Receding Horizon Control Using Graph Search for Multi-Agent Trajectory Planning

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

15-09-2021 / 17-09-2021

CITATION

Scheffe, Patrick; Pedrosa, Matheus Vitor de Andrade; Flaßkamp, Kathrin; Alrifaaee, Bassam (2021): Receding Horizon Control Using Graph Search for Multi-Agent Trajectory Planning. TechRxiv. Preprint.
<https://doi.org/10.36227/techrxiv.16621963.v1>

DOI

[10.36227/techrxiv.16621963.v1](https://doi.org/10.36227/techrxiv.16621963.v1)

Receding Horizon Control Using Graph Search for Multi-Agent Trajectory Planning

Patrick Scheffe, Matheus V. A. Pedrosa, Kathrin Flaßkamp and Bassam Alrifaae

Abstract—It is hard to find the global optimum to general nonlinear and nonconvex optimization problems in reasonable time. This paper presents a method to transfer the receding horizon control approach, where nonlinear, nonconvex optimization problems are considered, into graph-search problems. Specifically, systems with symmetries are considered to transfer system dynamics into a finite state automaton. In contrast to traditional graph-search approaches where the search continues until the goal vertex is found, the transfer of a receding horizon control approach to graph-search problems presented in this paper allows to solve them in real-time. We prove that the solutions are recursively feasible by restricting the graph search to end in accepting states of the underlying finite state automaton. The approach is applied to trajectory planning for multiple networked and autonomous vehicles. We evaluate its effectiveness in simulation as well as in experiments in the Cyber-Physical Mobility Lab, an open source platform for networked and autonomous vehicles. We show real-time capable trajectory planning with collision avoidance in experiments on off-the-shelf hardware and code in MATLAB for two vehicles.

SUPPLEMENTARY MATERIAL

A demonstration video of this work is available online [1]. The code is available at GitHub [2] and ready-to-run at Code Ocean [3].

I. INTRODUCTION

Solving nonlinear and nonconvex optimization problems under temporal constraints is hard, if not intractable. There exist several approaches to find good solution candidates for a nonconvex, nonlinear optimization problem. The variety of approaches, which are too many to review in detail, have in common that they simplify the original problem at some point or another within the optimization algorithm as a compromise between global optimality and computational efficiency [4]. In the following, we will focus on approaches that simplify the original optimization problem into a discretized version, i.e. the decision variable is restricted to a finite set of values.

One field of application which deals with nonconvex, nonlinear optimization is trajectory planning for mobile robots. Trajectory planning for mobile robots has been a topic of research for several decades. While the planning problem for

a single robot with linear dynamics in a static environment is considered as solved, there exist open questions when planning trajectories for multiple robots with nonlinear dynamics that interact in changing environments [5]. The problem with oversimplifying the model is that the trajectory following controller on the real system might not be able to follow the plan generated by the trajectory planner. In the worst case, even though the plan is feasible, due to the modeling error it might drive the system into an undesired state such as a collision. If trajectories are planned based on a detailed model, the robotic system is more likely to be able to follow the resulting plan. Using detailed models comes at the cost of increased computation time. An abstraction into a discretized system representation as it is proposed in this work can circumvent this problem.

The next paragraphs present this paper's contribution, the state of the art and this paper's structure.

A. Contribution

This paper presents the combination of a heuristic search on a finite state automaton (FSA) with a receding horizon framework known from model predictive control (MPC). We analyze the resulting method, which we term receding horizon graph search (RHGS), with tools from receding horizon control to show recursive feasibility. We evaluate the method by applying it to the problem of trajectory planning for multiple car-like robots. The trajectories are planned using a high fidelity dynamic model. The analytically determined steady-state conditions and the transitions between them are translated into a motion primitive automaton (MPA). Typically, graph search is used to find a path between a start and an end point. In contrast, we plan trajectories that aim to follow a reference trajectory while considering system dynamics and satisfying input and collision constraints. The effectiveness of the approach is shown in experiments in the Cyber-Physical Mobility Lab (CPM Lab) Lab [6].

