple challenges when designing trajectory planning and collision avoidance software. One of the biggest challenges in the field is creating a COLREGs-aware solution that is able to detect, analyze and resolve encounters fully autonomously. When it comes to trajectory planning there is a challenge in creating an algorithm that is both robust in its path finding capabilities and also able to react to changes in the environment while still being computationally efficient. Combining both into one package might be a job suited for nonlinear optimization, but pure nonlinear MPC usually encounter problems with getting trapped in locally optimal solutions due to the complexity of the problem.

1.4 Contributions

My work contributes the following:

- A cursory look at the benefit of combining historical traffic data with a COLREGs-aware

MPC-based trajectory planning and collision avoidance algorithm package.

- A proposal for how historical traffic data could be used in combination with an MPC-based trajectory planner by way of making a hybrid algorithm that uses Line of Sight guidance to parameterize the historical traffic data into a reference for the nonlinear optimization problem.
- Some simulation results of the propsed algorithm to indicate the performance of the methodology, as well as some suggestions as to how the performance could be improved in future work.

1.5 Outline

The remainder of this project report is structured as follows: Chapter 2 is an introduction to the technology, theory, and concepts touched on in the report. Chapter 3 details the Trajectory planning and collision avoidance algorithm I've put together. Chapter 4 contains an overview of the simulator, my results and a discussion of the performance. Lastly, chapter 5 concludes my study and outlines some potential future work.

2 Background

This chapter presents relevant background theory. In particular, it provides an introduction to utilizing AIS data for creating traffic patterns, trajectory planning for ASVs, model predictive control, and the COLREGs.

2.1 Traffic Pattern

There are no lanes on the ocean, but that doesn't mean traffic is random. A human navigator would use all the tools available in order to perceive and predict any possible obstacle their vessel might encounter. In order for an ASV to operate in the same environment, it must be able to do the same.

In an overview of current trajectory planning and collision avoidance methods for ASVs by (Huang et al. 2020) it is seen that most current models for predicting TS future trajectory rely on a simple linear interpolation of the traffic surrounding their vessel. That method is far too simplistic to accurately predict anything but the simplest maneuvers, it also neglects to use any of the tools a navigator would have at their disposal. One key tool is the Automatic Identification system (AIS). The AIS gives us information about a vessel's speed over ground, course over ground, position and rate of turn. A paper by (Schöller et al. 2021) shows that AIS data can be used to create a more accurate prediction method.

Another way AIS data can be useful is by generating a "traffic pattern" for the area we're interested in working in. AIS sends updates on all vessels in intervals of two to ten seconds. By storing all data received over a couple of days we can create an image of the traffic pattern in an area, see Figure 2. This information can either be used as a heatmap of where we are likely to encounter other vessels, or it can be used to generate commonly travelled routes which can be apllied as input to the proposed trajectory planning algorithm.

Compiling this database would require a lot of work on its own, it's mildly speaking a lot of data to crawl through. One can not simply feed a trajectory planning algorithm all this data, it would be overwhelming and yield no results. Instead we have to sift through the data and extract the information that is useful for the algorithm. Which parts of the data that are useful would ultimately depend on the specific trajectory planning algorithm used. However I have a handful of suggestions for what is useful information,

Figure 2: Traffic pattern from two days of AIS data logging using OLEX software.

and how one could extract it.

First suggestion is to merge trajectories that are nearby each other into singular paths with a few key waypoints for turns that we expect everyone travelling along the trajectory to traverse through, see Figure 3.

Another suggestion is to manually identify sources and sinks, and then extract a handful of paths for each possible route. The TSs are then tagged as following one and a simple straight path following algorithm with weak gain is used to simulate the ship's path. If the TS deviates from the tagged path, a new path is assigned and a new prediction is made.

2.2 Trajectory Planning Algorithm

Optimal control problems are generally computationally intensive, and have some major drawbacks when the problem is concave. Luckily neither of these problems are deal breakers for our purposes. Due to the inherent inertia of naval navigation it's okay to use a slower algorithm when planning ahead. When looking ahead a minute or two we don't need to update our path multiple times every second.

For the second problem we can utilize the same traffic patterns as we use for other agents to generate an initial guess for the nonlinear solver. This guess will alleviate the "concavity" problem that creates local minima which often cause the solver to get stuck.

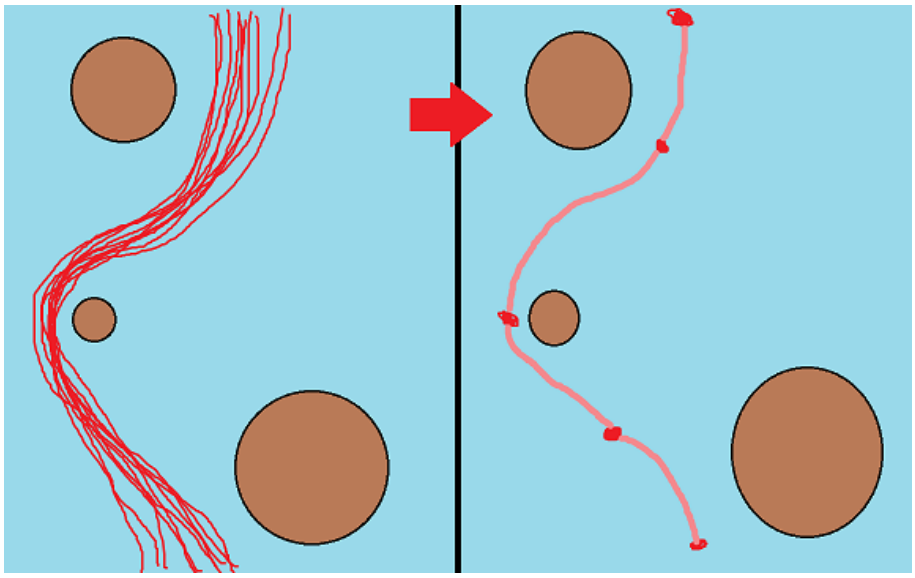

Figure 3: Averaging out the traffic data to a singular path.

2.2.1 Optimal Control Problem

The constrained optimal control problem can be formulated as

$$\begin{aligned} \text{Minimize} \quad & \mathbf{F}(\boldsymbol{\eta}(t), \mathbf{u}(t)) && \text{(1a)} \\ \text{subject to} \quad & \dot{\boldsymbol{\eta}}(t) = \mathbf{L}(\boldsymbol{\eta}(t), \mathbf{u}(t)) && \text{(1b)} \\ & \mathbf{h}(\boldsymbol{\eta}(t), \mathbf{u}(t)) \leq \mathbf{0} && \text{(1c)} \\ & \boldsymbol{\eta}(t_0) = \boldsymbol{\eta}_0 && \text{(1d)} \end{aligned}$$

where $\mathbf{F}$ is the objective function, $\boldsymbol{\eta}(t)$ is the position and attitude trajectory of the vehicle, $\mathbf{u}(t)$ is the control input trajectory and $\mathbf{L}$ is the kinematic model of the vehicle. Though the OCP can be solved the way it is set up in (1) it is more pratical to discretize it into a nonlinear program (NLP). With a method called direct multiple shooting, both state and control input are defined as decision variables, the NLP with N control intervals is:

$$\begin{aligned} \min_{\boldsymbol{\omega}} \quad & \mathbf{F}(\boldsymbol{\omega}) && \text{(2a)} \\ \text{subject to} \quad & \boldsymbol{\omega}_{lb} \leq \boldsymbol{\omega} \leq \boldsymbol{\omega}_{ub} && \text{(2b)} \\ & \mathbf{g}_{lb} \leq \mathbf{g} \leq \mathbf{g}_{ub} && \text{(2c)} \end{aligned}$$

where $\boldsymbol{\omega}$ is a vector of decision variables with $\boldsymbol{\omega}_{lb}$ & $\boldsymbol{\omega}_{ub}$ as lower and upper bounds for each state and control input. Also, $\mathbf{g}$ is a vector of constraint functions that includes both equality and inequality constraints. The vectors $\mathbf{g}_{lb}$ and $\mathbf{g}_{ub}$ are the lower and upper bounds for each constraint function, and the value of these bounds dictates whether or not the constraint is an equality or inequality. To solve the discretized system we must employ an integrator function like for example Runge-Kutta 4. I will admit, I do not understand Runge-Kutta well enough to give it a proper explaination, please read (Eriksen and Breivik 2017) for a more indepth look at how RK is used and why.

The objective function $\mathbf{F}$ can be definied in multiple different ways, this is the heart of the optimal control problem and the form of the function will dictate the behaviour of the ship. If we define the objective as

$$\mathbf{F}(\boldsymbol{\eta}(t), \mathbf{u}(t)) = (\boldsymbol{\eta} - \boldsymbol{\eta}_{ref})' * \boldsymbol{\alpha} * (\boldsymbol{\eta} - \boldsymbol{\eta}_{ref}) \tag{3}$$

where $\boldsymbol{\alpha}$ is some tuning parameter. We end up with a behaviour where the ship only cares about following its position and attitude reference and disregards any other factor that you might be interested in optimizing for. By including additional terms such as for example speed over ground or desired turn rate, and multiplying each term with a tuning parameter, you can fine tune the behaviour to the one you want.

2.2.2 Line of Sight Guidance

Line of Sight (LOS) guidance is a guidance method that uses two fixed reference points to create a path. The cross-track error y_e^p between current position and the path is then used in the feedback for desired course angle such that

$$\chi_d = \pi_p - tan^{-1}(K_p Y_e^p) \tag{4}$$

where χ_d is the desired course angle, π_p is the angle of the path relative to north, and K_p is proportional gain usually parameterized in terms of the lookahead distance. After following the path to the end reference point the references swap so that a new line is formed to another point further ahead. LOS guidance law is well described in Fossen 2011.

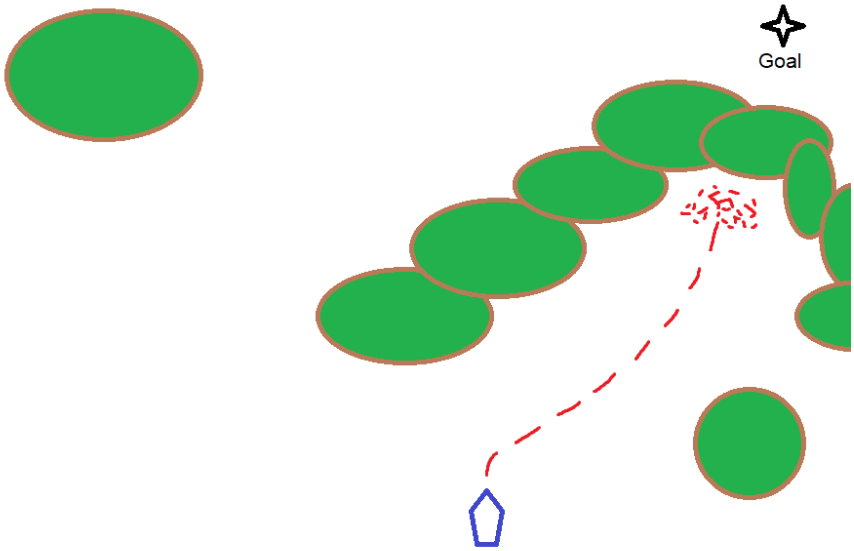

Figure 4: Due to the layout of the shoreline a local minimum is formed near the goal, causing the simple OCP trajectory (red line) to get stuck.

