TRANSNAV
http://www.transnav.eu
the International Journal
on Marine Navigation
and Safety of Sea Transportation
Volume 15
Number 3
September 2021
DOI: 10.12716/1001.15.03.01

Differentiable Programming for the Autonomous Movement Planning of a Small Vessel

C. Bahls[1] & A. Schubert[2]
[1] *German Aerospace Center, Neustrelitz, Germany*
[2] *University of Rostock, Rostock, Germany*

ABSTRACT: In this work we explore the use of differentiable programming to allow autonomous movement planning of a small vessel. We aim for an end to end architecture where the machine learning algorithm directly controls engine power and rudder movements of a simulated vessel to reach a defined goal. Differentiable programming is a novel machine learning paradigm, that allows to define a systems parameterized response to control commands in imperative computer code and to use automatic differentiation and analysis of the information flow from the controlling inputs and parameters to the resulting trajectory to compute derivatives to be used as search directions in an iterative algorithm to optimize a goal function. Initially the method does not know about any manoeuvring or the vessels response to control commands. The method autonomously learns the vessels behaviour from several simulation runs. Finally, we will show how the simulated vessel is able to fulfil some small missions, like crossing a flowing river while avoiding crossing traffic.

1 INTRODUCTION

1.1 *Autonomy and Automation*

In the scope of DLR's Transport Program - the project I4Port works on reaping the fruits of digitalization by making intermodal shipping more efficient, robust and transparent by focussing on transport processes in the port as a hub of a transportation network.

The aim of the project is to improve the efficiency and robustness of inter- and trans-modal logistic chains by making port processes more intelligent and informative as well as increase their integrity. The goal is to research technological and process-oriented approaches for optimised, efficient and secure traffic and logistic processes in a port as an intermodal hub. I4Port is a bridge into Helmholtz' Program Oriented Funding Period IV (PoF IV) where the Transport Program will put an increased emphasis on Nodes as Intermodal Hubs.

Part of the project I4Port are conceptual and technological developments for maritime and inland waterway traffic related to ports, for example: PPP and RTK for position, navigation and timing [2, 8, 15]. Additionally, the project concerns itself with the development of planning algorithms for autonomous vessels in busy waterways - the work presented int this paper.

In this paper we present a concept for the movement planning of an autonomous ferry - in an inland waterway. Ferries on inland waterways are an especially welcoming area of application for highly automatic or even autonomous systems for operating vessels. Usually a ferry will travel close to a predefined trajectory in a very limited and well know area. Also, the ships operations can easily be supported and augmented by land-based or other permanent infrastructure, especially sensors for traffic situation assessment and environmental influences like wind speed and direction as well as waterflow and -level.

In this paper we investigate the movements of a simulated vessel to evaluate a concept for autonomous movement planning using a state-of-the-art artificial intelligence algorithm. As a blueprint we will use the ferry crossing the river Warnow from Warnemünde to Hohe Düne in Rostock Port.

1.2 *Autonomy vs Automation*

As there are different uses of the words "autonomy" and "automation" - throughout this paper we will be using following definitions:

We will use the words "automation" or "automatic" when a system has little or no human involvement in operation and solves well-defined tasks that have predetermined conditional responses, for example a system that shows rule-based behaviour in a well-known and/or well-structured environment.

We will use the words "autonomy" or "autonomous" when a system has certain capabilities that allow it - within a defined and bounded application domain - to flexibly respond to unplanned situations thereby showing a certain degree of self-governance and self-directed behaviour. As an additional criterium the system's response should be adaptive to and/or learned from the environment.

1.3 *Manoeuvre Automation Levels*

The classifications of autonomy levels for surface vehicles are based considerably on the concept defined by the Society of Automotive Engineers (SAE) in 2014 [11]. With six levels, this classification ranges from purely manual to fully autonomous control. Only at the highest level does the control actually have a fully autonomous characteristic, so that the vehicle can independently reach a mission objective. The levels in between are characterised by a higher degree of automation.

