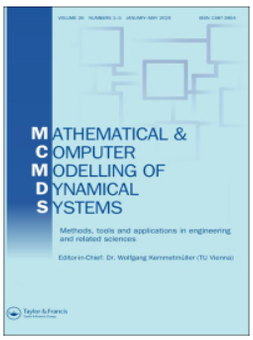


Taylor & Francis
Taylor & Francis Group

Mathematical and Computer Modelling of Dynamical Systems

Methods, Tools and Applications in Engineering and Related Sciences

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/nmcm20

Autonomous navigation of ships by combining optimal trajectory planning with informed graph search

Luis Lüttgens, Benjamin Jurgelucks, Heinrich Wernsing, Sylvain Roy, Christof Büskens & Kathrin Flaßkamp

To cite this article: Luis Lüttgens, Benjamin Jurgelucks, Heinrich Wernsing, Sylvain Roy, Christof Büskens & Kathrin Flaßkamp (2022) Autonomous navigation of ships by combining optimal trajectory planning with informed graph search, Mathematical and Computer Modelling of Dynamical Systems, 28:1, 1-27, DOI: 10.1080/13873954.2021.2007138

To link to this article: https://doi.org/10.1080/13873954.2021.2007138

Published online: 03 Feb 2022.

MATHEMATICAL AND COMPUTER MODELLING OF DYNAMICAL SYSTEMS
2022, VOL. 28, NO. 1, 1–27
https://doi.org/10.1080/13873954.2021.2007138

Taylor & Francis
Taylor & Francis Group

Autonomous navigation of ships by combining optimal trajectory planning with informed graph search

Luis Lüttgens*, Benjamin Jurgelucks*, Heinrich Wernsing*, Sylvain Roy*, Christof Büskens* and Kathrin Flaßkamp*

Center for Industrial Mathematics, Optimization and Optimal Control, University of Bremen, Bremen, Germany

ABSTRACT

Autonomous trajectory generation plays an essential role in the navigation of vehicles in space as well as in terrestrial scenarios, i.e. in the air, on solid ground, or water. For the latter, the navigation of ships in ports has specific challenges since ship dynamics are highly nonlinear with limited agility, while the manoeuvre space in ports is limited. Nevertheless, for providing support to humanly designed control strategies, autonomously generated trajectories have not only to be feasible, i.e. collision-free but shall also be optimal with respect to manoeuvre time and control effort. This article presents a novel approach to autonomous trajectory planning on the basis of precomputed and connectable trajectory segments, the so-called motion primitives, and an A*-search algorithm. Sequences of motion primitives provide an initial guess for a subsequent optimization by which optimal trajectories are found even in terrains with many obstacles. We illustrate the approach with different navigation scenarios.

ARTICLE HISTORY
Received 4 May 2021
Accepted 11 November 2021

KEYWORDS
Motion planning; optimal control; autonomous navigation; A* planning; ship dynamics

1. Introduction

Today, autonomous transportation is one of the most active research areas all over the world. Certainly, the most popular branch deals with self-driving cars. However, autonomous automobiles cover only a small fraction of transportation problems, since they are mostly the solution for medium-ranged private transportation. A huge task for the logistics industry is global trade. According to [1,2,3] about 90% of all goods are carried by ships today. Thus, the development of techniques for autonomous ship navigation is of great interest, too. Thereby, ship captains can be assisted in navigation tasks, which could traditionally only be solved by well-trained and experienced personnel. Thus, this article focuses on the autonomous[1] navigation of ships by providing feasible and, even more, optimal solutions in scenarios with challenging constraints. A sophisticated

CONTACT Kathrin Flaßkamp kathrin.flasskamp@uni-saarland.de Systems Modeling and Simulation, Saarland University, 66123 Saarbrücken, Germany

***Present address**: L. Lüttgens: Robert Bosch GmbH, 70839 Gerlingen, Germany; B. Jurgelucks: Mathematical Optimization, Humboldt-Universität zu Berlin, 10099 Berlin, Germany; H. Wernsing: Harris Orthogon GmbH, 28207 Bremen, Germany; S. Roy: LASE Industrielle Lasertechnik GmbH, 46485 Wesel, Germany; C. Büskens: Center for Industrial Mathematics, Optimization and Optimal Control, University of Bremen, 28359 Bremen, Germany; K. Flaßkamp: Systems Modelling and Simulation, Saarland University, 66123 Saarbrücken, Germany.

modelling is an indispensable prerequisite to allow for mathematical optimization. Classically, ship dynamics are modelled by ordinary differential equations, since for this modelling approach, established simulation techniques are at hand. However, trajectory optimization then requires nonlinear optimization techniques, which, in general, rely on good initial guesses and are computationally costly. This paper does not address the algorithmic barriers in optimization, but, instead, we reconsider the modelling step, i.e. propose and analyse a model abstraction that allows a combined application of state-of-the-art optimization techniques for discrete and continuous models, respectively.

The method's development has been motivated by the research at the University of Bremen within the project *GALILEOnautic* [4,5,6]. This project makes use of the global navigation satellite system *GALILEO* for manoeuvring in safety-critical areas such as ports. Optimized manoeuvring and automated navigation are realized by online-optimization using the software *WORHP* and *TransWORHP* [7,8,9]. Further aspects to investigate are cooperation of ships, model-based (feedback) control design, and a virtual reality 3D test environment. The *GALILEOnautic* project plans an application to the ferry crossing from the port of Rostock in Germany to Gedser in Denmark, a port with demanding waters.

The proposed approach within this article is based on the idea of *motion planning with motion primitives* by Frazzoli et al. [10]. In the offline phase of the method, a set of motions which is consistent with the given vehicle dynamics, the so-called motion primitives, are computed. The term 'primitive' originally refers to the motions being simple in the sense that humans would intuitively choose them to e.g. stear a vehicle. In a formal setting, primitives can be selected by model-based criteria as it is done in optimization, e.g. the primitive duration or its energy efficiency. Exploiting symmetries in the model allows to use and combine these primitives in various ways. Thus, during the online phase, an optimal sequence of primitives is searched which solves the given planning task by using a modified A* method. The size of the library of motion primitives is crucial for online-applicability. However, a coarse representation of the ship model by only using a few number of motion primitives generates suboptimal solutions. We address this issue by using the sequence of motion primitives as an initial guess [11,12] and apply an optimal control method using the full nonlinear dynamical model afterwards. Due to the sophisticated initial guess, the local solver of the direct optimal control method is more likely to converge quickly (cf. to the discussion and numerical studies in [11] or [13], for instance). We apply this approach to a ship model. A crucial challenge for the planning phase as well as for the post-optimization are restrictions due to environmental constraints. Using an occupancy grid to model port layouts, we guarantee that the motion primitive sequence and the post-optimized trajectory are feasible solutions.

The combination of A* planning with optimal control overcomes several shortcomings of the individual methods and thus provides a powerful approach to real-world applications:

- While classical planning focuses on geometric paths only (which are not necessarily always feasible), a sequence of motion primitives is feasible with respect to the (nonlinear) dynamics model, thus the path can be realized with less corrective feedback control.

- Using curves that combine states and controls within the A* (instead of sampling the control space only), we always have the state trajectory and control curve at hand which might be helpful for designing feedback controller to robustify solutions.
- Time-consuming optimal trajectory design can be done offline in the preparation phase. The optimized manoeuvres are stored in the library and can be used sequentially by the A* planning.
- Providing a sophisticated initial guess to the local optimization method within the nonlinear optimal control is likely to speed up convergence. Moreover, since the primitive sequence is an admissible solution, it can always be used as a fallback, if, for instance in a real-time Model Predictive Control (MPC) scheme, an optimal control solution cannot be provided in time.

Path planning for vehicles by concatenation of primitives has a long history dating back to Dubin's car, where arcs of circles have been used to find shortest paths, see [14] and also Reed's and Shepp's curves as a crucial extension [15]. A number of further extensions and variants are e.g. listed in the textbook [16]. Motion planning with motion primitives, as introduced by Frazzoli et al. in [10,17,18], also falls into the class of planning methods. In [19] a survey on planning methods suitable for applications to autonomous vehicles was conducted.

Optimal control, as a field of mathematical research, emerged from studying variational problems, see e.g [20] for a historical review. However, it required improved numerical techniques from the past 50–60 years in order to make optimal control applicable to real-world examples. Among different numerical approaches, direct optimal control has shown great applicability to large-scale problems [21]. A direct transcription of the optimal control problem into a nonlinear constrained optimization problem allows to use efficient NLP solvers such as IPOPT [22], KNITRO [23], SNOPT [24], or WORHP [7].

A detailed study of the numerical methods is out of the scope of this article. Instead, we focus on conceptually showing the combination of planning and optimal control methods in ship navigation.

Planning and optimization in ship manoeuvring was studied e.g. in [25,26,27,28,29]. See [29] for a recent research overview on ship collision avoidance methods, which identifies motion prediction as one of the crucial methods for collision prevention. In [30], for instance, a modelling approach based on field theory and particle simulations for collision avoidance is presented. Typically, collision avoidance considers much shorter manoeuvres (in time, as well as in deplacement) than our path planning approach [31]. also considers collision avoidance scenarios instead of large planning tasks. However, they build their solutions by pieces of trajectories and allow continuation of some pieces. This resembles our formalism of trim primitives and manoeuvres, presented in detail in the following. Contrarily to collision-avoidance, in [32] formation control of multiple autonomous vessels is addressed. The focus is on feedback control in order to address environmental disturbances induced by waves, wind and ocean currents. In e.g [26]., the focus lies on the sensor aspects, which are not covered in our work.