B. Related Work

In the literature, most methods for solving the trajectory planning problem can be categorized into either methods that are based on continuous optimization problem formulations, such as MPC, or methods that are based on discretized problem formulations, such as an A* search algorithm or the rapidly-exploring random tree approach [5]. In the following, we first present works that use an optimization problem formulation based on motion primitives, and second examine works that use a graph-based formulation.

This research is supported by the Deutsche Forschungsgemeinschaft (German Research Foundation) within the Priority Program SPP 1835 "Cooperative Interacting Automobiles" (grant number: KO 1430/17-1).

P. Scheffe and B. Alrifaae are with the Chair for Embedded Software, RWTH Aachen University {scheffe, alrifaae}@embedded.rwth-aachen.de

Matheus V. A. Pedrosa and Kathrin Flaßkamp are with the Chair of Systems Modeling and Simulation, Systems Engineering, Saarland University {matheus.pedrosa, kathrin.flaskkamp}@uni-saarland.de

The idea of state and/or control quantization is pursued in research on control theory for the last few decades (e.g. [7]). One method of quantization is selecting a finite number of control and state trajectories, also known as motion primitives. Each motion primitive determines a specific behavior of the system's dynamics. Multiple motion primitives with flexible duration can be concatenated for a longer trajectory of the states of a system [8]. Such systems are classified as a mixed logical dynamical (MLD) systems [9].

The sequence and duration of motion primitives that take an MLD system from the starting point to the goal point can be found by solving a hybrid optimization problem. A method to solve a hybrid optimization problem is mixed integer programming (MIP). MIP combines both continuous and discrete variables in one optimization problem. An approach for collision avoidance of networked vehicles, where collision constraints are modeled with binary decision variables, can be found in [10]. While MIP approaches can guarantee to find optimal solutions, the required computation time is high. For a general review on trajectory planning in a MIP framework, the reader can consult [11]. The approach of modelling the system as an MLD system and solving a hybrid optimization problem has been successfully applied to an aeronautic vehicle, where the state space is typically larger than for ground vehicles [8]. In [12], the approach was applied to autonomous ground vehicles with detailed system models for agile movements. In [13], multiple vehicles are coordinated based on motion primitives. Here, the vehicles' goal is to keep a rigid formation. A hierarchy is introduced with a layer of motion coordination primitives on top of motion primitives. These coordination primitives determine the respective motion primitive for each participating vehicle in the formation. In [14], a hierarchical trajectory planner for multiple vehicles is presented. A high level planner determines a sequence of motion primitives for participating vehicles. In contrast to [13], the vehicles avoid collisions with one another by concatenating the motion primitives resulting from a hybrid optimization. In [15], multiple vehicles are controlled in a centralized and a distributed manner using motion primitives. The paper presents two formulations of the optimal control problem (OCP), one by using a convex approximation and quadratic programming and one using MIP.

Early work in the area of graph-based trajectory planning for autonomous vehicles does not consider vehicle dynamics [5]. In [16], an A^* search algorithm for trajectory planning using motion primitives is applied. The approach includes obstacle avoidance in three dimensions for unmanned aeronautic vehicles. In trajectory planning tasks performed with graph search, path validations and edge evaluations are computationally demanding [5]. The number of edge evaluations can be reduced by employing a lazy approach. In lazy approaches, the actual cost of an edge is approximated by an easy to compute lower bound on the cost. The exact cost, which is computationally expensive, is only determined when the vertex connected with the edge is considered promising. Examples that employ a lazy approach are lazy weighted A^* [17], lazy