The most classifications for the marine world adapted the SAE levels, without considering the significant differences between car and ship control. This starts with the dimensions of the vehicles, followed by the costs for a new building or even just the modification towards more automation. Finally, the personnel expense is higher as well as the knowledge and skills of a master are manifold more extensive than those of a car driver.

Often, the marine classifications include the remote control from shore [7, 10]. This presupposes that extensive and high-frequency communication between ship and shore can be realised, whereby the ship must also be equipped with far more sensors than is the case today with conventional ships, which are primarily controlled on sight. The amount of sensor data from additional cameras, lidar and radar sensors would significantly increase the data volume. Additionally, remote control requires the complete digitisation of the engine and propulsion systems in order to monitor and command their states from shore.

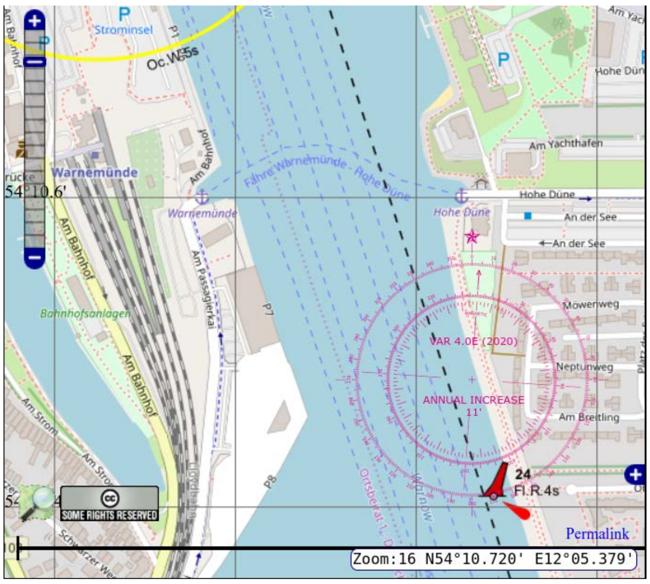

Figure 1. Ferry crossing in Warnemünde, using a map from openseamap.org

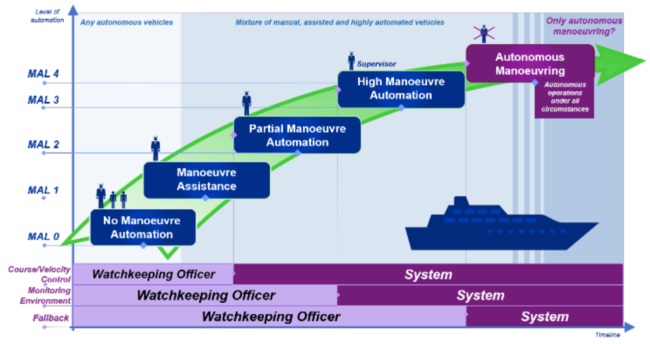

Figure 2. Manoeuvre Automation Levels, adapted from [13]

The concept of Manoeuvre Automation Levels (MAL) is a user-centred approach for gradual automation of conventional ships which are already in service today [13]. It is initialised by assistance of the manual control and the stepwise addition of automatic functionalities via a Manoeuvre Assistance System (MAS).

The approach is designed to both increase safety and efficiency already in manually controlled vessel manoeuvring in higher structured environments such as ports and coastal areas, and to transparently communicate the new automatic functions to the watch officer so that she can continue to exercise her responsibility for the vessel.

In the first level Manoeuvre Assistance (MAL1), the MAS is introduced to bundle the necessary nautical information and show the motion prediction in the electronic navigational chart (ENC). This level also serves to familiarise the nautical personnel with the MAS functions.