2.2.3 Model Predictive Control

Model predictive control (MPC) is a control method where the nonlinear optimal control problem is combined with a finite horizon iterative model of the vessel. When the nonlinear OCP is solved we take the first state and control input from the optimal solution and apply it to our model, the rest of the optimal solution is discarded. This may sound like a waste of processing power, but this method is what allows the vessel we're controlling to react to its environment. If we were to use the whole optimal state and input sequence from the OCP we'd be locked in to that trajectory. By only using the first instance from the optimal solution and rerunning the nonlinear OCP problem afterwards we can account for unexpected changes in the environment.

2.3 Utilizing Traffic Data

Now that we have a set of "crowd sourced" trajectories courtesy of our AIS, or other data collection method, and the algorithm for which to calculate our own trajectory. The task at hand is to combine these two into a process that's better than the sum of its parts. One of the common downfalls when using nonlinear optimal control for trajectory planning is that the optimal solution might converge on a locally optimal solution that is not desirable. For example the vessel could get stuck in an area formed by a concave land formation. In such a case the optimal control problem deems it unoptimal to leave

the concave area because it cannot "see" that there is a more optimal solution further away, see Figure 4.

If we introduce historical traffic data into the situation, we might see where other vessels that are passing by usually traverse. If we parameterize the traffic data into a set of waypoints so that they can be used as reference for the OCP the generated trajectory would be able to avoid the undesired locally optimal solution and instead find the way to the goal, see Figure 5.

Another way the traffic data might prove useful is in predicting the trajectory of other vessels in the area. Such a prediction would be extremely helpful for risk assessment and collision avoidance purposes. If the predctive power of historical traffic data is good enough we could potentially resolve risky situations before they happen.

2.4 Collision Avoidance

Collision avoidance is a multi-step process involving awareness, planning and action. Traditionally the officer on watch would assess a given situation, plan an action based on training, education, experience, and then order the appropriate maneuver. For an autonomous system the first and last step are easily implemented, however mapping appropriate action based on the situation is a very complex problem. The rules of conduct at sea are often ill-defined or written in natural language, meaning every implementation of autonomous collision avoidance will be different. This is a big hurdle for ASVs because one of the key aspects of collision avoidance at sea is intent and expectations. Clearly showing our own intent and observing the intent of others sets expectations for how each ship in a given situation should act, and if the expectations don't match the risk of collision increases. Luckily there is a standard to base our collision avoidance algorithm on, in 1972 the International Maritime Organization published their "International Regulations for Preventing Collisions at Sea", or COLREGs for short. As long as we make our autonomous vessel COLREGs compliant then in theory it should avoid conflict with all other COLREGs compliant vessels at sea. For vessels that are not COLREGs complaint a contingency plan of emergency maneuvers would be needed, and as technology improves autonomous vessels would ideally be able to clearly communicate their intent digitally and directly to other vessels.

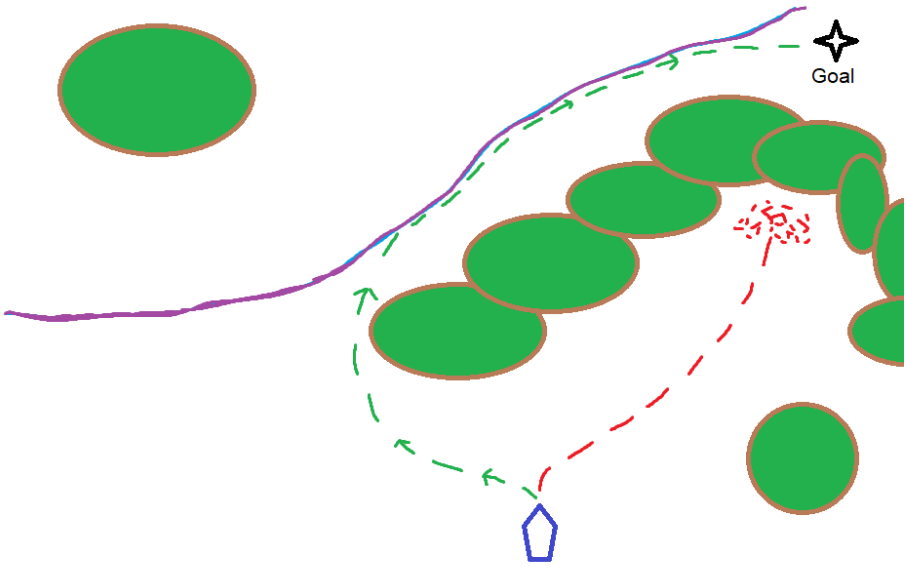

Figure 5: With the help of the traffic pattern (purple line), the vessel is able to navigate along the green line past the obstacle, and get to its goal.

2.4.1 COLREGs

COLREGs is a set of 38 rules designed to standardize the appropriate action to take when encountering other vessels at sea. In their own words, the rules apply to all vessels upon the high seas, but the rules do NOT supersede special case rules made by Governments of any State. Basically the COLREGs rule is the general set of rules to fall back on if there are no other special case rules that apply. Special rules can for example be how certain harbours, rivers, lakes, etc demand a different behaviour due to local constraints and restrictions. It's important to keep in mind that COLREGs are the general case and not always the superior ruling, but special cases will not be considered further in this paper.

The rules are concerned with various aspects of collision avoidance, there are rules for safe speeds, look-out, risk assessment and so on. While all rules are equally important, for our purposes we are mostly concerned with the rules regarding steering when overtaking, approaching head on, and crossing: rule 13-15. (Cockcroft and Lameijer 2012)

Rule 13: Overtaking

Vessels overtaking any other shall keep out of the way of the vessel being overtaken. Vessel shall be deemed to be overtaking when coming up with another vessel from a direction more than 22.5 degrees abaft her beam. When a vessel is in doubt as to whether she is overtaking another, she shall assume that this is the case. Any subsequent alteration of

the bearing between the two shall not make the overtaking vessel a crossing vessel.

Rule 14: Head-on

When two power-driven vessels are meeting on reciprocal or nearly reciprocal courses so as to involve risk of collision. Each shall alter her course to starboard. Such a situation shall be deemed to exist when a vessel sees the other ahead or nearly ahead. When a vessel is in any doubt as to whether such a situation exits she shall assume that it does exists and act accordingly.

Rule 15: Crossing

When two power-driven vessels are crossing so as to involve risk of collision, the vessel which has the other on her own starboard side shall keep out of the way and shall, if the circumstances of the case admit, avoid crossing ahead of the other vessel.

Rule 16: Give-way Vessel

Each vessel which is directed to keep out of the way of another vessel shall, so far as possible, take early and substantial action to keep the way clear.

Rule 17: Stand-on Vessel

When one of two vessels is to keep out of the way the other shall keep her course and speed. The latter vessel may however take action to avoid collision by her maneuver alone, as soon as it becomes apparent to her that the vessel required to keep out of the way is not taking appropriate action in compliance with these rules. When, from any cause, the vessel required to keep her course and speed finds herself so close that collision cannot be avoided by the action of the give-way vessel alone, she shall take such action as will best aid to avoid collision.

2.4.2 Situation Assessment

The COLREGs are, mildly speaking, not automation friendly. The language is convoluted and made for humans who are already familiar with maritime terminology. So let's go through and deduce how our autonomous vessel should behave to achieve COLREGs compliance. The COLREGs outline three situations, which add up to a total of 6 variations:

- Own-Ship is meeting a vessel head on.
- Own-Ship is overtaking a vessel.

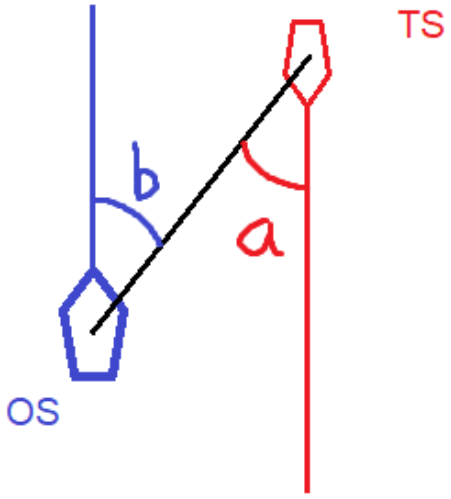

Figure 6: Relative bearing of OS and TS towards each other respectively. Here, b maps from $[0°, 360°]$ while a is the smallest signed angle mapped from $[-180°, 180°]$.

- Own-Ship is being overtaken.
- Own-Ship is crossing and shall stand-on.
- Own-Ship is crossing and have to give-way.
- The situation is safe.