shortest path [18] or lazy receding horizon A^* [19]. The algorithm's name in [19] is Lazy Receding Horizon A^* , which originates from two horizons of evaluating the true cost of edges and a lazy estimate of the cost. This is not to be confused with the receding horizon on the tree depth as it is proposed in the present work. Both approaches can actually be combined to further decrease computational effort. A variant of A^* termed Hybrid A^* reduces the discretization error that occurs when associating expanded vertices to cell centers of the world grid as A^* does. Instead of associating vertices to the cell centers, continuous state values are associated to the cells alongside their corresponding cost. This means that vertices are discarded if others with a better cost estimations are found, so suboptimal solutions can be returned. In [20], Hybrid A^* is used for path planning for one vehicle in static, unknown environments and validated in the 2007 DARPA Urban Challenge. Hybrid A^* is also used in [21] for trajectory planning for a single vehicle. Trajectories are planned with a simplified vehicle dynamics model for longitudinal and lateral motion. The planning horizon is limited in both time and distance traveled. Simulation results are given for a straight road segment. Limiting the depth of a graph search algorithm for real-time capability was also explored in [22]. In that work, the trajectory for one aerial vehicle is optimized in a receding horizon fashion. The vehicle moves in a static environment and uses a precomputed cost function to evaluate graph vertices during search. The approach is evaluated in simulation. In [23], the planning time available due to real time constraints is split into two parts. In the first part, an algorithm looks for an optimal trajectory from the current pose to the goal pose. If the algorithm does not terminate during the available time, a receding horizon planner is activated as a backup solution to plan ahead long enough for the vehicle to operate safely in the immediate future. In order to avoid getting stuck at local suboptimal solutions, the cost-to-go function is based on a reduced trajectory planning problem with static obstacles and a low-fidelity maneuver automaton. In [24], multiple resolutions are introduced to the state lattice on which the graph search operates. Trajectory planning therefore gives coarser results in greater distance to the vehicle, but can be accelerated compared to trajectory planning in state lattices with uniform resolution.

In [25], a Monte Carlo Tree Search on quantized control inputs generates cooperative trajectories. The approach is evaluated in simulation and multiple traffic scenarios. In [26], [27], trajectory planning based on graph search was extended to multiple vehicles. These works do not generate a motion graph for the system, but use discretized control inputs as their motion primitives without smooth transitions. The approach is evaluated in simulation on specific traffic scenarios like an intersection or two merging lanes.

C. Paper Overview

This paper is organized as follows. In section II, we review properties of systems with symmetries, introduce primitives and a FSA based on primitives. In section III, we propose

RHGS based on FSA, and show that it fulfills the recursive feasibility property. In section IV, the approach is applied to trajectory planning for car-like robots. The section presents the kinematic single-track vehicle model, the MPA, and the trajectory planning framework. In section V, the approach is evaluated in simulation and experiments.

II. MODELING WITH A FINITE STATE AUTOMATON

Consider the time-invariant nonlinear dynamical control system $\mathcal{S}$ described by the following ordinary differential equation:

$$\dot{\mathbf{x}}(t) := \frac{d}{dt}\mathbf{x}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \quad (1)$$

with initial condition $\mathbf{x}(0) = \mathbf{x}_0$, the state vector $\mathbf{x}(t) \in \mathcal{M} \subseteq \mathbb{R}^n$ and the control input $\mathbf{u}(t) \in \mathbb{R}^m$ for time $t \geq 0$, an n -dimensional manifold $\mathcal{M}$, and a vector field f , which its properties will be described further on this section.

While, in general, the solution of a nonlinear equation is not analytically tractable, the dynamics equations of control systems can be simplified by exploiting their structural properties. Specifically, we are interested in dynamical systems with invariance property such that we can quantize the state and control values.

Let $\mathcal{TM}$ be the tangent bundle of $\mathcal{M}$, the function $\varphi_u : \mathcal{M} \times \mathbb{R}^m \rightarrow \mathcal{M}$ represents the state flow and $\mathcal{L}_{\text{loc}}^\infty([0, \infty), \mathbb{R}^m)$, $m \in \mathbb{N}$, be the space of Lebesgue-measurable and locally absolutely integrable functions. We assume the mapping $f : \mathcal{M} \times \mathbb{R}^m \rightarrow \mathcal{TM}$ of system (1) as being continuous and locally Lipschitz w.r.t. the state $\mathbf{x}(t)$. Then, we guarantee the existence and uniqueness of the solution in the form