In the second level Partial Manoeuvre Automation (MAL2), short manoeuvre sequences are introduced, which can be initialised and supervised via MAS, e.g., automatic berthing or automatic collision avoidance manoeuvres in open waters. A prerequisite for automatic manoeuvres is a digital manoeuvre plan, which is displayed in the ENC. The plan is converted into a trajectory that forms the target for the automatic manoeuvring. By comparing of manoeuvre plan, actual and predicted motion track, the officer of the watch can evaluate if the trajectory control is working correctly. In the worst case, it should take over manual control again.

In MAL3, High Manoeuvre Automation, the automatic manoeuvre sequences are expanded. The procedure may include an entire crossing from port to port, which is based on a digital plan for the current weather situation or other framework conditions.

The highest level MAL4 presents the Autonomous Manoeuvring. There is no crew on board the ship, neither on the bridge nor in the engine control room. The vessel is autonomously controlled under all circumstances. This absolute formulation shows how unlikely this scenario is for longer missions.

In the research project GALILEOnautic, the presented concept of manoeuvre automation was adapted with different MALs for different vehicle types.

The hybrid ferry BERLIN (Scandlines shipping company) was equipped with a MAS with tools for motion prediction and consumption optimisation in MAL1 [12]. The ferry, with its defined route between Germany and Denmark and tight schedule, is supported by MAS to speed up the berthing processes and make them more efficient. In order to establish this MAS, models for the dynamic motion behaviour and the engine processes have been developed. For the future personalisation of the MAS user interface via the menu, the nautical staff was consulted.

Currently automatic manoeuvring in MAL2 is prepared with the digitised German research vessel DENEB. Automatic berthing in the port of Rostock is planned, which should firstly test in the open sea. The same berthing manoeuvres but with actuator failures or obstacles in the planned path should be realised with small unmanned surface vehicles in preparation of MAL3.

The introduction of highly automatic and autonomous systems can also be seen in light of the challenges of a demographic transition. A transformation that should include the support of an aging and changing workforce, especially with a younger generation expecting a more digitalized and challenging working environment. The technological transformation should be done in a manner that the existing workforce does not lose their abilities - such a de-skilling could lead to more accidents and harm for people and infrastructure.

So, the introduction of automatic and autonomous systems should take the workers along with it, clearly defining responsibilities as in the Manoeuvre Automation Levels is one aspect of this.

2 METHODS

The aim of this research is, for the planning algorithm, to directly learn the fulfilment of an objective from using the control inputs of a vessel, without the intermediate and additional step of path planning.

Methods for this kind of end-to-end learning for autonomous movement planning are manifold.

One, more mathematical, way would be the use of the calculus of variations - augmenting a trajectory $y_p(t)$ by a set of parameters p to describe the systems response to the environment. One then minimizes a Functional $J(y_P)$ with respect to the parameter set p. For example, the functional $J(y_p)$ can be a path integral along the trajectory $y_p(t)$. This approach can become very difficult very easily, because it needs to evaluate derivates of the functional $J(y_p)$ with respect to the parameter-set p analytically.

Another, more recent, approach - from artificial intelligence - for adaptive control of dynamical systems is the use of recurrent neural networks (RNN), where one constructs a parametrised graph - a network of simple arithmetic operations - describing the systems response to input and the computation of the objective function. The solution can then be found by repeatedly using backpropagation to compute gradients with respect to the objective function and successively get new search directions for iterative parameter updates - leading to new weights in the neural network that over time will minimize the objective function.

A third and relatively new option is differentiable programming. It allows to write down the systems parametrised response to control inputs, the resulting trajectory and the subsequent calculation of the objective function as imperative source code and to use automatic syntax transformations and an analysis of the information flow while running the program to compute gradients that can be used as new search directions in the parameter space. Repeatedly re-running the code and following the gradients as search directions enables to minimize the objective function and allows to reach the programmed goal of the mission.

The research question we investigated was: How far does one get, using differentiable programming for computing the search directions and simple gradient descent to minimise the objective function (called loss function in machine learning) to enable the autonomous movement-planning of a simulated vessel?