The last one is not really a situation per se, but it's important that a classification is held fixed until it's properly resolved to prevent flip-flopping between behaviours. There are three important aspects to consider when evaluating which COLREGs situation the OS is in: distance to the TS, relative bearing of both ships, relative speed of both ships. When all these are known we can quite easily assess which situation we are in. For a visual guide to relative bearing see Figure 6. First consider the distance between our OS and TS, if the distance is less than some threshold we should consider ourselves to be in a situation. Next, consider the relative bearing between OS and TS, we can define angle intervals to classify regions in which the TS finds itself in. The threshold for each interval and how many you want is up for interpretation, except when considering overtaking, which states that "Vessel shall be deemed to be overtaking when coming up with another vessel from a direction more than *22.5* degrees abaft her beam". This means we must have a region defined for ships ahead of us with a relative bearing between -25.5 and 22.5 degrees. With the regions defined, consider the angle of relative bearing from TS to OS, similar regions can be defined for where our OS is in relation to the TS as seen in Figure 7. Lastly, consider the relative speeds of the vessels, if the speeds are such that

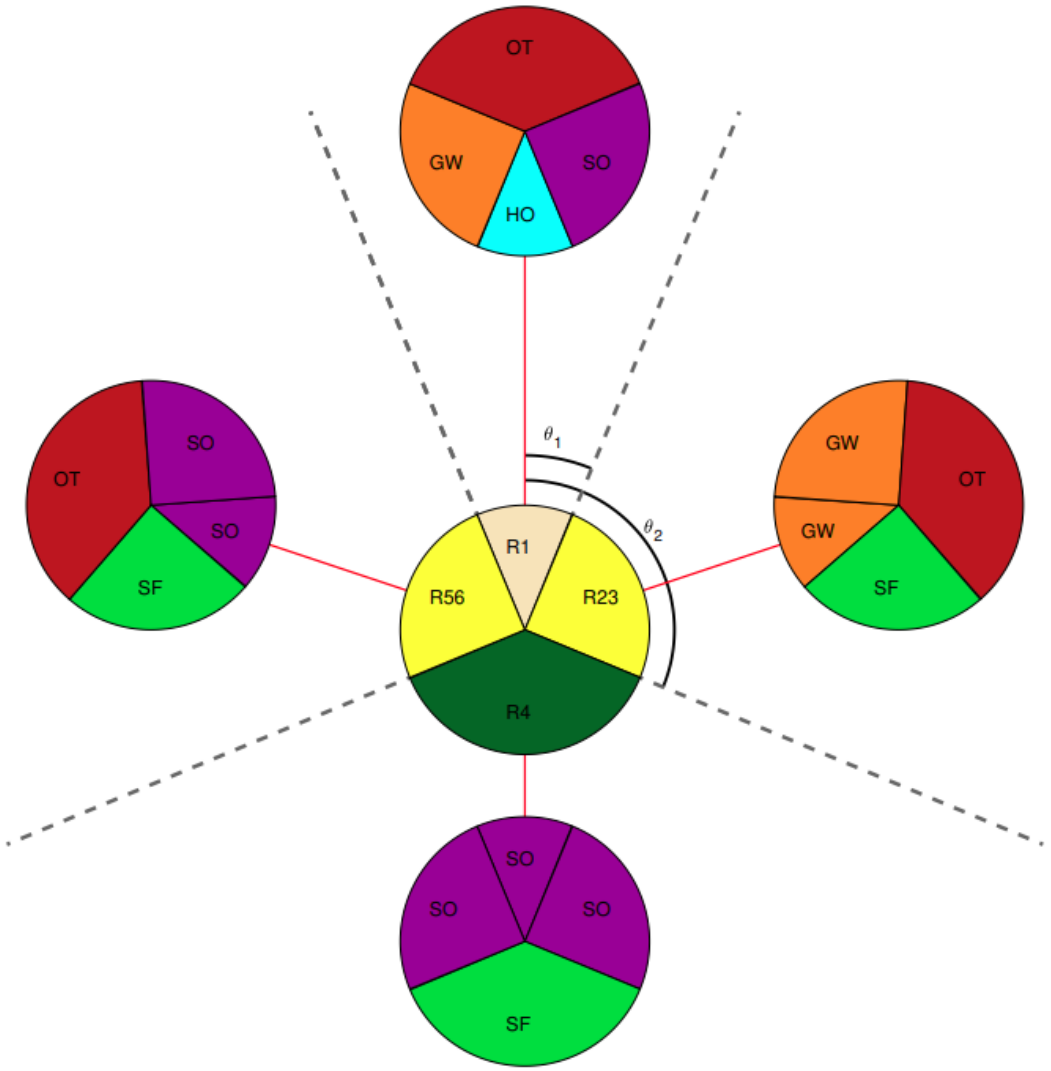

Figure 7: Assigning COLREGS flag: with OS in the center we can place the TS in one of four regions. Similarly the relative bearing from TS to OS can be assigned regions with region 1 pointed directly at the OS and the rest following in a clockwise rotation. Courtesy of Emil Thyri.

the distance between the two ships is increasing we can consider it a safe situation. An encounter that at some point has been classified as something other than safe, can be considered to be concluded when the encounter is reclassified as safe.

3 Trajectory Planner and Collision Avoidance

3.1 Assumptions and Preliminaries

The method proposed in this chapter requires a handful of assumptions to be made, and some prerequisites are taken as given. These decisions were made to contain the scope of the project.

First, no physics based model is used for any of the vessels involved. Under normal circumstances and for a full implementation, the results from the trajectory planner would need to be fed through a reference filter and then passed into a vessel motion control system, where the control actions are applied to a model of the vessel dynamics. Instead we simply assume that the vessel we're working with is capable of following the optimal trajectory, and so the results from the trajectory planner are used to directly update the state of the vessel. This assumption also justifies not taking currents or wind disturbances into account, since a well optimized controller for a properly actuated vessel is expected to be able to deal with a reasonable amount of disturbances.

Secondly, the vessel we're working with has an adequate sensor suite setup to detect all obstacles in the area. An autonomous ship cannot account for obstacles it doesn't know about, and so we work with the assumption that our vessel is sufficiently geared to detect and track everything it needs to know about. If the vessel is not capable of detecting every obstacle on its own, external inputs to fill in the gaps in detection is necessary for the trajectory planner to function properly.

Thirdly, historical traffic data for the area we're interested in working within is readily available and the task of converting this data into a traffic pattern that can be used to predict other vessels is already done. It is not strictly necessary for this data to be sourced from AIS; LIDAR and cameras could also work.

3.2 Initial Conditions

There are a handful of things we must take care of before CasADi can solve the NLP problem. First we need to decide on a time horizon, the horizon is how far into the future the program will consider when calculating an optimal solution. The longer the horizon the more information will be considered, but longer horizons are computationally more expensive. The horizon has other implications for the solution, but we'll get to those

when we discuss solving the NLP problem. In addition to a horizon we need to decide how often a new control instance is calculated, this has less implications for the control problem, but faster control intervals are of course more computationally expensive.

With those out of the way we can move on to the actual interesting bits. We'll want to grab our OS's current position in North and East and put it into an easily accessible variable for later, then we need to know what our reference trajectory will be. This is the reference trajectory generated with the help of a traffic pattern to see where other ships usually go. There are many ways to generate such a reference trajectory but with our assumptions made above we will simply use a LOS guidance law with the same time step length as this algorithm. The output from the LOS guidance law gives us a series of waypoints and velocities for each time step, perfect for what we need. Any other parameterization of the traffic pattern can be made, the only important thing is that the step lengths must be equal.

3.2.1 Preparing Static Obstacles

Static obstacles are defined as n-point polygons in a 2xM matrix, where each obstacle has n columns and each obstacle is separated by a column of NaN. For our NLP problem we need to interpolate every line in the polygon into a set of points that we can use as circular constraints. To do this a subroutine is run before the main loop that iterates through every column in the static_obs matrix and expands it by inserting intermediate columns to fill the gaps between two points. The density of these additional points is based on the distance between the two points we're interpolating. Bigger distance means we need to use less density to prevent overloading our NLP problem with unnecessary amounts of constraints as these interpolated points add up very quickly. The algorithm used to interpolate the Static_Obs polygons into a set of points is described by Algorithm 1.

3.2.2 Preparing Dynamic Obstacles

Every TS is considered its own obstacle, and they are all handled the same way. With the assumption that we can predict where TSs will travel, we generate their projected path forward with a LOS guidance law. The resulting trajectory will be used later on when generating constraints. Alternatively we can assume that every TS will maintain their current course and speed and project their trajectory as a straight path. In addition

Algorithm 1 Pseudocode: Interpolate exterior lines of static obstacle polygons

Input: *Static_obs* ▷ Static_obs is a 2xM matrix of NaN separated polygons

$k \leftarrow 1$
for $i \leftarrow 1, M$ **do**
  if Static_Obs(1,i) is not NaN and Static_Obs(1,i+1) is not NaN **then**
    $Dist \leftarrow Static\_Obs(:, i+1) - Static\_Obs(:, i)$
    $X \leftarrow sign(Dist) * t$ ▷ t is decided based on desired point density
    $Interpolated(:, k) \leftarrow Static\_Obs(:, i)$
    $k \leftarrow k + 1$
    $j \leftarrow 1$
    while $dist \geq -1$ **do**
      $Interpolated(:, k) \leftarrow Static\_Obs(:, i) + j * X$
      $k \leftarrow k + 1$
      $j \leftarrow j + 1$
      $Dist \leftarrow Dist - X$
    end while
  end if
end for
return *Interpolated* ▷ Interpolated is a 2xM_2 matrix of points covering the exterior of Static_Obs

to projecting a trajectory for each TS we must also know what COLREGs situation to assign to each encounter. There can be multiple encounters happening simultaneously so each TS is assigned its own flag that denotes which COLREGs situation the encounter is classified to be. It's important that these flags remain consistent across multiple iterations of the trajectory planning algorithm, so once an encounter has been classified we do not overwrite the flag unless the situation is deemed to be safe again. If there is no flag set we run the algorithm described by Algorithm 2.

The final step of setting up the initial conditions is instancing the NLP problem for CasADi, while I will explain how it's set up for my algorithm, it's highly recommended that you read (Andersson et al. 2019) if you want to fully understand how CasADi works. Initialize all variables as SX.sym and organize them into column vectors; in my case two X states for north and east position, two inputs U for velocity in north and east respectively, and a reference for each state and input. Next we need to define our $\dot{x}$ and objective function. With the assumptions outlined above I'm choosing a non-physical model for $\dot{x}$ and a very simple objective.

$$\begin{aligned} \dot{x} &= u \\ L &= (x - x_{ref})' * a * (x - x_{ref}) + b * (u' * u - u'_{ref} * u_{ref})^2 \end{aligned} \tag{5}$$

With a and b being adjustable weights. We then make the system into a CasADi function, this becomes our continuous time dynamics, the discrete time dynamics are created by the RK4 method shown in Algorithm 3.