In [33], an overview on planning and navigation for vessels and sailboats is given. The presented methods for collision avoidance are based on discretized manoeuvres, thus sharing some similarities with our approach. The path planning literature survey is

focused on sailboats. It lists graph-based search methods in contrast to potential field methods and classical sailing. Closely related to our work [34], also considers autonomous navigation tasks of vessels and proposes a discrete planning method, called fast marching method. While this method is originally a pure path planning method, the authors integrator ship kinematics. Then, a PID controller has to be designed in order to optimize the trajectory. Here, our post-optimization approach provides more flexibility due to a general nonlinear optimization solver. However, we only provide an open-loop solution, not a feedback controller.

As it is the case in our approach, the kinematics and dynamics model and also similar objective functions, e.g. control effort, are considered in [28] for planning for autonomous marine vehicles. However, they apply a projection operator method instead of our two-step approach of planning and direct optimal control. While their focus lies on computing collision-free manoeuvres for multiple vessels, we focus on navigation in ports; this makes it hard to compare the methods directly. In [25], planning on a base of trajectories is performed. In contrast to our approach, the trajectories do not stem from so-called trim primitives and optimal manoeuvres. Moreover, we use different optimization methods (nonlinear gradient-based versus ant-colony optimization). Another quite different approach is based on a potential flow design for path planning as presented in [27]. Autonomous navigation has been successfully realized for a sailboat as reported in [35]. Here, the focus is on the overall implementation and less on the guidance task. Within the proposed control architecture, our motion planning with primitives approach could provide optimal guidance trajectories as high-level control inputs.

1.1. Outline

We proceed by introducing a ship model in Section 2. Then, a formal definition of motion primitives follows, and it is applied to compute symmetry and primitives of the ship model in Section 3. In Section 4, we design a manoeuvre automaton for the ship model, which forms the basis for our planning method. Also, the A* algorithm and the occupancy map generated for port environments are introduced. The method is evaluated in several scenarios with numerical results presented and discussed in Section 5. Finally, we give concluding remarks in Section 6.

2. Modelling

We are using the ship model proposed in [36] with the same choice of parameters,

$$\dot{x}(t) = u(t)\cos(\Psi(t)) - v(t)\sin(\Psi(t)) \tag{1}$$

$$\dot{y}(t) = u(t)\sin(\Psi(t)) + v(t)\cos(\Psi(t)) \tag{2}$$

$$\dot{\Psi}(t) = r(t) \tag{3}$$

$$X(t) = \big(\dot{u}(t) - v(t)r(t) - r(t)^2 x_G\big)m \tag{4}$$

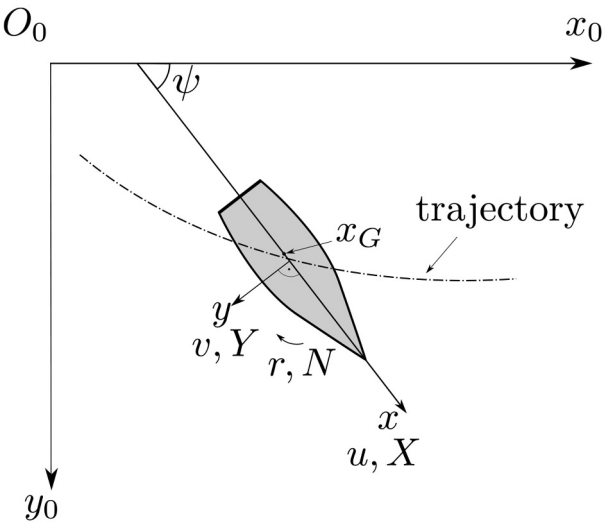


Figure 1. Introduction of coordinates and forces for the ship model (adapted from [36]).

$$Y(t) = (\dot{v}(t) + u(t)r(t) + \dot{r}(t)x_G)m \tag{5}$$

$$N(t) = \dot{r}(t)I_{zz} + (\dot{v}(t) + u(t)r(t))x_G m \tag{6}$$

If we neglect Equations (4)–(6), we are left with first-order kinematic Equations (1)–(3). The evolution of the spatial displacements x_0 and y_0 depend on the longitudinal and lateral velocities u and v, and the current heading Ψ. The angular velocity is denoted by r. Considering u, v, r as the control inputs, this model resembles simple vehicle models, also known as holonomic robots, for instance (cf [37]). The system can instantaneously start to move into any direction independent of the current orientation. This is in contrast to a nonholonomic model, which includes constraints given by wheels.

Compared to street or aeronautic vehicles, a ship is far less agile. This is modelled in Equations (4)–(6). Adding these equations to our model gives a system of six ordinary differential equations. Here, m represents the mass of the ship, x_G denotes the centre of mass along the x-component of the coordinate system attached to the ship (see Figure 1).

In [36], a port-starboard symmetry is assumed and the ship coordinate system's origin is chosen to lie on this symmetry axis. Thus, the centre of mass must lie on this axis, i.e. y_G is equal to zero. The term I_{zz} denotes the moment of inertia with respect to the z-axis.

The capital letters X, Y denote forces and a N a moment acting on the ship. The ship model we are using in this article is independent of external forces such as water depth, wind, or time history effects, and there is no explicit dependency of the position of the ship (see [36]). The control inputs of the full model are the propeller thrust and the rudder angle, again we refer to [36] for details.

3. Motion primitives for a ship model

3.1. General definitions

Many dynamical systems show symmetries. Here, we are interested in continuous symmetries that express themselves in terms of invariances. Illustrative examples are given by translational or rotational invariances of a mechanical system in the plane. We consider a dynamical system on an n-dimensional manifold M, given by $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t))$ with $\mathbf{x}(t) \in M \subseteq \mathbb{R}^n$ and $\mathbf{u}(t) \in \mathbb{R}^m$. Let $\mathcal{T}M$ denote the tangent bundle of M. We assume $f : M \times \mathbb{R}^m \to \mathcal{T}M$ to be continuous and locally Lipschitz w.r.t. its first argument in order to guarantee existence and uniqueness of the solution $\varphi_u(\cdot; \mathbf{x}^0)$ on a compact time interval $[0, T]$, $0 < T < \infty$, for $u \in \mathcal{L}^1([0, T], \mathbb{R}^m)$. $\mathcal{L}^1([0, T], \mathbb{R}^m)$ denotes the space of Lebesgue-measurable and absolutely integrable functions on the domain $[0, T]$. We recall from [38] the following definitions.

Definition 3.1 (Symmetry Group). Let M be a smooth manifold, $(G, \circ)$ a Lie-group, and Γ a left-action of G on M. Then, we call the triple (G, M, Γ) a *symmetry group* of the system $\underline{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t))$ if the property

$$\varphi_u(t; \Gamma(g, \vec{x}_0)) = \Gamma(g, \varphi_u(t; \vec{x}_0)) \qquad (t, g, \vec{x}_0) \in \mathbb{R}_{\geq 0} \times G \times M \tag{7}$$

holds for all $u \in \mathcal{L}^1([0, T], \mathbb{R}^m)$.

Definition 3.2 (Motion Primitive). Let (G, M, Γ) be a symmetry group. Then, two trajectories $\varphi_u(\cdot; \vec{x}_0)$ and $\varphi_u(\cdot; \vec{y}_0)$ are called *equivalent*, if there exists $g \in G$ such that

$$\varphi_u(t; \vec{x}_0) = \Gamma(g, \varphi_u(t; \vec{y}_0)) \qquad t \geq 0.$$

A *motion primitive* is the equivalence class of all trajectories equivalent to $\varphi_u(\cdot; \vec{x}_0)$ w.r.t. the left action Γ.

Note that the same control function u is assumed for all members of a motion primitive.

Definition 3.3 (Trim Primitive). Let (G, M, Γ) be a symmetry group and let g denote the associated Lie algebra of G. Then, a trajectory $\varphi_u(\cdot; \vec{x}_0)$ is called a *trim primitive if there exists a* Lie algebra element $\xi \in g$ such that

$$\varphi_u(t; \vec{x}_0) = \Gamma(\exp(\xi t), \vec{x}_0) \quad \text{and} \quad u(t) \equiv \bar{u} = \text{const.} \qquad t \in [0, T].$$

The original definition of trim primitives goes back to Frazzoli in [10]. Trim primitives can be interpreted as the extension of relative equilibria (see e.g. [39],) to systems with control inputs. Loosely speaking, trim primitives are simple motions (despite nonlinear dynamics), which are generated by the symmetry action. The control input has to stay constant along a trim primitive. In fact, the 'trimmed input' is the reason why these motion primitives are called trim primitives or trims for short.