As a framework for our foray into differential programming we use the software DiffTaichi, that was presented at the International Conference on Learning Representations in 2020 [4]. The Code is available on GitHub [3].

2.1 *A Cursory Overview*

Using the framework DiffTaichi allows to write plain (but annotated) Python code describing the simulated physics of the vessel, the parametrised response of the ship to control inputs and the objective of the ship's movement in an imperative coding style. The resulting source code will be transformed using an abstract syntax tree (AST), the internal representation of the source code, before it is compiled for the underlying architecture (CPU, GPGPU, OpenCL).

In DiffTaichi the sequence of execution steps can be recorded during a program run on what is called a tape. Using the tape of recorded instruction from a specific simulation run we can compute a partial derivate of the calculated loss - with respect to the parameters - to calculate the gradient as a new search direction for the next optimization step in the parameter space.

To a very rough approximation a program in DiffTaichi might look like the following:

```
import taichi as ti

# define variables
loss = ti.var(dt=ti.f32)
system_state = ti.var(dt=ti.f32)

# place the variables in memory
ti.root.place(system_state)
ti.root.place(loss)

# set flag to compute gradients
ti.root.lazy_grad()

# define a kernel
# describing the system
@ti.kernel
def run_system():
  # evolve system_state
  # for example, by time-stepping

# define the objective functions
@ti.kernel
def compute_loss():
  # compute the loss
  # from system_state

# run the time-stepping scheme
run_system()

# compute the objective (loss)
with ti.Tape(loss):
  compute_loss()
```

Running that program once we will get a gradient *"system_state.grad"*. This gradient can be used for further computations, for example as a search direction in the parameter space to minimize the objective function. Running the program repeatedly while updating parameters and recomputing the parameterized system's trajectory accordingly we are enabled to reach the goal encoded in the objective function.

2.2 *Physical Model*

The DiffTaichi kernel *"run_system"*, to be provided by us, has two parts to it, a physical model and a time stepping scheme to evolve the systems state.

We choose to describe the physical state of our simulated vessel by its position (x, y coordinates in an ENU-frame centred at the point of departure), speed through water (v), heading (φ) as well as its turn rate (ω).

During the simulated motion of the vessel the autonomous movement planning algorithm gets to control the ships engine power and the position of the rudder. Therefore, the system is underactuated. As a simplification we assume that the control inputs affect the engines state and rudder position without any delay. By changing the power of the engine and therefore the thrust, the algorithm can accelerate or decelerate the vessel. By controlling the rudder, the turn rate can be influenced, as long as the ship is moving relative to the water.

To model the ship's acceleration or deceleration, we have to include drag forces on the hull. The drag force has a hydrodynamic component, but is dominated by wave-making, so we approximate it to scale proportional to the 3rd power of the speed through water (v), with the proportionality factor being the vessels hull speed and a drag coefficient.

For controlling the ship's turn-rate the autonomous movement planning algorithm gets to control the rudder's position. To compute the forces acting on the rudder and therefore the angular acceleration, we use the approximate method for calculating the forces from Żelazny published in [14].

To complete the physical model for the simulation we need some static parameters, like, the density of water, but also the ship's mass, width, length overall and the length of its water line (and therefore the hull-speed). For the purpose of our calculations we took the data from DLR's research vessel Aurora (Table 1).

Table 1.DLR Research Vessel Aurora

Length	7 [m]
Width	2.54 [m]
Ship mass	2800 [kg]
Length of waterline	6.5 [m]
Hull-speed	$1.25 \cdot \sqrt{lwl} = 3.2$

Figure 3. DLR Research Vessel Aurora

Additionally, to compute changes in the turn rate (the angular acceleration) from rudder forces we need an approximation of the ships rotational moment of inertia. This moment will change depending on mass distribution, e.g. how the vessel is loaded - for example as passengers move. As a first approximation we used the momentum of inertia of a cuboid of the same volume and weight. The angular moment of inertia of the vessel itself could be measured when the ship is craned out of or into the water to refine this estimation.