Algorithm 2 COLREGs assessment algorithm

Input: η_{OS}, η_{TS}

$phi_1 \leftarrow 22.5$

$phi_2 \leftarrow 112.5$ ▷ phi1 should be 22.5, phi2 is tunable based on rule interpretation

$b_0 \leftarrow$ Relative bearing from OS to TS

$b_{0\_180} \leftarrow$ Smallest Signed Angle of b_0 ▷ Map b0 to [-180, 180]

$a_0 \leftarrow$ Relative bearing from TS to OS

if $abs(a_0) \leq phi_1$ **then** ▷ Determine TS region

    $TSR \leftarrow 1$

else if $a_0 \geq phi_1$ and $a_0 \leq phi_2$ **then**

    $TSR \leftarrow 2$

else if $a_0 \leq -phi_1$ and $a_0 \geq -phi_2$ **then**

    $TSR \leftarrow 3$

else

    $TSR \leftarrow 4$

end if

if abs $(b_{0\_180}) \leq phi_1$ **then** ▷ TS is ahead of OS

    $Flag$ assigned based on TSR, see Figure 7

else if $b_0 \geq phi_1$ and $b_0 \leq phi_2$ **then** ▷ TS is ahead on OS' starboard side

    $Flag$ assigned based on TSR

else if $b_{0\_180} \leq -phi_1$ and $b_{0\_180} \geq -phi_2$ **then** ▷ TS is ahead on OS' port side

    $Flag$ assigned based on TSR

else ▷ TS is behind OS

    $Flag$ assigned based on TSR

end if

return $Flag$

```
f = Function('f', {x, u, x_{ref}, u_{ref}}, {xdot, L});
```

The last two steps are to initialize the rest of the system and to fill in our initial conditions. The remaining pieces of the system are the vectors w, $w0$, lbw, ubw, g, lbg, ubg which can all be initialized as empty arrays. And an integral variable J which starts off as 0. We can now set our final initial conditions, $Xk = MX.sym('X0', 2)$ will be the state variable for the main loop. Xk is then inserted as the first element of w which will end up becoming the vector containing all states and control variables by the end of the main loop. Whenever w is updated we must give the newly inserted state or control variable an lower and upper bounds, this is done by adding an equal amount of elements to lbw and ubw, for lower and upper bounds respectively, as w so that the length of w, lbw and ubw are always the same. Additionally $w0$ needs to have the same length as w as it is the vector containing the initial guess at what the optimal solution will be, we can just add a zeros for initial conditions. Finally, we add our very first constraints into the vector g:

```
g = [g, {initial_pos-Xk}];
lbg = [lbg; 0; 0];
ubg = [ubg; 0; 0];
```

Algorithm 3 RK4

Begin
$M = 4$ ▷ RK4 steps per interval
$DT \leftarrow$ dt for algorithm divided by M
$X0 =$ MX.sym('X0' ,2)
$U =$ MX.sym('U0',2)
$Xd =$ MX.sym('Xd',2)
$Ud =$ MX.sym('Ud',2),
$X = X0$
$Q = 0$
for J = 1 to M **do**
    $[k1, k1_q] = f(X, U, Xd, Ud)$
    $[k2, k2_q] = f(X + k1 * DT/2, U, Xd, Ud)$
    $[k3, k3_q] = f(X + k2 * DT/2, U, Xd, Ud)$
    $[k4, k4_q] = f(X + k3 * DT, U, Xd, Ud)$
    $X \leftarrow X + DT/6 * (k1 + 2 * k2 + 2 * k3 + k4)$
    $Q \leftarrow Q + DT/6 * (k1_q + 2 * k2_q + 2 * k3_q + k4_q)$
end for
$F =$ Function('F', {X0, U, Xd, Ud}, {X, Q}, {'X0', 'p', 'Xd', 'Ud'}, {'xf', 'qf'})
End

With Initial_pos simply being the current north and east position of our ship. With that we're done with initial conditions and can move on to the main loop.

3.3 Main Loop

The main loop will iterate through our time horizon and for each iteration there are four main operations to conduct:

- New NLP variable for control
- Integrate till end of interval
- New NLP variable for state
- additional constraints

The first three are relatively easy and painless to implement, the fourth is easy to implement but rather time consuming depending on how detailed you want your dynamic constraints to be. Let's start from the top. Our NLP variable for control is Uk and it's initialized the same way we've initialized NLP variables already, as with the initial conditions we need to append the new variable to the end of w and similarly lbw and ubw need the upper and lower bounds allowed for Uk. Then we update our initial guess vector $w0$ with more zeros equal to the amount of control variables we have. the code ends up like this:

```
Uk = MX.sym(['U_' num2str(k)], 2);
w = {w{:}, Uk};
lbw = [lbw; -10; -10];
ubw = [ubw;  10; 10];
w0 = [w0;  0; 0];
```

Integrating until the end of the interval is the part that ensures we go somewhere during our control interval. We grab our desired position and velocity references mentioned when we set up the initial conditions and then we run the function for discrete time dynamics made by the RK4 method. The function outputs two variables, xf and qf, we save xf for later and integrate up qf by adding it to J. All of this is done with 5 lines of code:

```
x_ref_i = reference_trajectory_los(1:2,k+1);
u_ref_i = reference_trajectory_los(3:4,k+2);
Fk = F('x0', Xk, 'p', Uk, 'Xd', x_ref_i, 'Ud', u_ref_i);
Xk_end = Fk.xf;
J=J+Fk.qf;
```

Then we make a new NLP variable for state, this is simply the Xk for the next iteration and it's done the same way as the last time we made an Xk.

3.3.1 Constraints

The constraint vector, g, contains both our equality and inequality constraints, or rather g contains all the constraint functions, while lbg and ubg contains the limit. If lbg and ubg have the same value it's an equality constraint, otherwise it's an inequality. The first constraint we want is the equality constraint that ensures that the end of the current interval is equal to the beginning of the next interval

```
Xk = MX.sym(['X_' num2str(k+1)], 2);
w = [w, {Xk}];
lbw = [lbw; -inf; -inf];
ubw = [ubw;  inf;  inf];
w0 = [w0; 0; 0];
```

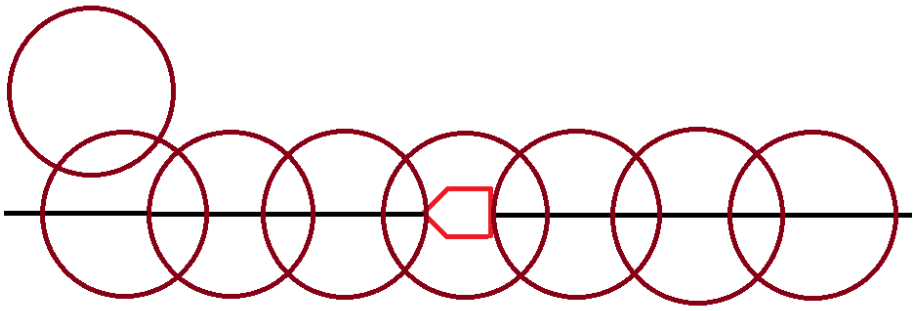

Figure 8: A TS traveling due west, the center of each circle is a point in the constraints vector g, while the circumference of the circle shows the bounds

```
g = [g, {Xk_end-Xk}];
lbg = [lbg; 0; 0];
ubg = [ubg; 0; 0];
```

The next constraints we want to add are the points in our interpolated static_obs matrix, these are already well prepared and we add them much the same way we added the equality constraint for Xk_end. Iterating through all the columns in the interpolated matrix and adding the distance between Xk and the point for the current column to g. Then use the lower bounds lbg to set how close to the point we are allowed to be. These are inequality constraints because we don't care how far away we are as long as it's above the desired minimum distance. The process for adding dynamic obstacles is a bit more involved because we need to consider COLREGs. We already assigned COLREGs flags to each TS, so that part is solved with a simple if-else structure, deciding where to put the constraints is a bit more up in the air. Consider a Head-on (HO) situation, we should always prefer to turn to our starboard as the COLREGs rule say, so far as this is possible. We also want to indicate reasonably early that we do intend to yield to our starboard side. Lastly we don't want to immediately go back to our original path because that might take us through some unnecessary wake. The constraints I designed for a HO situation can be seen in Figure 8. How far ahead of the TS we should place points should ideally be based on both the speeds of TS and OS, as well as the length of both ships. Bigger ships and faster ships should have constraints going further ahead of it to account for lack of mobility or time to react. Every COLREGs situation should have its own pattern for distributing these constraints origin points.

3.4 Updating States

The last section of the algorithm is where we use CasADi's ipopt functionality to solve the NLP problem we've made during the main loop. First we instance a solver:

```
prob = struct('f', J, 'x', vertcat(w{:}), 'g', vertcat(g{:}));
solver = nlpsol('solver', 'ipopt', prob);
```

It's not very intuitive to see what happens here, basically we create the NLP problem with function J, our states and control variables w and constraints g. Then we instance a solver and tell CasADi we want it to use ipopt to solve the problem. Before actually solving the problem we'll want to implement a quick trick to speed up consecutive reruns of the algorithm. Solving the optimization problem without any initial guess can be very time consuming, so on later reruns we'll want to use the previous optimal solution as an initial guess. In addition to speeding up consecutive runs it also ensures that we're likely to converge on a similar local minimum, this reduces the amount of jumping between local minimums NLP solvers might encounter. This trick is easily implemented by saving the w_opt output from the solver, into a persistent variable, and if the persistent variable is not empty we set $w0 = previous\_w\_opt$. Now let's call the solver and find our w_opt:

```
sol = solver('x0', w0, 'lbx', lbw, 'ubx', ubw,...
        'lbg', lbg, 'ubg', ubg);
w_opt = full(sol.x);
```

Here, w_opt is a vector that contains the optimal states and control variables, but since we're doing MPC here we're actually only interested in the first new state and the control variable that will get us there. With our generous assumption that our ship is capable of precisely following the optimal solution we can directly update the OS position η_{OS} and velocity $\dot{\eta}_{OS}$ with the optimal position and speed from w_opt. And with that we're done.

4 Simulation Results

4.1 Simulator

The simulations are run in a simulator implemented in MATLAB, which has support for running an arbitrary number of vessels. A simulation is initialized from a file, where each vessel is defined by its initial condition, vessel model and set of waypoints. In addition a controller is assigned for each vessel. For the simulations presented here, the TSs run a simple LOS guidance controller, while the OS runs the propsed method from Chapter 3.

4.2 Simulations Overview

The method that we proposed in this work will be demonstrated through a set of six simulated scenarios. Each scenario will test a different aspect of the algorithm, some will be general trajectory planning in absence of TSs, and some will be about testing the COLREGs compliance. Before diving into the results let's get a brief overview of each scenario. The first three scenarios are inspired by some of the scenarios used by Loe 2007

Walk in the park: This is the most basic scenario, it's simply here to show that the algorithm does indeed work. There are three islands in the environment and we're simply trying to get from one end of the area to the other. This scenario actually highlights one of the bugs I encountered with NLP so it's still a somewhat interesting result.

Local minimum: A common pitfall of NLP solutions to trajectory planning problems is that they get trapped in local minimums created by the environments. This can be a simple shoreline formation that creates a bowl to trap the ship with. This is where the traffic pattern assist comes in handy for our ship, essentially turning the algorithm from a simple MPC to a hybrid algorithm capable of following a reference path instead of simply trending towards a goal.

Blocked Path: There will be moments at sea where our path will be blocked by larger ships and suddenly we find ourselves without any valid solution to the NLP problem. It's important to see how our algorithm reacts to such a scenario.

Canal Crossing: A simple task of getting from one end of a canal to the other, made slightly problematic by a non-COLREGs-compliant TS directly next to our OS at the beginning. A different crossing vessel is also included that we must yield for.

Canal Crossing, Head-on: A twist on the previous scenario. This time the non-COLREGs-compliant TS does cooperate and yield as it should, but a different obstacle has been introduced, a different TS crossing the canal in the opposite direction.