3.2. Symmetry for ship models

The kinematic ship model

$$\dot{\vec{x}}(t) = \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\Psi}(t) \end{pmatrix} = \begin{pmatrix} u(t)\cos(\Psi(t)) - v(t)\sin(\Psi(t)) \\ u(t)\sin(\Psi(t)) + v(t)\cos(\Psi(t)) \\ r(t) \end{pmatrix} =: f(\vec{x}(t), \vec{u}(t))$$

on $M = \mathbb{R}^2 \times S^1$ and with control $\vec{u} = (u, v, r)^T$ is invariant w.r.t. pure translations as well as w.r.t. a combination of rotations and translations, where the latter is less obvious when just inspecting the equations of motions. A candidate for the symmetry group can be represented in homogeneous coordinates as

$$G := \left\{ A \in \mathbb{R}^{4\times 4} : A = \begin{pmatrix} \cos(\Delta\Psi) & -\sin(\Delta\Psi) & 0 & \Delta x \\ \sin(\Delta\Psi) & -\cos(\Delta\Psi) & 0 & \Delta y \\ 0 & 0 & 1 & \Delta\Psi \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta\Psi \end{pmatrix} \in \mathbb{R}^3 \right\} \tag{8}$$

with group action $\Gamma : G \times M \to M$ acting by matrix multiplication on the homogeneous representation of $\vec{x} = (x, y, \Psi)^T \in M$, i.e. for $A \in G$

$$\Gamma(A, \vec{x}) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \cos(\Delta\Psi) & -\sin(\Delta\Psi) & 0 & \Delta x \\ \sin(\Delta\Psi) & -\cos(\Delta\Psi) & 0 & \Delta y \\ 0 & 0 & 1 & \Delta\Psi \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ \Psi \\ 1 \end{pmatrix}.$$

Proposition 3.4. *The kinematic ship model is invariant w.r.t. symmetry* (G, M, Γ).

Proof. Note that the ship model can be written as a matrix-vector multiplication, i.e.

$$\begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\Psi}(t) \end{pmatrix} = \begin{pmatrix} \cos\Psi & -\sin\Psi & 0 \\ \sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ r \end{pmatrix}$$

and the ODE's analytic solution is given by

$$\begin{pmatrix} x(t) \\ y(t) \\ \Psi(t) \end{pmatrix} = \varphi_\mu \left(\begin{pmatrix} x_0 \\ y_0 \\ \Psi_0 \end{pmatrix}, t \right)$$

$$= \begin{pmatrix} x_0 + \int_0^t u(s)\cos\left(\int_0^s r(\tau)d\tau + \Psi_0\right) - v(s)\sin\left(\int_0^s r(\tau)d\tau + \Psi_0\right)ds \\ y_0 + \int_0^t u(s)\sin\left(\int_0^s r(\tau)d\tau + \Psi_0\right) + v(s)\cos\left(\int_0^s r(\tau)d\tau + \Psi_0\right)ds \\ \Psi_0 + \int_0^t r(s)ds. \end{pmatrix}.$$

Then, direct calculations show that

$$\varphi_u(\Gamma(A,\vec{x}_0)) = \varphi_u\left(\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\end{pmatrix}\begin{pmatrix}\cos(\Delta\Psi)&-\sin(\Delta\Psi)&0&\Delta x\\\sin(\Delta\Psi)&-\cos(\Delta\Psi)&0&\Delta y\\0&0&1&\Delta\Psi\\0&0&0&1\end{pmatrix}\begin{pmatrix}x_0\\y_0\\\Psi_0\\1\end{pmatrix},t\right)$$

$$= \varphi_u\left(\begin{pmatrix}x_0\cos(\Delta\Psi)-y_0\sin(\Delta\Psi)+\Delta x\\x_0\sin(\Delta\Psi)+y_0\cos(\Delta\Psi)+\Delta y\\\Psi_0+\Delta\Psi\end{pmatrix},t\right)$$

$$= \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\end{pmatrix}\begin{pmatrix}\cos(\Delta\Psi)&-\sin(\Delta\Psi)&0&\Delta x\\\sin(\Delta\Psi)&-\cos(\Delta\Psi)&0&\Delta y\\0&0&1&\Delta\Psi\\0&0&0&1\end{pmatrix}\begin{pmatrix}\varphi_u(\vec{x}_0,t)\\1\end{pmatrix}$$

$$= \Gamma(A,\varphi_u(\vec{x}_0,t))$$

3.3. Trim primitives for rigid bodies moving without restrictions in a plane

According to Definition 3.3, controls have to be constant along trim primitives. For the kinematic ship model, two types of solutions are possible: moving on a straight line with no angular velocity or moving on a circle with an angular velocity equal to the magnitude of the ship's velocity divided by the radius of the circle.

In these cases, we can analytically compute the flow.

For $r\equiv 0$ and u, v constant, we have

$$\begin{aligned}x(t) &= x_0 + ut\cos(\Psi_0) - vt\sin(\Psi_0)\\ y(t) &= y_0 + ut\sin(\Psi_0) + vt\cos(\Psi_0)\\ \Psi(t) &= \Psi_0,\end{aligned} \tag{9}$$

which means indeed following a straight line defined by the initial heading Ψ_0 and the constant controls (u, v).

Now assume $r\neq 0$, but (u, v, r) constant. Then, the motion is defined by

$$x(t) = x_0 + \frac{u}{r}(\sin(rt+\Psi_0) - \sin(\Psi_0)) + \frac{v}{r}(\cos(rt+\Psi_0) - \cos(\Psi_0)) \tag{10}$$

$$y(t) = y_0 - \frac{u}{r}(\cos(rt+\Psi_0) - \cos(\Psi_0)) + \frac{v}{r}(\sin(rt+\Psi_0) - \sin(\Psi_0)) \tag{11}$$

$$\Psi(t) = \Psi_0 + r\cdot t \tag{12}$$

Trim primitives are generated by elements of the Lie algebra that corresponds to the symmetry group. The kinematic ship model's symmetry group G (cf. (8)) is a subgroup of the special Euclidean group,

$$SE(3) = \left\{A\in\mathbb{R}^{4\times 4} | A = \begin{pmatrix}R&t\\0&1\end{pmatrix} : RR^T = Id, \det(R) = 1, t\in\mathbb{R}^3\right\},$$

that combines rotation and translation in three dimensions. With Definitions 1.3 and 1.4 from [40] it is easy to see, that G is a closed subgroup of SE(3), thus a Lie group itself. The corresponding Lie algebra (cf. e.g [41].), denoted by se(3), is given by $\mathrm{se}(3) = \mathrm{so}(3)\times\mathbb{R}^3$ and can be interpreted as being comprised of the three rotational and three translational velocities. The Lie algebra of rotational matrices is the group of skew symmetric matrices, so se(3) can be written as

$$\begin{pmatrix} 0 & -\omega_3 & \omega_2 & \rho_1 \\ \omega_3 & 0 & -\omega_1 & \rho_2 \\ -\omega_2 & \omega_1 & 0 & \rho_3 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

with parameters $\omega_1, \omega_2, \omega_3, \rho_1, \rho_2, \rho_3$. Thus, for the kinematic ship model, the Lie algebra is a subalgebra of se(3) with only three degrees of freedom

$$g := \left\{ \Lambda \in \mathbb{R}^{4\times 4} : \Lambda = \begin{pmatrix} 0 & -\omega & 0 & \xi \\ \omega & 0 & 0 & \zeta \\ 0 & 0 & 0 & \omega \\ 0 & 0 & 0 & 0 \end{pmatrix}, \xi \in \mathbb{R}, \zeta \in \mathbb{R}, \ \in \mathbb{R} \right\} \tag{13}$$

The exponential map of $\lambda \in g$ can then, for instance, be computed via the Rodriguez formula (see [41]).

Proposition 3.5. *Trim primitives of the kinematic ship model are given by*

(a) *straight lines, if control values and Lie algebra elements satisfy*

$$\xi = u\cos(\Psi_0) - v\sin(\Psi_0)$$

$$\zeta = u\sin(\Psi_0) + v\cos(\Psi_0),$$

or

(b) *circular arcs, if*

$$\begin{aligned} \omega &= r, \\ \zeta &= -rx_0 + u\sin(\Psi_0) + v\cos(\Psi_0), \\ \xi &= ry_0 + u\cos(\Psi_0) - v\sin(\Psi_0). \end{aligned} \tag{14}$$

Proof. Case (a), 'Straight lines': Here, $\omega = 0$ in Equation (13). On the one hand, a suitable Lie algebra element $(\xi, \zeta, 0)$ can be parametrized by t and mapped under the exponential map, such that it acts on a vector $\vec{x}_0$ by

$$\Gamma(\exp \Lambda t, \vec{x}_0) = \begin{pmatrix} 1 & 0 & 0 & \xi t \\ 0 & 1 & 0 & \zeta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ \Psi_0 \\ 1 \end{pmatrix} = \begin{pmatrix} x_0 + \xi t \\ y_0 + \zeta t \\ \Psi_0 \\ 1 \end{pmatrix}.$$

On the other hand, from Equation (9) we know that solutions on straight lines have to be of the form

$$x(t) = x_0 + ut\cos(\Psi_0) - vt\sin(\Psi_0)$$

$$y(t) = y_0 + ut\sin(\Psi_0) + vt\cos(\Psi_0)$$

$$\Psi(t) = \Psi_0.$$

Component-wise comparison leads to the following conditions for a trim primitive

$$\xi = u\cos(\Psi_0) - v\sin(\Psi_0)$$
$$\zeta = u\sin(\Psi_0) + v\cos(\Psi_0).$$

Case (b), 'Circular arcs': Again, we have to show that $\Gamma(\exp \Lambda t, \vec{x}_0)$ with a Lie algebra element chosen according to Equation (14) generates valid solutions of the dynamics. By Rodriguez formula, we obtain

$$\exp(\Lambda t) = \begin{pmatrix} \cos(\omega t) & -\sin(\omega t) & 0 & \frac{\xi}{\omega}\sin(\omega t) - \frac{\zeta}{\omega}(1-\cos(\omega t)) \\ \sin(\omega t) & \cos(\omega t) & 0 & \frac{\xi}{\omega}(1-\cos(\omega t) + \frac{\zeta}{\omega}\sin(\omega t) \\ 0 & 0 & 1 & \omega t \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{15}$$