Using the systems state and the approximated vessel dynamics, we get a system of 5 ordinary differential equations in 5 variables (position: x, y, velocity through water: v, heading: φ, rate of turn: ω):

$$\dot{x} = v \cdot \sin(\varphi) + w_x$$
$$\dot{y} = v \cdot \cos(\varphi) + w_y$$
$$\dot{\varphi} = \omega$$
$$\dot{v} = \frac{F_{thrust} - F_{drag}}{m}$$
$$\dot{\omega} = \frac{l \cdot F_{turn}}{I_{rot}}$$

Where I_{rot} is the ship's rotational moment of inertia and l is the distance of the rudder from the vessel's centre of mass.

We solve this system numerically by integrating it using an explicit Euler scheme of first order. This integration also has to be implemented in DiffTaichi specific source code to be part of the differentiable program.

In addition, we follow the authors of the DiffTaichi paper in making sure to have an additional time step at zero crossings of v, as in those points the thrust changes sign. This addition is important to get meaningful derivatives of any objective function with respect to the parameter inputs when zero crossings occur. In the DiffTaichi paper [9] it is called the time of impact (TOI) as the authors are trying to make a roboter walk.

2.3 *Objective Functions*

The kernel "*compute_loss*" we showed in the cursory overview is an objective function one has to select in such a way as to suitably encode the goal of the planning algorithm.

During our research of differential programming for autonomous movement planning we used increasingly demanding and complex objective functions.

We started simple, with an objective that minimizes the distance to the opposite shore:

```
@ti.kernel
def compute_loss_arrival_beach(i: ti.i32):
 loss[None] = integral[i]
             + (y[i]-goal_y)**2
```

The variable *i* in this case indicates at what time-step this objective functions is being evaluated. As shown before this kernel usually will be evaluated after the last time-step.

The "*integral[i]*" in the example above will be used to optimize certain aspects of the overall trajectory, for example energy consumption, travel time or the length of the trajectory. It could also be used to penalize strong variations in controlling inputs or the velocity to smooth the overall sailing experience.

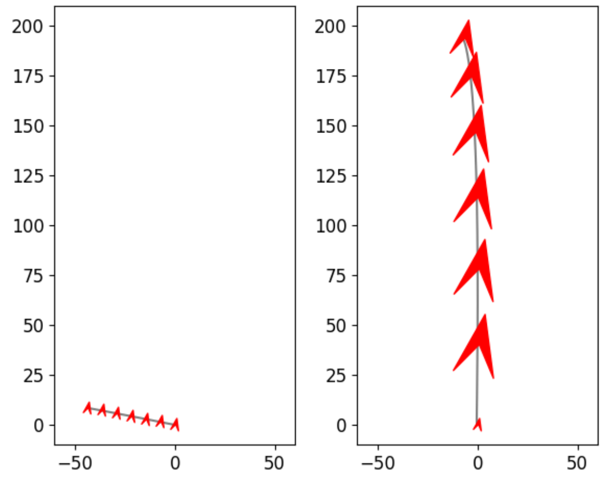

Figure 4 Objective: crossing of the flowing river, drifting in the first run, discovering the use of thrust to cross (the arrow's size indicates the relative velocity through water)

3 RESULTS

Now that we have a fully differentiable model, we can put it in an optimization loop, where we run the explicit time-stepping to get a full trajectory, calculate the objective function (the loss) with the computed trajectory as an input and get back gradients as new search direction for updated controlling inputs that will minimize the objective after a few runs.