Canal Crossing, overtaking: Also a twist on the original canal crossing, but instead of meeting a TS crossing in the opposite direction we encounter a slow moving TS crossing the same way as our OS.

4.3 Results

4.3.1 Walk in the Park

The results from this scenario are shown in Figure 9. In the first frame we see that the optimal solution wants to take us on a wild detour. In the next frame we see that as we get close to the goal the optimal solution "sticks" to the reference trajectory more the closer we get. In the last frame we see that we successfully arrived at the goal without having to take the detour. While the ship manages to get to the goal just fine as it should, the optimal solution found by the NLP solver does raise some questions. It clearly wants us to take a long detour around the northernmost island, and there is a small bump about halfway through the transit to avoid a non-existent obstacle. While our ship ends up not doing the detour we do follow the bump, which is not ideal if there were other vessels in the area. One of the cornerstones of COLREGs is that every vessel needs to clearly communicate intent, and this half-circle maneuver out of nowhere muddles our intent.

4.3.2 Local Minimum

See Figure 10. In the first frame, we see that we do not start at the beginning of the reference path, but we're still able to generate a smooth trajectory to the goal. In frame two and three we see that the ship easily navigates to the goal. In this scenario we successfully conquer one of the most common problems of MPC trajectory planning, getting stuck in terrain. Beyond that this result is nothing spectacular. By utilizing traffic patterns to generate a reference path we're essentially turning the algorithm into a hybrid method of MPC and LOS guidance.

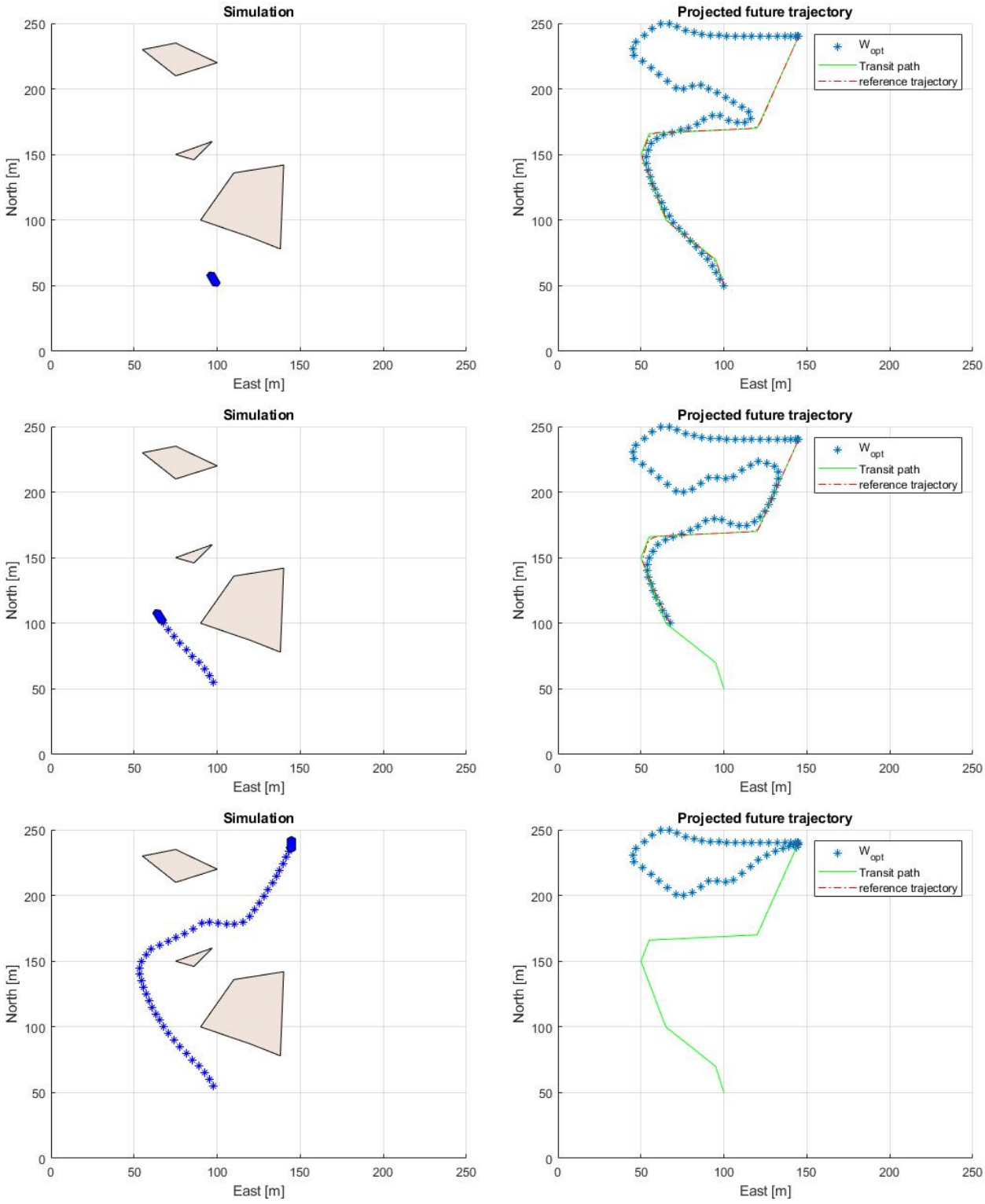

Figure 9: Result from Walk In The Park, chronologically from top to bottom.

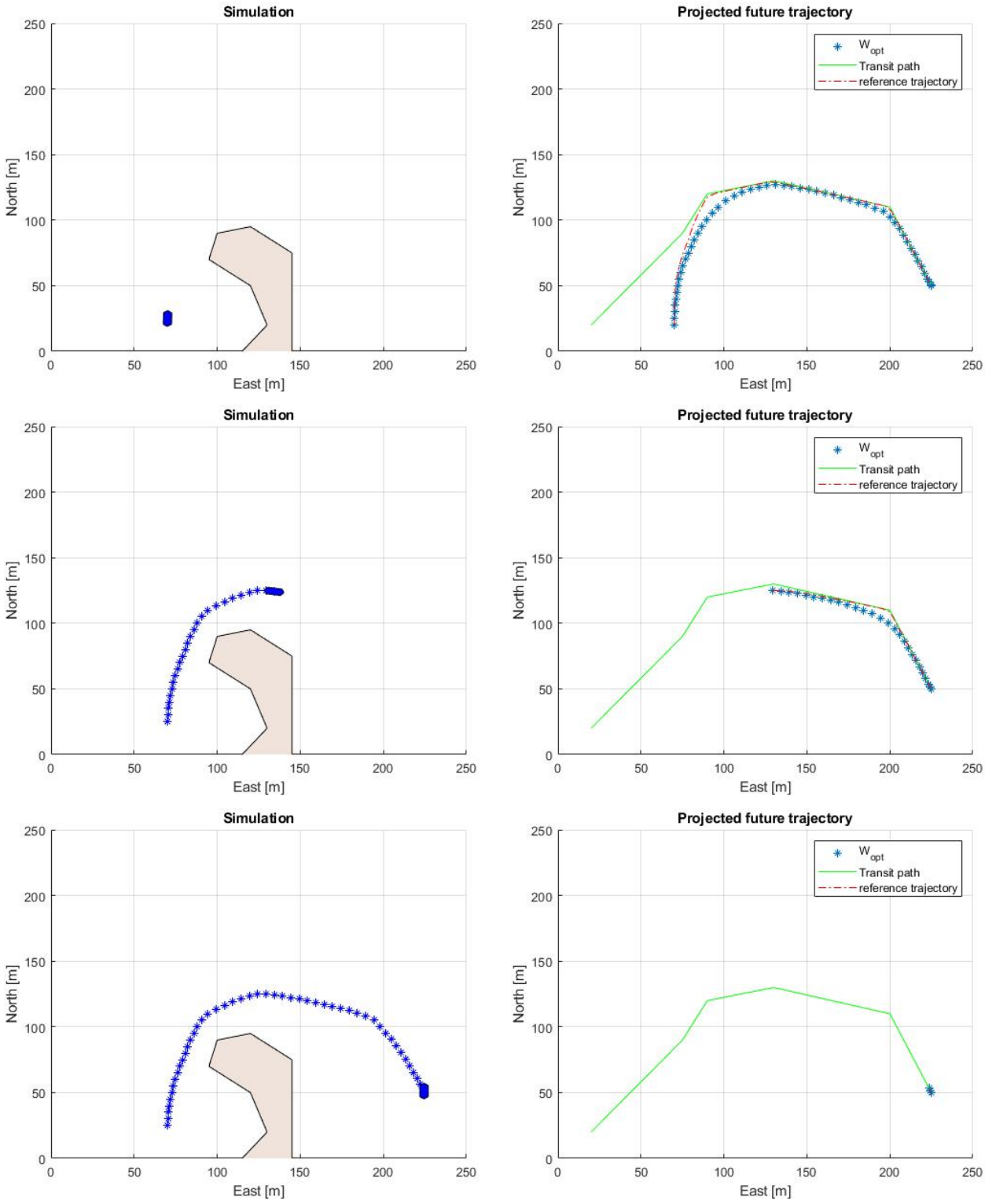

Figure 10: Result from Local Minimum, Chronologically from top to bottom.

4.3.3 Blocked Path

The results are shown and described in Figure 11. In the first frame the path is clear, and a mostly straight trajectory is generated, the small bump is due to the reference path being slightly too close to the edge of the opening. In the next frame we see that an obstacle has suddenly blocked the path, in a realistic situation this could be a large ship slowly crossing by as we approach the pass. When the pass is blocked the solution converges on an infeasible point, and our ship slows down significantly. In the last frame we see that as soon as the passage is clear again our ship resumes it's trajectory towards the goal. The behaviour here is pretty good; when there is no solution to the trajectory planning problem we would ideally slow down and eventually stop completely in order to wait for a feasible solution, and that is exactly what happens.