Following the definition of trims, we compute

$$\begin{aligned} \Gamma_{\exp(\Lambda t)}(\vec{x}_0) &= \begin{pmatrix} \cos(\omega t)\cdot x_0 - \sin(\omega t)\cdot y_0 + \frac{\xi}{\omega}\sin(\omega t) - \frac{\zeta}{\omega}(1-\cos(\omega t)) \\ \sin(\omega t)\cdot x_0 + \cos(\omega t)\cdot y_0 + \frac{\xi}{\omega}(1-\cos(\omega t)) + \frac{\zeta}{\omega}\sin(\omega t) \\ \psi_0 + \omega t \end{pmatrix} \\ &= \begin{pmatrix} \sin(\omega t)\cdot(-y_0 + \frac{\xi}{\omega}) + \cos(\omega t)\cdot(x_0 + \frac{\zeta}{\omega}) - \frac{\zeta}{\omega} \\ \sin(\omega t)\cdot(x_0 + \frac{\zeta}{\omega}) + \cos(\omega t)\cdot(y_0 - \frac{\xi}{\omega}) + \frac{\xi}{\omega} \\ \psi_0 + \omega t \end{pmatrix} \end{aligned} \tag{16}$$

Now, comparing the third component of Equation (16) to the solution for $\Psi(t)$, we see that $\omega = r$ must hold. Rewriting the first two components of the solution given in Equation (10) using trigonometric identities, we see that these are equivalent to Equation (16), if the following conditions to the parameters hold

$$\begin{aligned} \omega &= r, \\ \zeta &= -rx_0 + u\sin(\Psi_0) + v\cos(\Psi_0), \\ \xi &= ry_0 + u\cos(\Psi_0) - v\sin(\Psi_0). \end{aligned} \tag{17}$$

Thus, we have found the defining conditions of a trim.

The conditions derived in the proposition relate the Lie algebra elements to the control values and, moreover, this relation depends on the initial point $(x_0, y_0, \Psi_0)^T$. However, given the initial point and control values (r, u, v), the corresponding Lie algebra element that generates a trim is uniquely defined. As a consequence of Prop. 3.5, we see that any triple of constant controls generates a trim. This is important for designing the manoeuvre automaton.

3.4. Numerical computation of trims and connecting manoeuvres

We have not been able to take over the analytical computation of trim primitives via Lie group symmetry action to the full ship dynamics. However, trim primitives of the simplified ship model give crucial intuition about how trims for the full model may look like. With that intuition, it is possible to numerically approximate the trim primitives. To this aim, we have to find the corresponding controls for a given velocity. There are multiple ways of solving this task. Forward integration over a long time horizon does the trick as the forces will balance over time independent of the initial state. Alternatively, velocity-control-pairs could be obtained by an optimal control problem.

For the computation of trim-connecting manoeuvres, we formulated and solved optimal control problems. Note that these problems are typically much less complicated than the full original optimal control problem, as we solely have to connect two kinematics states of the ship without considering any geometric constraints. Thus, these kinematics states are chosen to lie in the previously computed trim primitives. Within the optimal control problem, corresponding boundary conditions are considered. Objective functions can be chosen in correspondence with the cost function of the A* algorithm (cf. Section 4) and, if applicable, consistent with the objective functions used within the post-optimization (cf. Section 4.4).

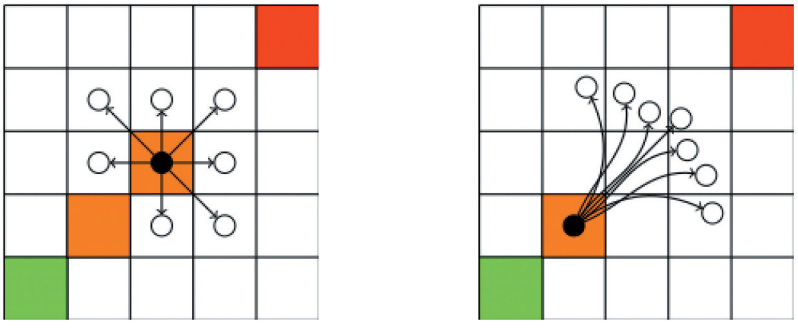
4. Trajectory planning algorithm

Our approach to trajectory planning is fundamentally based on motion primitives. Among them, trim primitives play a special role due to their properties derived in Section 3. In the following, we first describe the general procedure of our trajectory planning algorithm before we give more information on the application to our specific problem.

As a reminder, the goal of the algorithm is to compute feasible trajectories and fill the gap between optimal control problems, which heavily rely on sophisticated initial guesses close to the optimal trajectory, and path planning algorithms, which work great with a control algorithm, but provide too little information to be directly used as an initial guess. This is precisely where motion primitives come in handy.

Motion primitives can be computed offline before planning a trajectory. Because of their symmetry properties, these precomputed trajectory snippets can also be smoothly glued together to form a smooth complete trajectory. Hence, the main problem is to identify a sequence of motion primitives that form a smooth path from the initial starting point to the sought final location.

In order to identify this sequence, we use a variation of the A* search algorithm. An A* search algorithm works on a graph data structure and finds the shortest path from one node to another. The standard A* algorithm is complete and optimal. From a given current node, the A* algorithm explores all neighbouring nodes and adds them to a list of visited nodes. In standard A* implementations, such a neighbour is typically a discrete grid position right next to the current one (see Figure 2(a)). This is where the motion primitives come into play in our case. We alter the definition of a neighbour from the strict grid structure to a state that can be reached within the



(a) regular A* approach (b) hybrid A* approach

Figure 2. Representation of different variations of the A* algorithm.

execution of one motion primitive (see Figure 2(b)) and satisfy a few feasibility conditions as detailed out below. This has the consequence that we still benefit from the excellent properties on the A* algorithm but planning trajectories instead of simple paths. Thus, we can use them far more effectively as an initial guess for a solver for NLP (Nonlinear Programming) problems. Generally, the adaptation of the A* algorithm to incorporate neighbours based on the dynamics of the system, not necessarily using motion primitives, instead of a fixed neighbourhood map is known as a hybrid A* approach, see e.g. [42].

4.1. Preparation phase

One shortcoming of many A* approaches to this setting in the literature is the search for neighbours based on the system dynamics. Short portions of a trajectory are glued together often without regard to the differential equations. This difficulty is sought to be evaded by utilizing the more formal setting of motion primitives. The underlying theory allows us to make a smart selection of trajectory snippets for the A* algorithm with provable properties.

As a prerequisite to applying the A* algorithm, a motion primitive library has to be computed, i.e. a finite number of motion primitives have to be chosen. Typically, one starts with selecting trim primitives. Every type of trim primitive, which might be of interest in control scenarios, shall be represented. An approach for generating the set of trim primitives is to define a grid on the Lie algebra of appropriate size. (Recall that Lie algebra elements typically correspond to rotational or translational velocities.)

Considering the simple ship model, (1)-(3), trajectories on trim primitives can be directly connected. Thus, it is possible to build a manoeuvre automaton with trim primitives only. For more complex dynamics, e.g. the full ship model (1) – (6), a set of manoeuvres have to be computed, which connects pairs of trim primitives. It is not mandatory nor recommended to interconnect each trim primitive with all other trim primitives because it increases the computational effort of the planning. However, since

the motion primitive automaton approximates the continuous nonlinear dynamics, larger automata have better reachability properties and allow for a better approximation of optimal solutions of the original dynamics.

Based on the definition given in [10], we define manoeuvre automata for the simple and full ship model.

We start with defining a minimal example automaton for the simple ship dynamics.

Definition 4.1 (Manoeuvre Automaton for Simple Ship Model). Given the ship model in (1) – (3), the manoeuvre automaton is defined as

$$MA_{SSD} = \{\Sigma, Q, \delta, q_0, F\}$$

- Q is the collection of trim states

ID	Type	Controls
q_1	'rest'	$(u, v, r) = (0, 0, 0)$
q_2	'straight ahead'	$(u, v, r) = (1, 0, 0)$
q_3	'clockwise turn'	$(u, v, r) = (1, 0, -1)$
q_4	'anticlockwise turn'	$(u, v, r) = (1, 0, 1)$

- $\Sigma = \{\pi_1, \ldots, \pi_{12}\}$, i.e. there are $\binom{4}{2}$ trivial manoeuvres (zero duration), since instantaneous switches between any tuple of trims are possible
- δ defines the graph structure; here we have a fully connected graph, see Figure 3
- $q_0 = Q$ is the initial state, which can be on any trim,
- $F = Q$ is the set of accepted final states, i.e. every trim is accepted as final trim.

Definition 4.2 (Manoeuvre Automaton for Full Ship Model). Given the ship model (1) – (6), the manoeuvre automaton is defined as

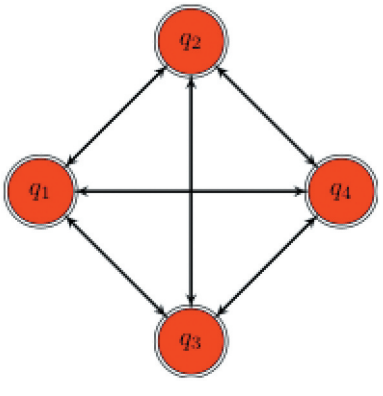


Figure 3. The state machine to the set of motion primitives in Definition 4.1.