After we got the basic algorithm working (e.g. the ferry discovering how to cross the river) of course we want the vessel to actually arrive where it can berth (at the Position with the coordinate pair: *goal_x, goal_y*):

```
@ti.kernel
def compute_loss_arrival_point(i: ti.i32):
 loss[None] = integral[i]
             + (y[i]-goal_y)**2
             + (x[i]-goal_x)**2
```

We might also find it helpful for the ship to arrive with almost now relative speed to the quay to not damage port infrastructure or injure passengers and also to allow an automatic docking procedure to take over:

```
@ti.kernel
def compute_loss_arrival_speed(i: ti.i32):
 loss[None] = integral[i]
   + (x[i]-goal_x)**2 + (y[i]-goal_y)**2
   + c1 * ((ti.sin(phi[i])*v[i]+flow_x)**2
          +(ti.cos(phi[i])*v[i]+flow_y)**2)
```

Where *c1* is a penalty constant to scale the contribution of the final velocity to the overall loss appropriately. In our experiments we fixed *c1* at 100.

To help passengers to get off our ferry safely we prefer the vessel to arrive alongside the quay, so we encode that in an objective too:

```
@ti.kernel
def compute_loss_arrival_head(i: ti.i32):
 loss[None] = integral[i]
  + (x[i]-goal_x)**2 + (y[i]-goal_y)**2
  + c1 * ((ti.sin(phi[i])*v[i]+flow_x)**2
         +(ti.cos(phi[i])*v[i]+flow_y)**2)
  + c2 * ti.cos(phi[i])**2
```

We use *c2* is a penalty term to scale the final heading's contribution of the to the overall loss function appropriately. In this experi-ments we fixed *c2* at 1000.

Using these increasingly complex objective functions allows the autonomous planning algorithm to reach more complex goals successively (see Figure 5).

3.1 *Collision Avoidance*

Apart from arriving at the correct place, with almost no speed and having a compatible orientation or heading we might want to optimize certain characteristics of the trajectory along its path.

For example, this could be the minimization of the ship's energy consumption, but we can also use it to ensure that we do not run into other ships while we fulfil the objective of crossing the river.

To further this aim, we defined exclusion zones (for example ship domains) that our vessel should not come close to or enter. Of course, these exclusion zones move together with the vessels defining them when they advance. So, for each step along its path, the planning algorithm will be penalized for being in such a - possibly progressing - exclusion zone, or being too close to one.

In principle it enables the computation of gradients of the overall algorithm if such a penalty function as a component is differentiable itself. Also, we found that it assists the convergence of the overall algorithm if the penalty function does not have a pole where the obstacle is. In our current research we settled with a Gaussian function centred a little in front of the obstacle but including it within a standard deviation.

```
# penalty term for being too close
# to the crossing vessel (at any time)
penalty_cpa = ti.exp(
 -((x-exclusion_x)**2/var_x +
   (y-exclusion_y)**2/var_y +
 2*(x-exclusion_x)*(y-exclusion_y)/cov_xy )
```

When we defined the several objective (loss) functions above they included an "*integral[i]*" term that will be used to sum this varying penalty contributions along the trajectory of the vessel - especially including the closest point of approach (CPA) - and combine it with the aim of minimizing the time travelled, the overall length of the trajectory and the overall variation in speed:

```
# this objective will be integrated
# along the path of the ferry
integral[i] = integral[i-1] + delta_t * (
 weight_cpa_penalty * penalty_cpa +
 weight_distance * ((d_x)**2+(d_y)**2) ) +
 weight_integral_speed * (v**2) )
```

As an illustration, the moving exclusion zone for a crossing ship in the waterway is signified by the blue ellipses in Figure 5.

After including these penalty terms, we ran the optimization loop again. The convergence was a lot slower than for those computations that did not include a moving obstacle in the objective function. We also see, that the propagation of gradients for the controlling inputs from the end of the trajectory to the early movement phases could be improved.