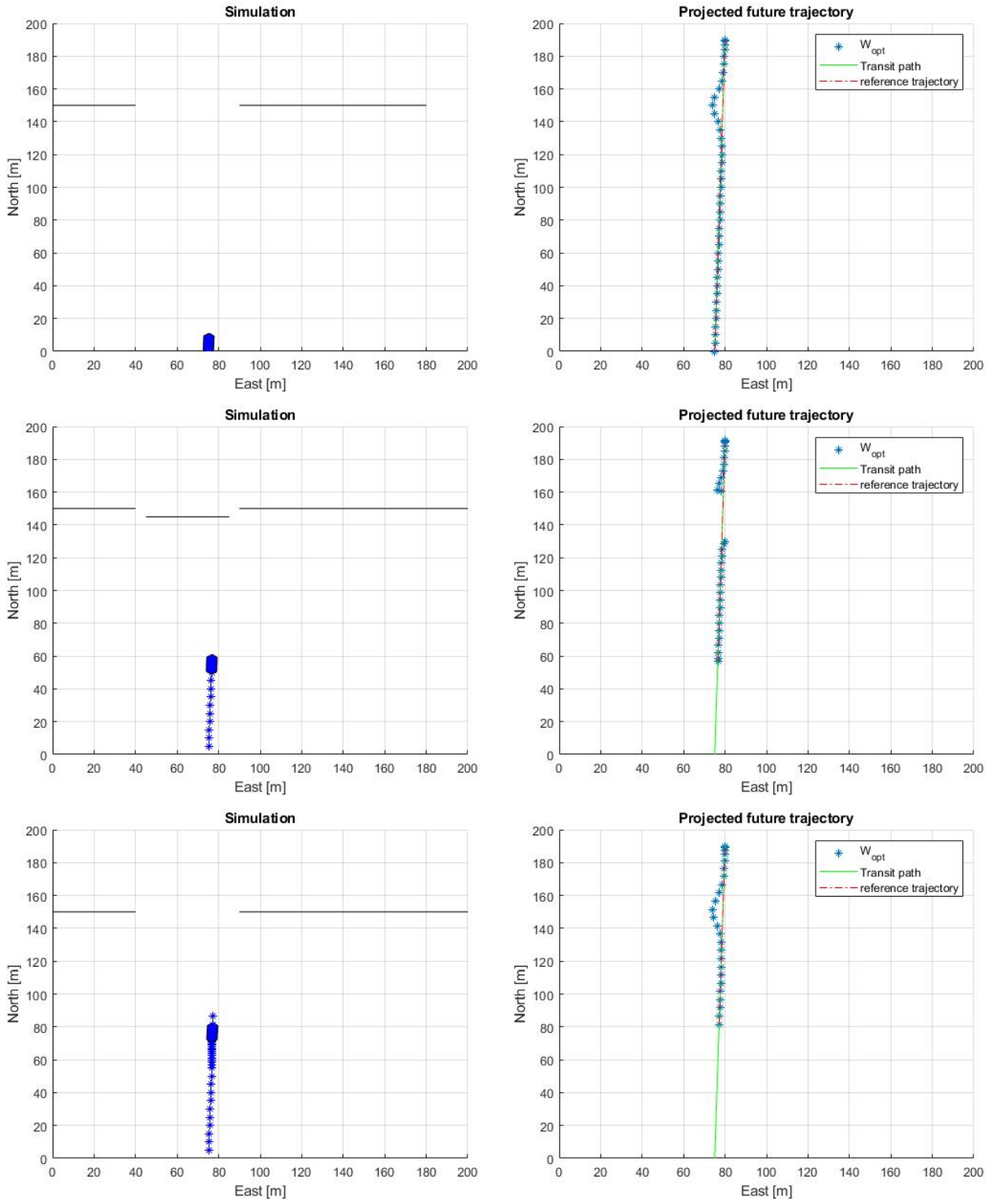

Figure 11: Result from Blocked path, chronologically from top to bottom.

4.3.4 Canal Crossing

Results in Figure 12. In the first frame we see that we're expecting to give way to the TS approaching from the east. In frame two we see that we're forced to yield to the TS approaching from the west, despite our OS having the right of way. In the third frame we've passed behind the TS forcing its way through and we're continuing towards our goal. In the last frame we see that we do arrive at our goal, having yielded to both TSs. We see that our OS is capable of performing emergency maneuvers even when encountering a non-COLREGs-compliant TS. We also see that our OS is capable of giving way to TS that have the right of way in a crossing situation, however the decision to yield should ideally come earlier.

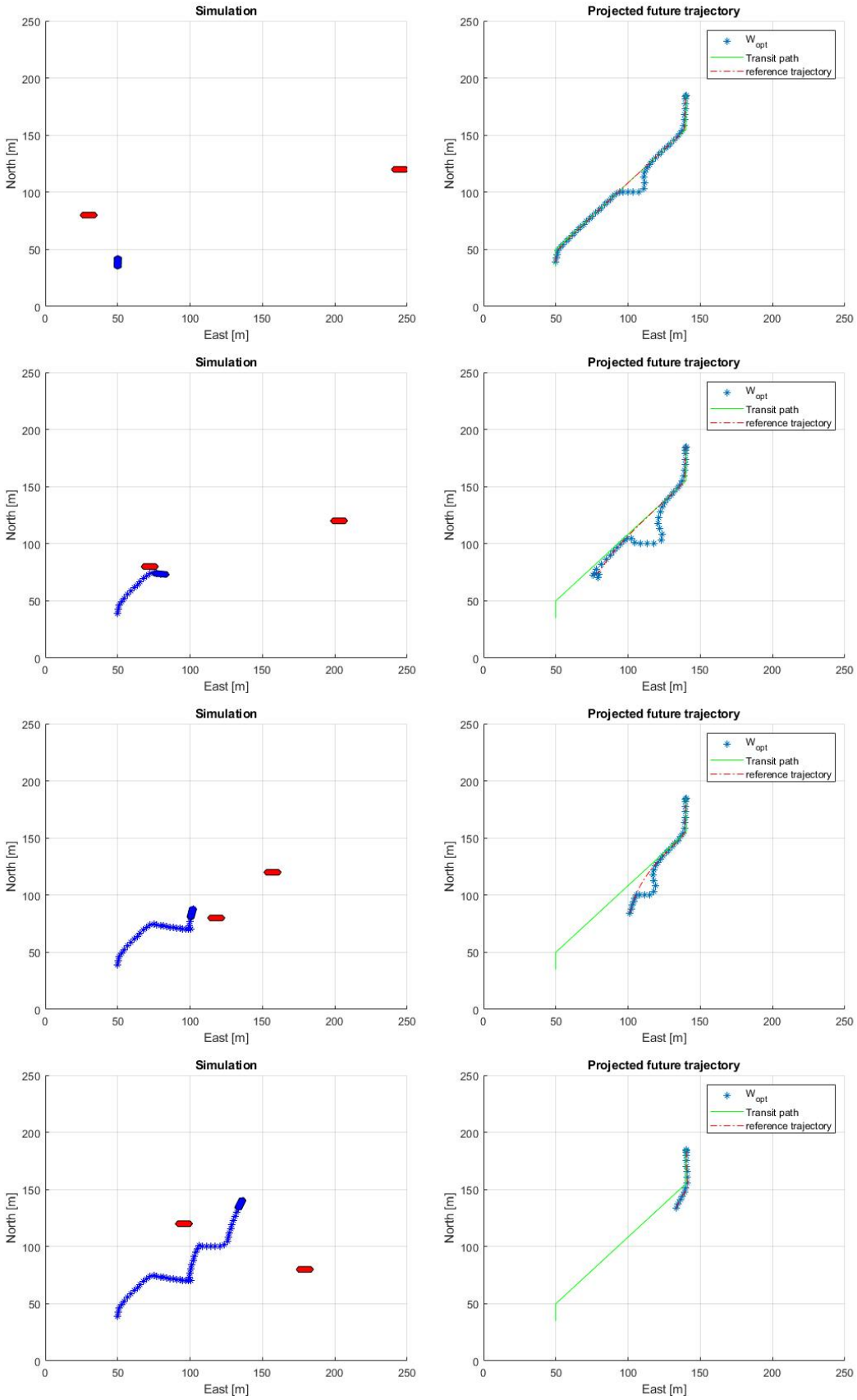

Figure 12: Result from canal crossing, chronologically from top to bottom.

4.3.5 Canal Crossing, Head-on

The results in Figure 13 show COLREGs-compliant behaviour displayed again, but the decision to yield for give-way is still made too late in my opinion. In the first frame we can see that we're already anticipating to yield to our starboard for the oncoming TS. This is because we know it's coming to cross in the opposite direction. In frame two and three we see the maneuver executed and in the last frame we see that we successfully yielded to both TSs.

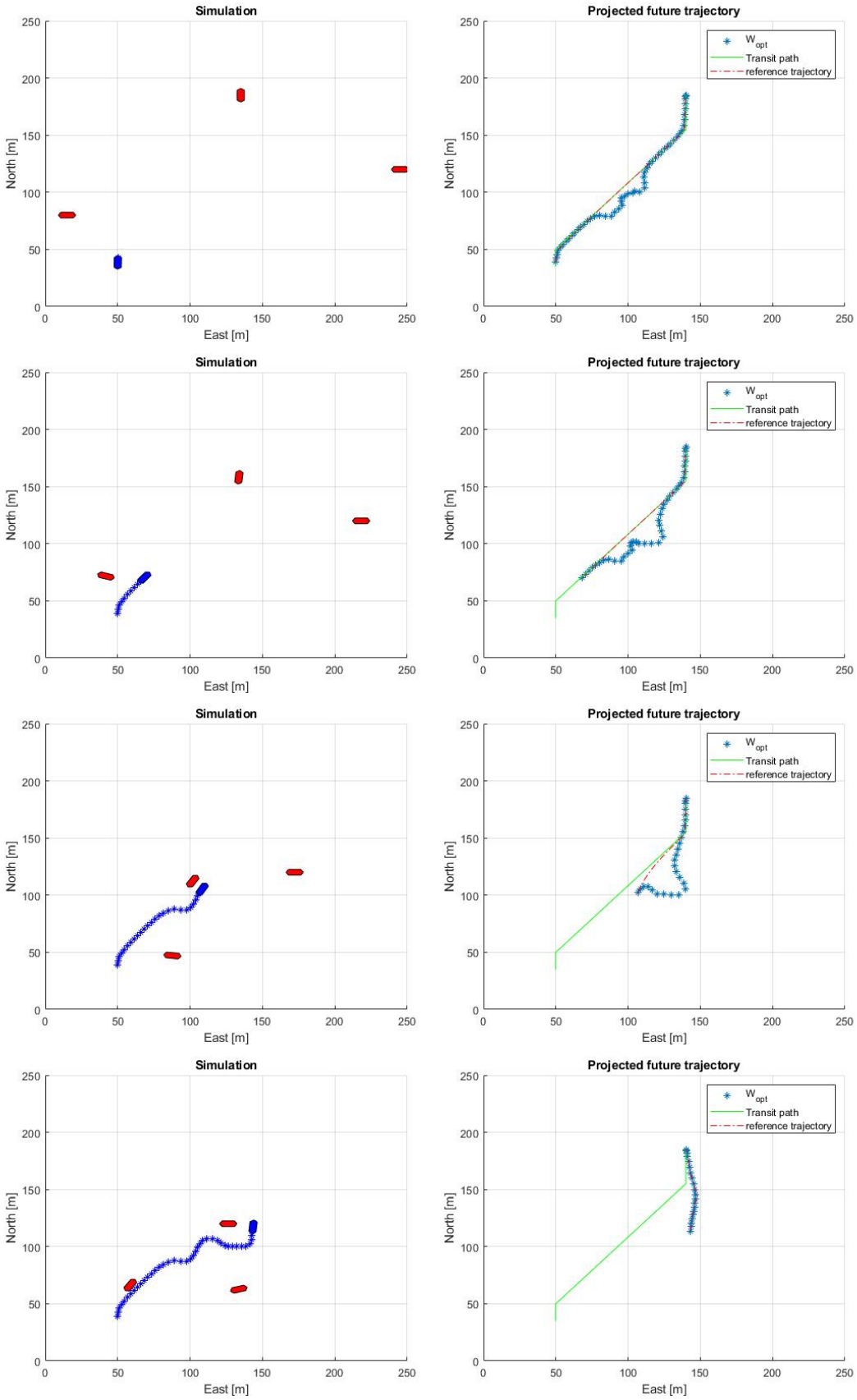

Figure 13: Result from Head-on canal crossing, chronologically from top to bottom.