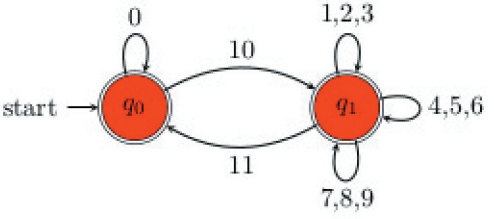


Figure 4. The state machine to the set of motion primitives in Definition 4.2.

$$MA_{FSD} = \{\Sigma, Q, \delta, q_0, F\}$$

- Q is the collection of trim states: 'resting', 'going straight with 3.0 *kn* velocity' (generated by a propeller rps of 0.602402)
- Σ the set of manoeuvres
- δ is the mapping between trims according to Figure 4,
- q_0 is the initial state 'rest'.
- $F = Q$ is the set of accepted states, i.e. every trim is accepted as final trim.

ID	Type	Description
0	trim primitive	resting state
1	trim primitive	going straight
2	manoeuvre	rotation 45 degrees counter clockwise
3	manoeuvre	rotation 45 degrees clockwise
4	manoeuvre	rotation 90 degrees counter clockwise
5	manoeuvre	rotation 90 degrees clockwise
6	manoeuvre	rotation 180 degrees clockwise
7	manoeuvre	rotation 180 degrees counter clockwise
8	manoeuvre	rotation 10 degrees clockwise
9	manoeuvre	rotation 10 degrees clockwise
10	manoeuvre	accelerating from rest
11	manoeuvre	decelerating until rest

Note that the automatons defined in Definition 4.1 and Definition 4.2 do not allow backward motion.

4.1.1. Time discretization

The manoeuvre automata from Definition 4.1 and Definition 4.2 cannot be used in the A* algorithm, yet. The last step is a time-discretization of the trims. That is, the trims are transformed into manoeuvres of fixed duration, cf [43]. However, by applying multiple discretized trim snippets, which each have a short time duration, in a row, a continuous duration can be approximated.

Once we have constructed the entire graph, we are done with the preparation phase of the algorithm. In the next sections, we introduce the other building blocks of the algorithm and see how we can put motion primitives to good use.

4.2. Building blocks of the algorithm

4.2.1. A* search algorithm

The first building block of our algorithm is the (hybrid) A* search algorithm. This algorithm is part of the informed graph-search algorithms and was discussed in the literature many times e.g. [44]. As a quick reminder, the core steps of the A* algorithm are:

Algorithm 1 (A* search algorithm).

1: Add initial state to the A* graph and initialize neighbouring nodes as unvisited
2: **while** List of unvisited nodes is non-empty **do**
3: Get cheapest node from the list of unvisited nodes and mark node as current node and visited
4: **if** current node is the target node **then return Success**
5: **else**
6: Find all unvisited neighbours of current node and update costs
7: **return error**

We do not touch the first four steps of this algorithm. However, we will alter the expansion step. Traditionally, the algorithm acts on a two-dimensional grid, and the adjacent cells on that grid become the new neighbours of the current node. Our search space, which we will introduce in the next section, is four-dimensional. This and the fact that a huge ship is limited in its movement require a better fitting definition of a neighbour cell. This is where we combine motion primitives and the A* search algorithm. To find the neighbour cells, which become nodes in the A* graph, we first execute each motion primitive from our motion primitive library. Note that, in contrast to existing methods, this is particularly inexpensive as the finite set of motion primitives can be precomputed due to the underlying symmetry conditions. However, a final check whether the execution was feasible is still mandatory. If so, the final cell, together with additional information is added to the A* graph as a node, the executed primitive is added to the graph as an edge.

4.2.2. Search space

We ended the previous chapter with the statement that the motion primitives are being executed to grow the A* graph. The A* graph is a discrete data structure, but motion primitives are continuous motion, which has been discretized with a high resolution. We need an object that acts as a link between the discrete A* graph and the continuous motions of the ship. For that purpose, we introduce the search space.

Definition 4.3 (Search space). A search space is a four-tuple $\mathcal{S}[N, A, S, G]$ where:

N	is a set of states
A	is a set of arcs connecting the states
S	is a nonempty subset of N that contains start states
G	is a nonempty subset of N that contains goal states.

In our case, a single state consists of four bits of information, a two-dimensional position, the orientation of the vehicle and a discrete dynamic state of the vehicle. Thus, the set of states N we are considering is a bounded subset of the following Cartesian product:

$$N \subset \mathbb{R}^2 \times [0, 2\pi) \times \mathbb{N}.$$

The set of arcs A is the library of motion primitives used for the exploration by the A* algorithm. In our application, S is always a one-element set, and G is a one-dimensional subspace of N where the free parameter is the orientation of the vehicle.

The search space is continuous in three dimensions. If we keep them continuous, our planning algorithm will face difficulties planning a trajectory between two discrete points in a continuous environment. A discretization of the set of states causes that we cluster near points. Experiments have shown that a spatial discretization of $15m \times 15m$ cells and a resolution of $\frac{\pi}{50}$ for the orientation are reasonable choices. We further decided to exclude geometrical constraints from the search space so that the search space is an obstacle-free representation of the environment. A geometrical constraint is any obstacle that blocks an edge to connect two states. An example of a geometrical constraint that is particularly important for the examples below is quay walls. More sophisticated constraints, which we have not considered so far, but are possible in the future, are the ship's draft and a limitation of the velocity. We are using an occupancy grid for this job, which is further discussed in the next chapter.

4.2.3. Occupancy grid

The final component we need is a binary occupancy grid that encodes for each cell in the search space, whether it is occupied, i.e. blocked by land, or free. We decided to separate the occupancy grid from the search space to allow different cell sizes. The grid cell size of the occupancy grid should be at least as large as the cell sizes in the search space. We were interested in the question of whether a lower resolution of the occupancy map increases the quality of our results. The underlying idea is that a single grid cell of the search space is smaller than the dimensions of a large ship that means if the algorithm plans a trajectory very close to the quay walls, the ship could interfere with the walls, and the planned solution is of poor quality. We can prevent this from happening by increasing the cell sizes in the occupancy grid map. However, this problem can also be overcome by using a barrier term. The barrier term is half of the diameter of the ship. When we create the occupancy grid, we block each cell that is closer to a quay wall than the barrier term. This technique allows us to keep a higher resolution but prevents unexpected behaviour of the algorithm. A second reason for the separation of the search space and the occupancy grid is that it allows us to use the same search space for multiple ship models at the same time, because ships with different lengths and widths need distinct occupancy maps.

4.3. Function principle of the algorithm

Now we are all set to look at the algorithm at runtime, which combines the presented building blocks. For that purpose, we start with the pseudo-code of the algorithm, followed by a detailed explanation of each step.

Algorithm 2 (Motion planning with primitives algorithm).

1: Add initial state to the A* graph and initialize neighbouring nodes as unvisited
2: **while** List of unvisited nodes is non-empty **do**
3: Get cheapest node from the List of unvisited nodes and mark as current node and visited
4: **if** current node is the final node **then return Success**
5: **for each** feasible motion primitive **do**
6: Execute motion primitive
7: **if** Motion primitive execution was collision-free **then**
8: Add the final state of the motion primitive execution to the list of unvisited nodes if not already added
9: Add motion primitive as an edge to the A* graph
10: **else continue**
11: **return error**

The similarity to the pseudo-code of the A* search algorithm (see Algorithm 1) is not surprising as we consider this algorithm to be an augmented version of the classic A*. Both algorithms begin with the initialization. The initial state, consisting of position orientation and a dynamic state of the ship in the form of a trim primitive, becomes the root node of the A* graph and the first element in the list of unvisited nodes. We then enter a while-loop, which begins with the extraction of the node with the cheapest combined cost. The combined cost consists of the energy spent to reach the current node, denoted as actual cost, and the Euclidean distance to the final position, as a heuristic underestimation for the remaining future costs. In our implementation, the final orientation is unrestricted. If the cheapest node is also the final node, then the optimality property of the A* ensures that we can return successfully. For most nodes, this test fails, and we continue with the loop. The next step is to explore the search space and search for the neighbours of the currently cheapest cell. For that purpose, we enter a second loop and iterate over each feasible motion primitive. We obtain the set of feasible motion primitives by mapping δ from Definition 4.1 or Definition 4.2. The execution takes place in the search space, where we begin at the current node, transcribed into a cell, and track movement of the ship. We map the current location onto the occupancy grid to monitor collisions. In the case of successful execution, the A* graph and list of unvisited nodes grow. The outer loop gets repeatedly invoked until one of two things happen. First, the list of unvisited nodes is empty, which means the algorithm was unable to find a solution to the given problem. By the completeness property of the A* algorithm, it is clear that there is no possible solution. Second, in order to assure that the programdoes not crash due to insufficient memory, if the overall number of nodes reaches a maximal value the program is stopped.

For the numerical examples in the following section, a D*-software (i.e. a *dynamic* A*) implementation for planning with primitives provided by Marin Kobilarov, Autonomous Systems, Control and Optimization (ASCO) Laboratory, Johns Hopkins University, available at https://github.com/jhu-asco/dsl was adapted.