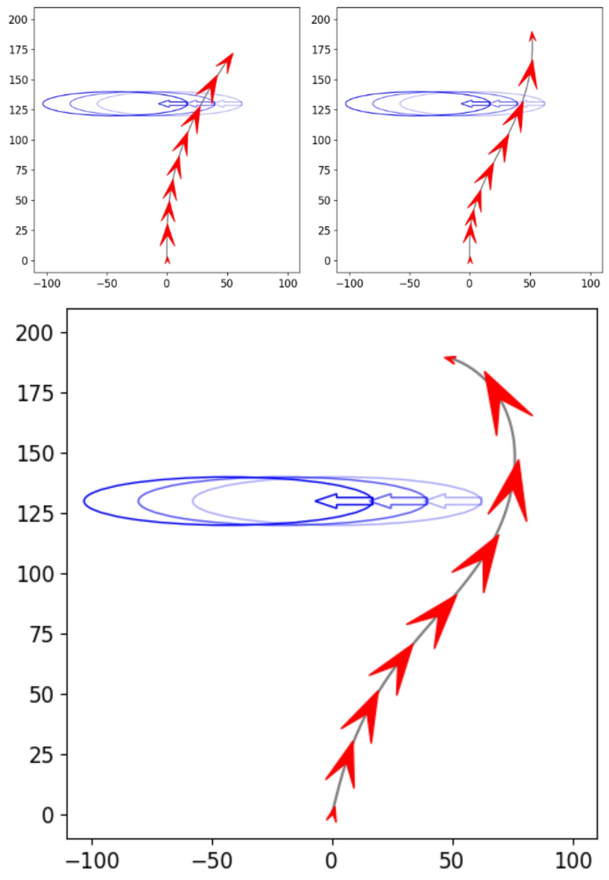

Figure 5. The vessels trajectory after 1024, 8129 and 16384 optimizations steps. The ellipses signify the crossing vessel's exclusion zone.

4 CONCLUSION

Our numerical experiments show that differentiable programming can be used for autonomous movement planning with a simple model of a river crossing ferry. The algorithm allows the vessel to flexibly interact with and react to its environment for example avoiding another vessel crossing its path.

The algorithm learned from the physical model, for example goal-oriented steering and the selection of its initial direction for an optimal departure.

Actually expected, but still encouraging, the algorithm independently discovered the use of reverse thrust shortly before docking to reduce its relative speed. Surprisingly the control commands (for power and steering) were very smooth and do not have abrupt discontinuities.

Possible lines of research that we envision for the further development of the method are an improved selection of search directions and step sizes when updating model parameters from gradients.

As we saw above the algorithm somewhat suffers from diminishing gradients along the path especially when avoiding a crossing obstacle. Necessary changes in the controlling inputs only slowly propagate from the end of the trajectory to the early phases. We want to investigate how this can be overcome by selecting a sufficient initial trajectory.

Additionally, we want to increase the speed of convergence in general. To accomplish this, we will have a look at different optimizations algorithm like Stochastic Gradient Descent (SGD) [1, 5, 9] and Adam (using adaptive moment estimation) [6].

For the purpose of evaluating and validating our autonomous movement planning algorithm, it should be connected to a ship handling simulator, that uses its own physical model and includes process and measurement noise. While directing the ship towards its goal, the planning algorithm would then be run iteratively to re-plan future movements and controlling inputs from the current state (which advances continuously), while using a method to repeatedly estimate the vessels state and parameters.

The overall program, incorporating our autonomous movement planning, will have to repeatedly estimate position, speed, heading and turn rate, while the algorithms parameters to be estimated include the vessel's weight (and distribution), draught, momentum of inertia as well as external influences like water currents.

Of course, the ship will need updated estimates for the position, speed and heading of other vessels to take their future trajectories into account. This could be improved upon by having multiple ships interact with each other cooperatively - for example by using planning algorithms that inform each other about their planned trajectory and finding optimal trajectories for all ships involved.

When optimising the motion behaviour of a single ship, the local environment, the specific dynamic motion model und the engine states can be considered to find an optimal path or trajectory. The COLREGs (Collision avoidance regulations) have to be part of the applied algorithms to integrate an automatically controlled vehicle into the local traffic situation with a mixture from surface vehicles with different automation level and degree of manoeuvrability.