4.3.6 Canal Crossing, Overtaking

The results from the final scenario are in Figure 14. Here we see that we maintain good distance from the ship we're overtaking the whole time. For this scenario I chose to move the TS crossing from the east because it made the result more visually cluttered without affecting our ability to properly overtake.

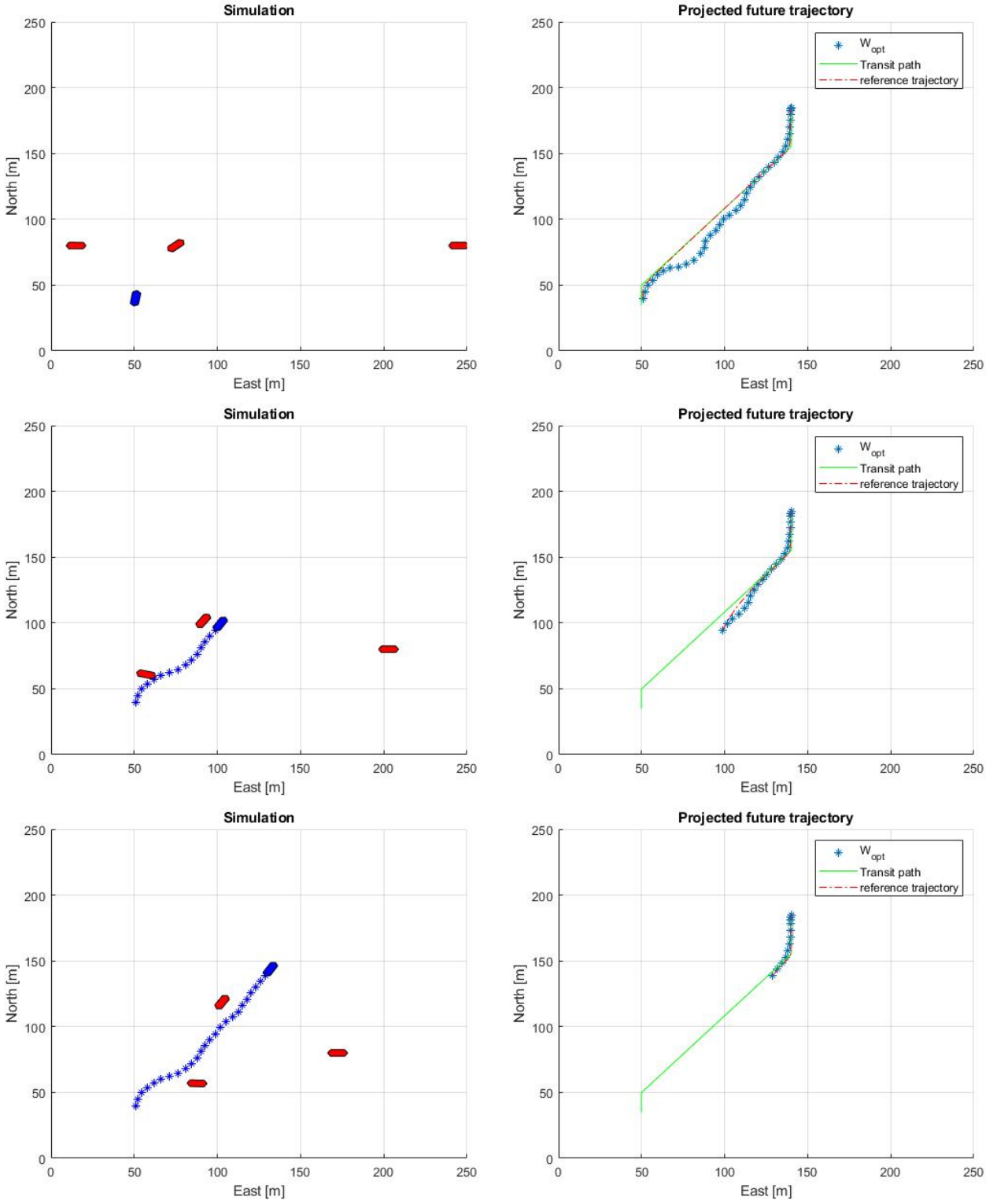

Figure 14: Result from Overtaking canal crossing, chronologically from top to bottom.

4.4 Discussion

Let's begin the discussion with the "easy" scenarios where there are no TSs. In these scenarios we observe that the algorithm is capable of planning a trajectory that follows a reference path, but it does have some issues. In "Walk in the park" especially we observe a strong deviation between the optimal solution and the reference path. I'm fairly certain that the reason for this behaviour is the excessive amounts of active constraints. Many sections of these islands do not need to be active as constraints, but I don't have code sophisticated enough to dynamically discard constraints that are technically out of our line of sight. One solution here is to only consider static obstacles that are within a certain distance from our reference path. In the Local Minimum and blocked path scenarios there are less active constraints and so the results from these do not exhibit the same deviation from the reference. Beyond the quirk of generating a sub-optimal path when there are too many active constraints another potential issue with the algorithm is the lack of dynamic finite horizon for the NLP problem. This issue doesn't show itself too much in these scenarios, but it's a problem I encountered a lot during development. Basically what can happen is that as the OS approaches the final goal more and more of the future states will end up at rest in the goal, this in turn makes it less and less important how we get to our goal. Once we're there we're going to spend so many iterations at rest that the lead-up doesn't impact the objective much anymore. By having a dynamic finite horizon that gets shorter as we approach the goal we would maintain the importance of the reference path all the way until we reach the goal.

Over on COLREGs: the situation assessment algorithm performs satisfactory in the scenarios I ran, but it's not perfect. One problem I ran into while testing is that if the assessment algorithm runs on the very first iteration of the simulation there are sometimes bugs with the direction agents are facing, they don't match up with the direction they're supposed to have, and that leads to wrong classification. The solution I came up with to combat this was to simply not run COLREGs assessment on the very first iteration. One iteration is a very short amount of time and this delay had no noticeable effect on the COLREGs compliance of the algorithm. Another potential issue with the algorithm is lack of care for the distance between agents, which is not an issue when I design scenarios that take place in a 200x200 meter square, but if you wanted to use this algorithm to get far away you would naturally start encountering vessels far far away in the distance and assign a COLREGs classification way before any proper situation arises. A simple distance check should suffice to prevent this from happening, a more advanced algorithm

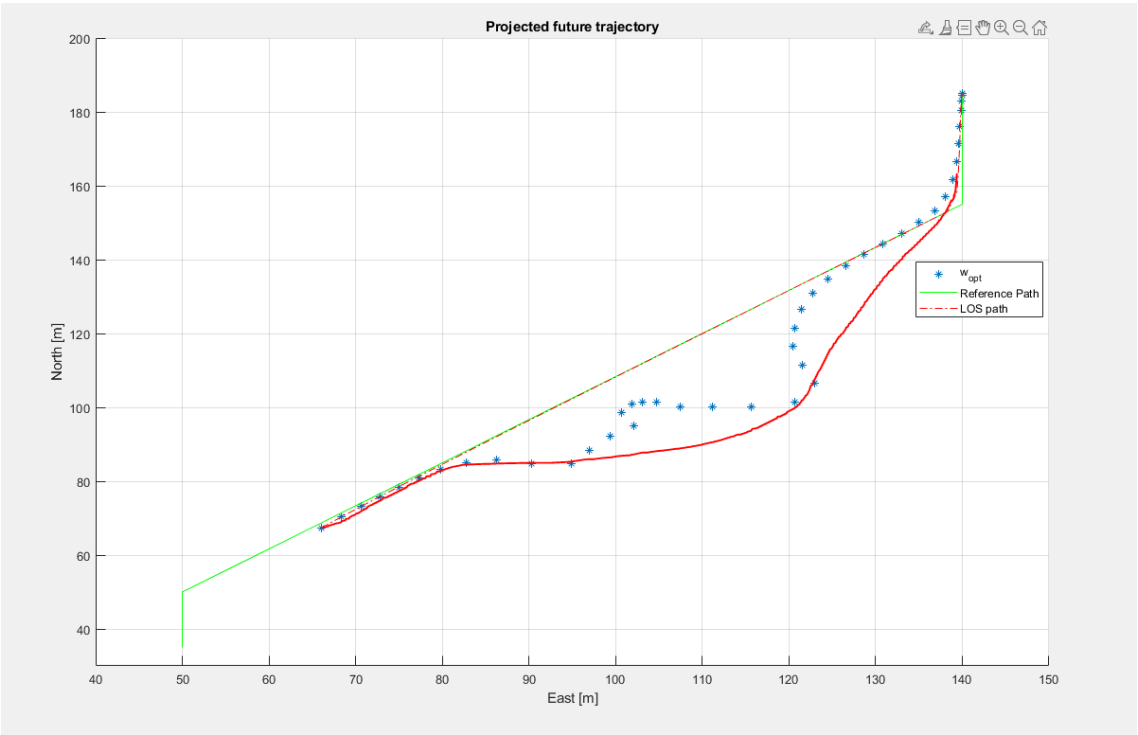

Figure 15: Idea for how the smoothed out optimal trajectory might look with a 2-pass algorithm.

would be some sort of full risk assessment algorithm that looks at both distance between agents, and also distance of closest point of approach as well as time until closest point of approach. Only if these closest point approach values exceed some threshold do we run the COLREGs situation assessment algorithm.