4.4. Post-optimization

As shown in detail in the construction process in Sections 3 and 4, a resulting sequence of motion primitives is admissible to the (nonlinear) ship model and the given constraints as encoded in the occupancy grid. Moreover, A* provides an optimal sequence with respect to the objective function used within the planning. However, the search space of A* is only an approximation of the ship model state space. For that reason, we have not found an optimal solution to the following *optimal control problem (OCP)*, yet,

$$\min_{\vec{x},\vec{u},T} J(\vec{x},\vec{u}) = \int_{t_0}^{T} \ell(\vec{x}(t),\vec{u}(t))dt + P(\vec{x}(T))$$

$$\text{w.r.t. } \dot{\vec{x}}(t) = f(\vec{x}(t),\vec{u}(t)) \quad t \in [t_0,T] \,(\text{with dynamics given by Eqs.}(1)-(6)),$$

$$\vec{x}(t) \in \mathbb{X} \quad t \in [t_0,T], (\mathbb{X} \text{ is the free space encoded in the occup.grid}),$$

$$\vec{u}(t) \in \mathbb{U}(\text{defines control constraints}),$$

$$\vec{x}(t_0) \in S(\text{the set of start states, cf.Definition 4.3}).$$

For the cost functional $J(\vec{x},\vec{u})$, the following criteria can be considered, for instance,

$$J(\vec{x},\vec{u},T) = \int_{t_0}^{T} \left[w_1||\vec{u}(t)||^2 + w_2 \cdot 1\right]dt + w_3 \left\| \begin{pmatrix} x(T) \\ y(T) \end{pmatrix} - G \right\|^2,$$

which corresponds to 1) control effort, 2) duration,[2] and 3) goal state stabilization with weights $w_{1,2,3}$ and G used to denote a single goal state in Definition 4.3.

In order to solve the OCP, we employ a direct transcription method: Based on a time-discretization (not to be mixed up with the state-space discretizations used before) for the trajectory $\vec{x}$ and control curve $\vec{u}$, we transform the OCP into a *nonlinear constrained optimization problem (NLP)* of typically high, but finite dimension. For details on this standard approach in optimal control, we confer to the textbook [45], for instance. The resulting NLP is typically high-dimensional but possess sparsity-structure in the Jacobians [9],, and should therefore be addressed by sparsity-exploiting NLP solvers such as WORHP [7] and a corresponding variant for optimal control problems called TransWORHP [9]. However, all local NLP solvers have in common that they require initial guesses and, moreover, performance and successful termination highly depend on sophisticated choices of initial guesses. Thus, we use the NLP solver WORHP for solving the OCP based on the motion primitive sequences as initial guesses. This idea of how to implement the OCP for motion primitive post-optimization is described in detail in [11,12]. In these works, the sequence was an admissible solution to the OCP. Due to the occupancy grid discretization,

this only holds up to grid cell coarseness when using Algorithm 2. Note that using an already sophisticated feasible initial guess the occupancy-grid constraints do not pose as big a challenge for the NLP solver as it would without the prior A* step.

Besides faster and more robust convergence of the optimization, the quality of the resulting optimum is also important. While steered convergence to preferred local optima is quite challenging and a topic of current research, providing motion primitive sequences that are admissible to the OCP makes it likely to find an optimal solution in the vicinity, i.e. with qualitatively similar dynamic behaviour. In the following chapter, we present several scenarios for ship manoeuvring which strongly benefit from the two-step approach and exactly reflect the expected behaviour of the post-optimization as a local correction and optimization technique.

5. Numerical results

We will now illustrate the performance of the presented algorithm. As a preliminary step, we apply the A* algorithm to the kinematic ship model, before we show the whole methodology for a real scenario at the port of Rostock. Note that directly optimizing the OCP without first gaining an initial A* solution is not feasible as the optimization methods either take too long to reach an optimal solution or outright fail to converge.

5.1. A-motion planning with primitives*

Our first numerical example illustrates the concatenation of motion primitives. We consider the kinematic ship model introduced in (1)-(3) with trim primitives computed as detailed out in Section 3 and with the manoeuvre automaton defined in Definition 4.1. An example scenario is created as depicted in Figure 5. The ship is required to perform a turn manoeuvre, which cannot be done as turning on the spot due to its minimal

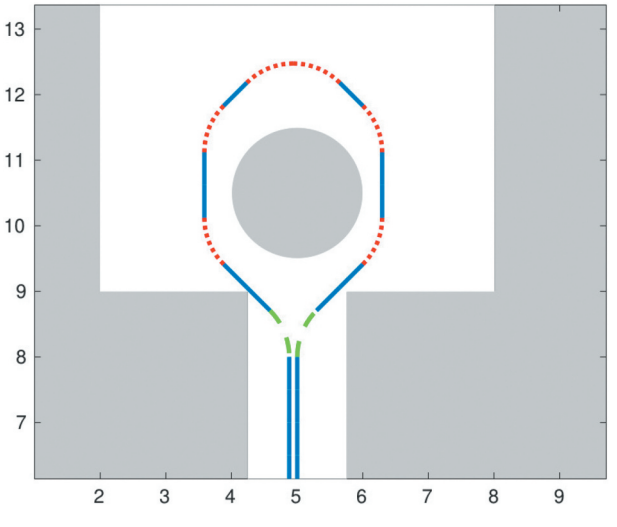


Figure 5. Motion primitive solution for a turning manoeuvre for the kinematic ship model. Different colours/lines represent different motion primitives.

Figure 6. Map of the port of Rostock which is used as a testing environment. Map data copyrighted OpenStreetMap contributors and available from https://www.openstreetmap.org, see [46].

turning radius. Start point and desired final point coincide at $(x, y) = (5.0, 6.0)$. The hybrid A* algorithm computes the optimal sequence of motion primitives that gives a solution to the turning task. For illustration, we draw the different types of trim primitives in blue (straight ahead), yellow (right turn), and purple (left turn) in Figure 5. Note that the trim primitives are stored with a fixed duration, since the A* algorithm requires a full discretization of the continuous-time problem. However, the same trim can be concatenated several times for longer straight or turning phases. The sequence of motion primitives is an approximation to the optimal solution of the original problem with full nonlinear dynamics. A larger library with trims of different turning radii, various speed, and shorter durations would lead to better results – at the cost of high computational effort. Alternatively, we propose to perform a post-processing step via an optimal control problem as discussed in Section 4.3.

5.2. Autonomous ship navigation in the port of Rostock

In this section, we apply the algorithm to large ships in the port of Rostock, see Figure 6. We show in three examples that the motion primitive algorithm enables the post-optimization to find optimal trajectories in a narrow environment. In all three cases the automaton given by Definition 4.2 was used.

In the first experiment, we start with a long trajectory, see Figure 7 with parameters given in Table 1. The starting point lies outside of the port in the Baltic sea and the final position roughly 12 km away in the city port of Rostock. One can notice that the algorithm did not take many explorations to connect the starting point with the end point. Most of the visited positions are close to the start or to the end point, which is a consequence of the geometry of the port and the fact that a specific cell must be reached. The cluster around

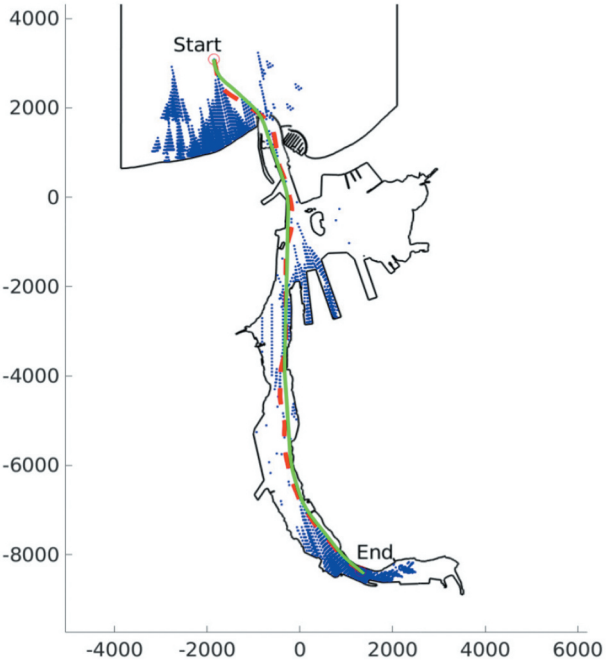


Figure 7. Computed ship manoeuvre into the port of Rostock. Sequence of motion primitives with repositioning on the search space grid is given in red (dashed), post-optimized trajectory is given in green (solid). Visited states are depicted in blue. Distances given in Metres.

Table 1. Parameters for first example.

Parameter	Value
Starting state (lat, long, degree)	54.198314, 12.072334, 170.0
final position (lat, long)	54.094846, 12.121948
discretization of the search space	15 m $\times$ 15 m $\times \frac{\pi}{50}$
Motion primitive computation time	1,301,236 sec.
A* graph	5939 vertices, 7075 edges
Number of motion primitive segments in solution	31

the starting point is due to the narrow entry of the port, so the algorithm explores this area in more detail. One observes cone-shaped objects on both sides of the starting point, which we assume to be caused by backward search within in the algorithm (cf. Section 4).

The next cluster is at the two berths, roughly at $(1000m, -3000m)$. Here, the algorithm gets stuck for a few explorations steps as these dead ends are on the connecting line between the harbour entry to the end point. Therefore, these positions have a small cost associated with. It takes the algorithm only few explorations to make good progress in the narrow canal starting at $(0m, -2000m)$ to $(0m, -8000m)$. The final cluster is around the final position, which is not surprising since the algorithm has to reach a very small cell $(225m^2)$. As a comparison the footprint of our ship, the ferry Mecklenburg-Vorpommern, is approx. $5800m^2$. The final motion primitive solution is indicated with the green dashed line, after post-optimization we obtain the solid green line.