The original manoeuvre trajectory has to be changed and optimised according the current situation. In consequence, the resulting calculated manoeuvre can be more a stopgap than an energetically optimal solution with minimal consumption and emission. Therefore, cooperative approaches should be investigated that optimise the cooperative motion behaviour of several ships or a fleet to achieve the best result for this group of ships.

Finally, within the framework of uncertainty quantification one could use the distribution in uncertain parameters, for example: non-linearities in the ship systems response, (future) trajectories of other vessels, to get a risk assessment for possible movement plans select an appropriate course of action.

REFERENCES

1. Bottou, L., Curtis, F.E., Nocedal, J.: Optimization Methods for Large-Scale Machine Learning. (2018).
2. Hesselbarth, A., Medina, D., Ziebold, R., Sandler, M., Hoppe, M., Uhlemann, M.: Enabling Assistance Functions for the Safe Navigation of Inland Waterways. IEEE Intelligent Transportation Systems Magazine. 12, 3, 123–135 (2020). https://doi.org/10.1109/MITS.2020.2994103.
3. Hu, Y.: DiffTaichi: Differentiable Programming for Physical Simulation, https://github.com/yuanming-hu/difftaichi, last accessed 2021/04/27.
4. Hu, Y., Anderson, L., Li, T.-M., Sun, Q., Carr, N., Ragan-Kelley, J., Durand, F.: Differentiable Programming for Physical Simulation. Presented at the International Conference on Learning Representations (2020).
5. Kiefer, J., Wolfowitz, J.: Stochastic Estimation of the Maximum of a Regression Function. The Annals of Mathematical Statistics. 23, 3, 462–466 (1952). https://doi.org/10.1214/aoms/1177729392.
6. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. Presented at the 3rd International Conference for Learning Representations , San Diego (2015).
7. Lloyd’s Register Groupe: Design Code for Unmanned Marine Systems. (2017).
8. Medina, D., Vilà-Valls, J., Hesselbarth, A., Ziebold, R., García, J.: On the Recursive Joint Position and Attitude Determination in Multi-Antenna GNSS Platforms. Remote Sensing. 12, 12, (2020). https://doi.org/10.3390/rs12121955.
9. Robbins, H., Monro, S.: A Stochastic Approximation Method. The Annals of Mathematical Statistics. 22, 3, 400–407 (1951).
10. Rødseth, Ø., Nordahl, H.: Definition of autonomy levels for merchant ships, Report from NFAS, Norwegian Forum for Autonomous Ships. (2017). https://doi.org/10.13140/RG.2.2.21069.08163.
11. SAE International: J3016B: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - SAE International, https://www.sae.org/standards/content/j3016_201806/, last accessed 2021/04/27.
12. Schubert, A.U., Kurowski, M., Damerius, R., Fischer, S., Gluch, M., Baldauf, M., Jeinsch, T.: From Manoeuvre Assistance to Manoeuvre Automation. In: Journal of Physics: Conference Series. , Trondheim, Norway (2019). https://doi.org/10.1088/1742-6596/1357/1/012006.
13. Schubert, A.U., Kurowski, M., Gluch, M., Simanski, O., Jeinsch, T.: Manoeuvring Automation towards Autonomous Shipping. In: Proceedings of the International Ship Control Systems Symposium (iSCSS). , Glasgow, UK (2018). https://doi.org/10.24868/issn.2631-8741.2018.020.
14. Żelazny, K.: Approximate Method of Calculating Forces on Rudder During Ship Sailing on a Shipping Route. TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation. 8, 3, 459–464 (2014). https://doi.org/10.12716/1001.08.03.18.
15. Ziebold, R., Gewies, S.: Long Term Validation of High Precision RTK Positioning Onboard a Ferry Vessel Using the MGBAS in the Research Port of Rostock. TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation. 11, 3, 433–440 (2017). https://doi.org/10.12716/1001.11.03.06.