As for COLREGs compliance the algorithm leaves some to be desired, for example look at figures 12 and 13. The behaviour does technically yield properly, but the movement ends up being quite zig-zag like. This is mostly a problem of how the dynamic obstacle constraints are placed, and is something that can be improved on with more development. Another idea I had for solving this problem that I did not have any time to explore was to make the algorithm a two-pass program, where the first pass is as the algorithm is now, and the second pass adjusts the proposed optimal solution to become a smoother curve. For example, in Figure 13 notice how turbulent the optimal solution appears to be. If we fixed in place the outermost extremal point, the turn where we turn from a heading due east, to one due north northwest. Make a smooth curve that takes us from where we are, through the extremal point, and then towards the reference again. We would not only end up with a smoother, more comfortable trajectory, it would also be more COLREGs-compliant because it signals our intent to give-way a lot earlier than the pure optimal path. See Figure 15 for a visual of what I mean.

Another minor thing worth remembering is that COLREGs are generalized rules, there might be special case rules that will supersede COLREGs in certain areas, this algorithm obviously can't account for those special case rules.

Lastly, I want to discuss traffic pattern assistance. Specifically the validity of the assumption that we can use the traffic pattern to generate a reference path for our OS, as well as predict the trajectory of TSs. After seeing the picture of AIS data saved over two days, see Figure 2, I think it's safe to conclude that historical traffic patterns can be used to generate a reference path for our own ship to follow, which sort of becomes a crowd-sourcing method of where it's safe to traverse. I believe that this sort of data would be a useful tool to any trajectory planning algorithm regardless of method. The assumption that we can accurately predict other agents out and about however is dubious at best. An article by (Schöller et al. 2021) does show that it's theoretically possible to get a better guess than simply assuming fixed course and speed. However the accuracy of this prediction introduces uncertainty that could affect the behaviour of the OS in unpredictable ways.

One thing I would have loved to do is to disable the prediction for TSs in the simulation and instead use the fixed speed and course assumption. Sadly when I tried doing that something broke within my implementation, and I can not figure out what goes wrong. The results when using fixed speed and course prediction are so bad that it would not be a fair comparison. Figure 16 shows the results from one of these attempts. The simulation is the same as the canal crossing with Head-On shown in Figure 13. In the second frame of the simple prediction scenario there is suddenly no feasible solution to the NLP problem. This causes our OS to slow down and get caught by the TS approaching from the west. The rest of the simulation is just our OS struggling to recover, the optimal solution continues to drift away from the reference and the whole scenario turns into a disaster. I'm actually not sure why this behaviour ends up being so bad, and this is definitely one of the things I would like to work more on in the future.

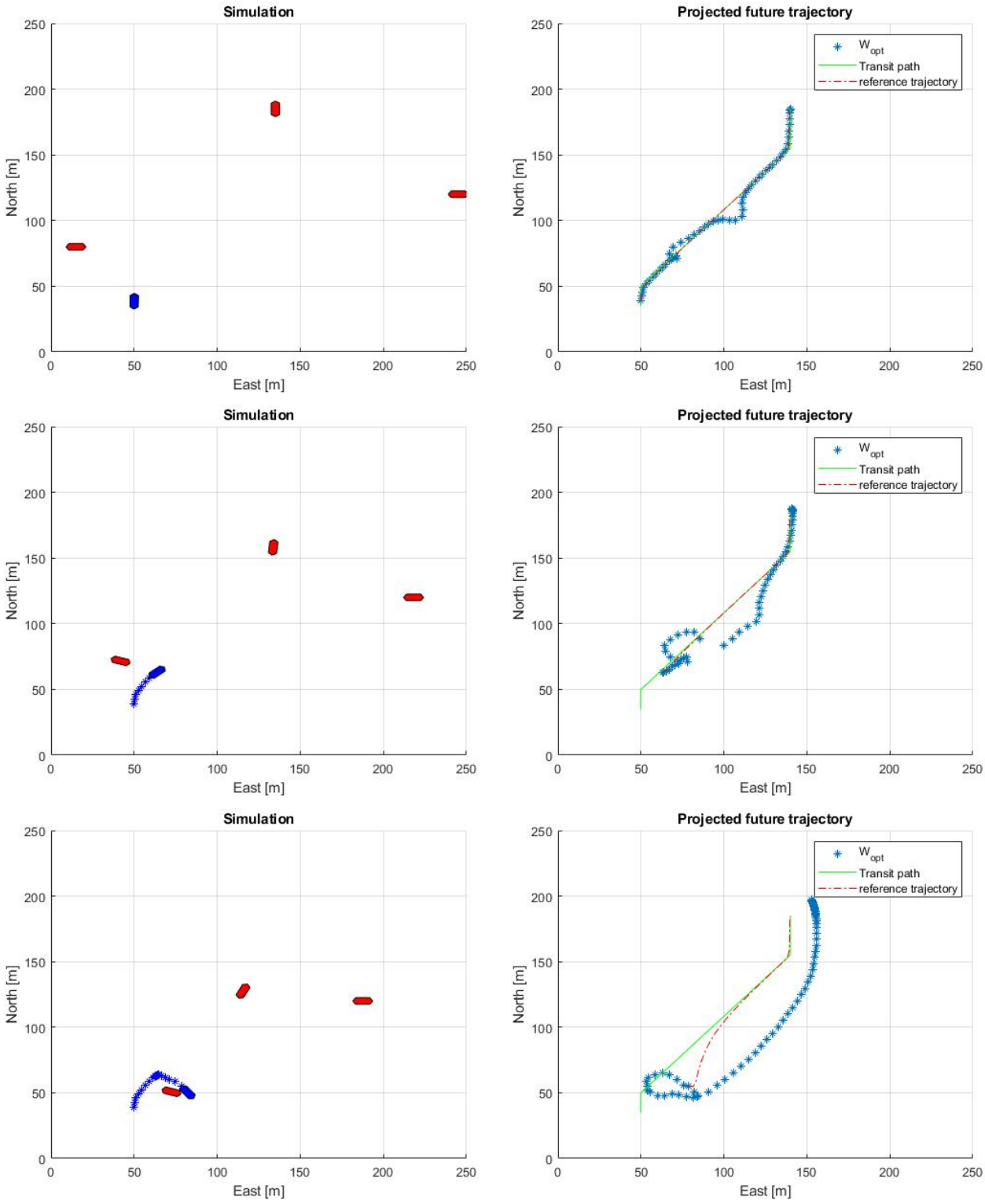

Figure 16: Canal crossing with Head-on situation using the assumption that TSs will maintain fixed course and speed.

5 Conclusion and Future Work

In conclusion, I have developed a hybrid Model Predictive Control (MPC) & Line of Sight (LOS) guidance trajectory planning algorithm that is able to follow a reference path. The algorithm is also COLREGs-aware, being able to assess COLREGs situations and behave accordingly to a somewhat satisfactory manner. The COLREGs assessment algorithm is rather simple and works based on the relative bearing from OS to TS as well as the relative bearing from Target Ship (TS) to ownship (OS). COLREGs assessment does not take distance or relative speeds into account, something that could be implemented for an even more robust situation assessment. The algorithm is able to avoid both static and dynamic obstacles alike, with the underlying assumption that our OS has sufficient sensory equipment to detect and measure distances to these obstacles, as well as knowing the speed over ground for dynamic obstacles. Another assumption the algorithm makes is that we're able to use historical traffic data gathered via AIS or some other sensor equipment to create a reference path the OS. This historical traffic data is also assumed to let us accurately predict the future behaviour of all other TS in the area, which definitely needs a lot more research and development to become a realistic assumption.

The COLREGs compliance could use a bit more number tweaking and constraint placement considerations, but at the very least our OS is not in direct violation of any COLREGs rules. That said the behaviour is a bit "zig-zag" when yielding, which is partly because all the constraints are circular. The distance between circles and the radius of each circle can cause a sort of bouncy trajectory as the optimal solution is dragged along the wall formed by the constraints. I also think that when giving-way or yielding for other target ships our OS doesn't maneuver early enough. One of the less considered rules would be the rule about signaling and maintaining clear intent as to where the OS is going. By going straight forward to the boundary of the constraint, and then "hugging" them the whole way we don't show clear intent that we're going to yield for a crossing TS.

5.1 Future Work

When it comes to future work I think one of the first things that should be improved upon is signaling clearer intent of where we want to go. Clear intent is important for every agent in the area to be able to make the proper judgement call with confidence, while uncertainty leads to errors which leads to accidents. Another aspect to improve

would be the objective function, I use a rather simple one for this algorithm, but with more time and development a much more advanced objective can be formulated, taking into account many more aspects than just the reference path and velocity. On the same topic, one of the things needed in order to apply this algorithm in a full scale test would be to replace the dynamics with a more realisitcally modelled vessel dynamics. Lastly I think the traffic pattern idea needs more work, which is one aspect that is not really explored much in this study, it's simply assumed that it's a possibility to use the data the way I want. Actually developing the algorithms and software to extract, process and make use of the traffic pattern data is a whole project in and of itself.

References

Andersson, Joel AE, Joris Gillis, Greg Horn, James B Rawlings and Moritz Diehl (2019). 'CasADi: a software framework for nonlinear optimization and optimal control'. In: *Mathematical Programming Computation* 11.1, pp. 1–36.

Cockcroft, AN and JNF Lameijer (2012). *Maneuvers to avoid collision A Guide to the Collision Avoidance Rules (pp. 170-172).*

Eriksen, H. Bjørn-Olav and Morten Breivik (2017). 'MPC-based mid-level collision avoidance for ASVs using nonlinear programming'. In: *2017 IEEE Conference on Control Technology and Applications (CCTA)* (Mauna Lani Bay Hotel). IEEE. Hawaii, USA, pp. 766–772.

Fossen, Thor I (2011). *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons.

Huang, Yamin, Linying Chen, Pengfei Chen, Rudy R Negenborn and PHAJM Van Gelder (2020). 'Ship collision avoidance methods: State-of-the-art'. In: *Safety science* 121, pp. 451–473.

Loe, Øivind (2007). 'Collision avoidance concepts for marine surface craft'. In: *Specialization project report. Norwegian University of Science and Technology (NTNU), Trondheim, Norway.*

Schöller, Frederik ET, Thomas T Enevoldsen, Jonathan B Becktor and Peter N Hansen (2021). 'Trajectory prediction for marine vessels using historical AIS heatmaps and long short-term memory networks'. In: *Proceedings of 13th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles Elsevier*. Vol. 54. 16. IFAC. Oldenburg, Germany: Elsevier, pp. 83–89.

Tam, CheeKuang and Richard Bucknall (2010). 'Collision risk assessment for ships'. In: *Journal of Marine Science and Technology* 15.3, pp. 257–270.

Vagale, Anete, Rachid Oucheikh, Robin T Bye, Ottar L Osen and Thor I Fossen (2021). 'Path planning and collision avoidance for autonomous surface vehicles I: A review'. In: *Journal of Marine Science and Technology* 26, pp. 1292–1306.

Woerner, Kyle (2016). 'Multi-contact protocol-constrained collision avoidance for autonomous marine vehicles'. PhD thesis. Massachusetts Institute of Technology, USA.