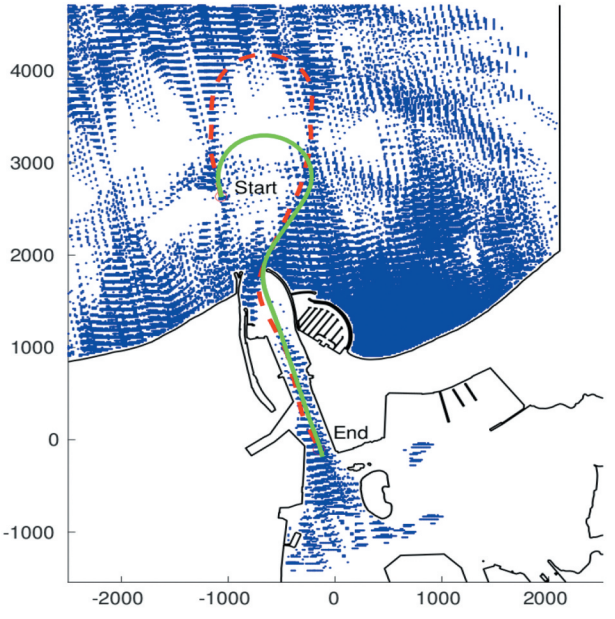


Figure 8. Resulting sequence of motion primitives (dashed red line) and post-optimized solution (solid green line) for a scenario, in which the ship has to turn before entering the port. Distances given in Metres.

Table 2. Parameters for second example.

Parameter	Value
Starting state (lat, long, degree)	54.194175, 12.084501, 350.0
final position (lat, long)	54.168907, 12.098929
discretization of the search space	15 m $\times$ 15 m $\times \frac{\pi}{50}$
Motion primitive computation time	16,243,933 sec.
A* graph	63,166 vertices, 77,825 edges
Number of motion primitive segments in solution	20

Recall that the ship has limited agility. In particular, it has a minimal turning radius and cannot turn on the spot. Similar to our preliminary example in Figure 5, in the next scenario, the ship has to perform a turning manoeuvre before entering the port. As one can see in Figure 8 (with parameters in Table 2), this starting condition leads to many more visited states by the A* algorithm. Due to the narrow entrance of the port, the resulting optimal sequence of primitives (depicted by the dashed red line) performs a large turning circle. The large detour is worsen due to the small-sized manoeuvre automaton (cf. Definition 4.2). To obtain a better manoeuvre, we perform the post-processing step as proposed in Section 4.4. The result is depicted by the solid green line. One can observe that the optimized trajectory is able to minimize the detour while still finding a feasible entering path into the port.

In our last example, the ship has to perform a 180°-turn starting directly in the narrow port entrance. For this scenario, the exploration nature of the A* algorithm is particularly crucial in order to find feasible solutions. In Figure 9 (corresponding parameters in Table 3), the resulting motion primitive solution is depicted by a dashed red line again. It performs

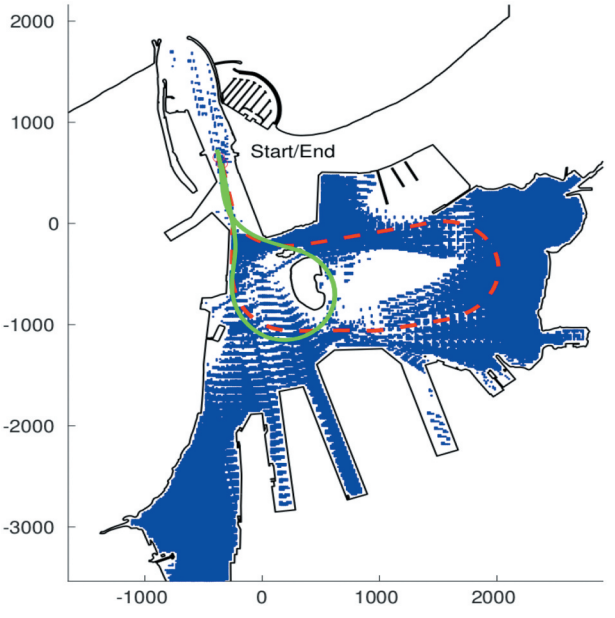


Figure 9. Motion primitive solution (dashed red line), post-optimization (green line) and explored states (blue dots). Distances given in Metres.

Table 3. Parameters for third example.

Parameter	Value
Starting state (lat, long, degree)	54.175922, 12.095341, 170.0
final position (lat, long)	54.177127, 12.094783
discretization of the search space	15 m $\times$ 15 m $\times \frac{\pi}{50}$
Motion primitive computation time	20,99,941 sec.
A* graph	62,392 vertices, 84,444 edges
Number of motion primitive segments in solution	22

the turning within the inner port, obeying the obstacles, i.e. Island. Again, we observe that the turning phase has a big detour due to the limited amount of primitives in the library. However, as in the previous example, the optimal solution that is obtained from the post-processing step (solid green line) minimized the detour. Note that the optimal solution seems to violate the state constraints given by the Island in the inner port. When checking the TransWORHP output, one sees that the constraints are fulfilled at each discretization point. To obtain obstacle avoidance also at intermediate time points, a finer time discretization has to be used. This is not within the scope of our numerical analysis, though.

6. Conclusion and future work

This article proposes a methodology for autonomous navigation of ships based on combined mathematical techniques:

(1) Modelling of ship dynamics and identifying structural properties (i.e. symmetry),

(2) Quantization of the dynamics to a finite set of motion primitives (i.e. trim primitives and manoeuvres as in [10]),
(3) Modelling of environmental constraints (i.e. the port layout) via occupancy maps,
(4) Graph-based planning via the A* algorithm on motion primitives, and
(5) Post-optimization by solving optimal control problems using a direct method.

While, individually, these methods are well established, e.g. A* algorithms in computer science or direct optimal control with nonlinear optimization in numerical mathematics, their combination and application to specific tasks in autonomous control is less well investigated. Thus, the presented case study of ship navigation via graph-based motion planning provides a helpful validation of the approach. The second and third scenarios in Section 5 show the advantages if planning and nonlinear optimization are combined: the A* algorithm performs a global search but provides a primitive sequence, which is only suboptimal w.r.t. the optimization criteria since it does not use the full dynamics. Contrarily, the direct optimal control method considers the nonlinear ship model, but requires a good initial guess for the nonlinear optimizer to converge to a globally efficient solution. As pointed out in the introduction, the navigation of ships in comparison to street or aeronautical vehicles has particular challenges due to limited agility of large ships and narrow, possibly congested space in a port. Since the agility and dynamics of ships are limited compared to those of road vehicles, for instance, computation times for real-world scenarios are less of a problem. Thus, autonomous navigation of ship in ports seem to be realistic in near future. Moreover, current research in *GALILEOnautic* [4,5,6] addresses the interaction of several ships.

Future work shall address robustness analyses of the computed solutions w.r.t. uncertainty in states and model parameters, as well as the design of a feedback control. In [38], for instance, model predictive control is designed using a motion primitive library and stability of this control scheme is shown for an academic robot example.

Notes

1. Since this article focuses on a proof-of-concept for the proposed method and does not include a real-world application, we omit a discussion whether a future version of this method would technically be considered as an assisted, an automated or an autonomous navigation system.
2. Note that $\int_{t_0}^{T} w_2 \cdot 1 dt = w_2 \cdot (T - t_0)$, i.e. the duration of the solution.

Acknowledgments

BJ and KF acknowledge funding by the Deutsche Forschungsgemeinschaft (German Research Foundation) within the Priority Program SPP 1835 'Cooperative Interacting Automobiles' (grant number: FL 989/3-1).

Disclosure statement

No potential conflict of interest was reported by the author(s).

ORCID

Benjamin Jurgelucks http://orcid.org/0000-0002-7516-0944
Sylvain Roy http://orcid.org/0000-0001-5297-6396
Christof Büskens http://orcid.org/0000-0001-7385-4670
Kathrin Flaßkamp http://orcid.org/0000-0001-5983-1907

References

[1] U.N.C. on Trade and D. (UNCTAD), *Review of maritime transport 2018*, 2021; available at https://unctad.org/webflyer/review-maritime-transport-2018. [Online, accessed 14-September-2021].
[2] OECD, *Ocean shipping and shipbuilding*, 2021; available at https://www.oecd.org/ocean/topics/ocean-shipping/. [Online, accessed 14-September-2021].
[3] ICS, *Shipping and world trade*, 2020; available at https://www.ics-shipping.org/shipping-facts/shipping-and-world-trade. [Online, accessed 23-July-2020].
[4] S. Roy, H. Wernsing, and C. Büskens, *Optimization of ship manoeuvring within the project galileonautic*, PAMM 17 (2017), pp. 813–814. Available at https://onlinelibrary.wiley.com/doi/abs/10.1002/pamm.201710374
[5] R. Zweigel, J.J. Gehrt, S. Liu, S. Roy, C. Büskens, M. Kurowski, T. Jeinsch, A. Schubert, M. Gluch, O. Simanski, E. Pairet-Garcia, F. Siemer, and D. Abel, *Optimal maneuvering and control of cooperative vehicles as case study for maritime applications within harbors*, in *2019 18th European Control Conference (ECC)*, Naples, Italy, 2019, pp. 3022–3027.
[6] M. Kurowski, S. Roy, J. Gehrt, R. Damerius, C. Büskens, D. Abel, and T. Jeinsch, *Multi-vehicle guidance, navigation and control towards autonomous ship maneuvering in confined waters*, in *2019 18th European Control Conference (ECC)*, Naples, Italy, 2019, pp. 2559–2564.
[7] C. Büskens and D. Wassel, *The ESA NLP solver WORHP*, in *Modeling and Optimization in Space Engineering* 73 Fasano, G., and Pintér, J., Springer, 2012, pp. 85–110 doi:10.1007/978-1-4614-4469-5_4.
[8] T. Nikolayzik, C. Büskens, and M. Gerdts, *Nonlinear large-scale Optimization with WORHP*, in *AIAA/ISSMO Multidisciplinary Analysis Optim. Conf.* Fort Worth, Texas, Vol. 13 doi:10.2514/6.2010-9136, 2010.
[9] C. Büskens and M. Knauer, *From WORHP to TransWORHP*, in *Proceedings of the 5th International Conference on Astrodynamics Tools and Techniques* Noordwijk, Netherlands, May, 2012.
[10] E. Frazzoli, M.A. Dahleh, and E. Feron, *Maneuver-based motion planning for nonlinear systems with symmetries*, IEEE Trans. Rob. 21 (2005), pp. 1077–1091. doi:10.1109/TRO.2005.852260.
[11] K. Flaßkamp, S. Ober-Blöbaum, and M. Kobilarov, *Solving optimal control problems by exploiting inherent dynamical systems structures*, J. Nonlinear Sci. 22(2012), pp. 599–629. doi:10.1007/s00332-012-9140-7.
[12] K. Flaßkamp, *On the optimal control of mechanical systems – Hybrid control strategies and hybrid dynamics*, Ph.D. diss., University of Paderborn, 2013.
[13] K. Flaßkamp, J. Timmermann, S. Ober-Blöbaum, and A. Trächtler, *Control strategies on stable manifolds for energy-efficient swing-ups of double pendula*, Int. J. Control (2014). doi:10.1080/00207179.2014.893450.
[14] L.E. Dubins, *On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents, American*, J. Math. 79 (1957), pp. 497–516. doi:10.2307/2372560.
[15] J. Reeds and L. Shepp, *Optimal paths for a car that goes both forwards and backwards, Pacific*, J. Math. 145(1990), pp. 367–393
[16] S.M. LaValle, *Planning Algorithms*, Cambridge: Cambridge university press. doi:10.1017/CBO9780511546877, 2006.

[17] E. Frazzoli and F. Bullo, *On quantization and optimal control of dynamical systems with symmetries*, in *Proceedings of the 41st IEEE Conference on Decision and Control*, Las Vegas, NV, USA, 2002, pp. 817–823. doi:10.1109/CDC.2002.1184606.
[18] E. Frazzoli, *Robust hybrid control for autonomous vehicle motion planning*, Ph.D. diss., Massachusetts Institute of Technology, 2001.
[19] B. Paden, M. Čáp, S.Z. Yong, D. Yershov, and E. Frazzoli, *A survey of motion planning and control techniques for self-driving urban vehicles*, IEEE Trans. Intell. Veh. 1 (2016), pp. 33–55. doi:10.1109/TIV.2016.2578706.
[20] H.J. Sussmann and J.C. Willems, *300 years of optimal control: From the Brachystochrone to the maximum principle*, IEEE Control Syst. Mag. 17 (1997), pp. 32–44. doi:10.1109/37.588098.
[21] T. Binder, L. Blank, H.G. Bock, R. Bulirsch, W. Dahmen, M. Diehl, T. Kronseder, W. Marquardt, J.P. Schlöder, and O. Von Stryk, *Introduction to model based optimization of chemical processes on moving horizons*, in *Online Optimization of Large Scale Systems: State of the Art*, M. Grötschel, S.O. Krumke, and J. Rambau, eds., Springer Berlin Heidelberg: Springer, 2001, pp. 295–340.
[22] A. Wächter and L.T. Biegler, *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*, Math. Program. 106 (2006), pp. 25–57. doi:10.1007/s10107-004-0559-y.
[23] R.H. Byrd, J. Nocedal, and R.A. Waltz, *Knitro: An integrated package for nonlinear optimization*, in *Large-Scale Nonlinear Optimization*, G. Di Pillo and M. Roma, eds., Springer US, Boston, MA, 2006, pp. 35–59. doi:10.1007/0-387-30065-1_4.
[24] P.E. Gill, W. Murray, and M.A. Saunders, *SNOPT: An SQP algorithm for large-scale constrained optimization*, SIAM Rev. 47 (2005), pp. 99–131. doi:10.1137/S0036144504446096.
[25] A. Lazarowska, *A trajectory base method for ship's safe path planning*, Procedia Comput. Sci. 96 (2016), pp. 1022–1031. knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 20th International Conference KES-2016. doi:10.1016/j.procs.2016.08.118.
[26] M. Schuster, M. Blaich, and J. Reuter, *Collision avoidance for vessels using a low-cost radar sensor*, in 19th IFAC World Congress, Vol. 47, 2014, pp. 9673–9678 IFAC Proceedings. doi:10.3182/20140824-6-ZA-1003.01872
[27] M.D. Pedersen and T.I. Fossen, *Marine vessel path planning & guidance using potential flow*, in 9th IFAC Conference on Manoeuvring and Control of Marine Craft, Vol. 45, 2012, pp. 188–193 . IFAC Proceedings. doi:10.3182/20120919-3-IT-2046.00032
[28] A. Häusler, A. Saccon, A. Aguiar, J. Hauser, and A. Pascoal, *Cooperative motion planning for multiple autonomous marine vehicles*, in 9th IFAC Conference on Manoeuvring and Control of Marine Craft Vol 45, 2012, pp. 244–249 . IFAC Proceedings. doi:10.3182/20120919-3-IT-2046.00042
[29] Y. Huang, L. Chen, P. Chen, R.R. Negenborn, and P. van Gelder, *Ship collision avoidance methods: State-of-the-art*, Saf Sci 121 (2020), pp. 451–473. doi:10.1016/j.ssci.2019.09.018.
[30] Y. Li and J. Zheng, *Real-time collision avoidance planning for unmanned surface vessels based on field theory*, ISA Trans. 106 (2020), pp. 233–242. doi:10.1016/j.isatra.2020.07.018.
[31] C. Tam and R. Bucknall, *Cooperative path planning algorithm for marine surface vessels*, Ocean Eng. 57 (2013), pp. 25–33. doi:10.1016/j.oceaneng.2012.09.003.
[32] K. Shojaei, *Observer-based neural adaptive formation control of autonomous surface vessels with limited torque*, Rob. Auton. Syst. 78 (2016), pp. 83–96. doi:10.1016/j.robot.2016.01.005.
[33] W. Jing, C. Liu, T. Li et al., *Path planning and navigation of oceanic autonomous sailboats and vessels: A survey*, J. Ocean Univ. China 19 (2020), pp. 609–621. doi:10.1007/s11802-020-4144-7.
[34] M.A. Hinostroza Muñoz, C. Guedes Soares, and H. Xu, *Motion planning, guidance and control system for autonomous surface vessel*, in *Proceedings of the ASME 2018 37th International Conference on Ocean, Offshore and Arctic Engineering*, Madrid, Spain. doi:10.1115/OMAE2018-78537, 2018.

[35] H.S. Stenersen, *Construction and control of an autonomous sail boat*, in 10th IFAC Conference on Control Applications in Marine SystemsCAMS 2016. Trondheim, Norway, Vol. 49, 2016, pp. 524–531. IFAC-PapersOnLine.
[36] P. Oltmann and S. Sharma, *Simulation of combined engine and rudder maneuvers using an improved model of hull-propeller-rudderinteractions*, Schriftreihe Schiffbau (1984), pp. 1–24. doi:10.15480/882.929.
[37] R. Holmberg and O. Khatib, *Development and control of a holonomic mobile robot for mobile manipulation tasks*, Int. J. Rob. Res. 19(2000), pp. 1066–1074. doi:10.1177/02783640022067977.
[38] K. Flaßkamp, S. Ober-Blöbaum, and K. Worthmann, *Symmetry and motion primitives in model predictive control*, Math. Control Signals Syst. 31 (2019), pp. 455–485. doi:10.1007/s00498-019-00246-7.
[39] J.E. Marsden and T.S. Ratiu, *Introduction to mechanics and symmetry*, in *Texts in Applied Mathematics*, 2nd ed, Vol. 17. Springer, New York, NY: Springer, 1999.
[40] B. Hall, *Lie groups, Lie Algebras, and representations: An elementary introduction*, in *Graduate Texts in Mathematics*, Cham at al: Springer, 2003.
[41] R. Murray, Z. Li, and S. Sastry, *A Mathematical Introduction to Robotic Manipulation*, Boca Raton et al.: CRC Press, 1994.
[42] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, *Practical search techniques in path planning for autonomous driving*, Ann. Arbor. 1001 (2008), pp. 18–80.
[43] M. Kobilarov, *Discrete geometric motion control of autonomous vehicles*, Ph.D. diss., University of Southern California, USA, 2008.
[44] P.E. Hart, N.J. Nilsson, and B. Raphael, *A formal basis for the heuristic determination of minimum cost paths*, in *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4 (), 1968, pp. 100–107 10.1109/TSSC.1968.300136 .
[45] M. Gerdts, *Optimal Control of ODEs and DAEs*, De Gruyter, 2021. doi:10.1515/97831102499962012.
[46] OpenStreetMap contributors, *Planet dump*, 2017. Available at: https://planet.osm.org; https://www.openstreetmap.org.