$$\mathbf{x}(\tau) = \varphi_u(\mathbf{x}_0, \tau) \quad (2)$$

on its maximal interval of existence $\tau \leq I_{\mathbf{x}_0, \mathbf{u}}$ for $\mathbf{u} \in \mathcal{L}_{\text{loc}}^\infty([0, \infty), \mathbb{R}^m)$. When the domain of the control function $\mathbf{u}$ is restricted to the compact interval $[0, T]$, then $\mathbf{u}|_{[0, T]} \in \mathcal{L}_{\text{loc}}^\infty([0, T], \mathbb{R}^m)$ and the solution uniquely exists on $I_{\mathbf{x}_0, \mathbf{u}} \cap [0, T]$.

Note that the flow $\varphi_{\mathbf{u}(t)}$ is not known analytically for nonlinear systems and general choices of $\mathbf{u}(t)$ and $\mathbf{x}_0$, but it needs to be approximated numerically. This might make model-based optimization and trajectory planning computationally challenging. However, the computational load can be reduced by exploiting the invariance of the system $\mathcal{S}$ w.r.t. a class of transformations on the state.

A. Trajectories for Systems with Symmetry

An important property of systems which can be exploited is the existence of symmetries in its dynamics. This means, when we consider a group operation due to the Lie group structure, the symmetry is a group action on the state such that the dynamics are invariant [28]. Many mechanical systems present this property, for example mobile robots, cars or helicopters [29], [8].

For the formal definition, let us first consider a Lie group $\mathcal{G}$ with identity element e and, for the system $\mathcal{S}$ described by (1), a left action of the group $\mathcal{G}$ on the state manifold $\mathcal{M}$ as

a smooth map $\Psi : \mathcal{G} \times \mathcal{M} \rightarrow \mathcal{M}$. Also, $\Psi(e, \mathbf{x}) = \mathbf{x} \forall \mathbf{x} \in \mathcal{M}$ and $\Psi(g, \Psi(h, \mathbf{x})) = \Psi(gh, \mathbf{x}) \forall g, h \in \mathcal{G}$ and $\mathbf{x} \in \mathcal{M}$. We also consider the abbreviation $\Psi_g(\mathbf{x})$ as being the operation $\Psi(g, \mathbf{x})$.

Definition 1 (Symmetry group). The triple $(\mathcal{G}, \mathcal{M}, \Psi)$ is a symmetry group for the system $\mathcal{S}$ if the property

$$\varphi_u(\Psi(g, \mathbf{x}_0)) = \Psi(g, \varphi_u(\mathbf{x}_0, t)) \quad (3)$$

holds $\forall (g, \mathbf{x}_0, t) \in \mathcal{G} \times \mathcal{M} \times \mathbb{R}_+$, $\mathbf{u} \in \mathcal{L}_{\text{loc}}^\infty([0, \infty), \mathbb{R}^m)$.

Equivalently, we say the system $\mathcal{S}$ is invariant w.r.t. the group action Ψ .

For mechanical systems, in general, the symmetry group is a subgroup of $SE(n) := T(n) \rtimes SO(n)$, where $T(n)$ is the group of translations, $SO(n)$ is the special orthogonal group and $\rtimes$ denotes the algebraic right semijoin operation. $SO(n)$ can be represented by the set $\{R \in \mathbb{R}^{n \times n} : R^T R = I, \det R = 1\}$, while one of the forms to represent $SE(n)$ is by a pair $(R, \Delta \mathbf{x})$, with $R \in SO(n)$ and $\Delta \mathbf{x} \in \mathbb{R}^n$, such that the group action is described by $\mathbf{x} \rightarrow R\mathbf{x} + \Delta \mathbf{x}$ [29].

To model an FSA from the system $\mathcal{S}$, a particular class of trajectories is fundamental: the steady-state motions, otherwise known as *relative equilibria*. Formally, we can define it as follows.

Definition 2 (Relative equilibrium). Let $(\mathcal{G}, \mathcal{M}, \Psi)$ be a symmetry group in the sense of Definition 1, $\mathfrak{g}$ be the Lie algebra of $\mathcal{G}$ with exponential map $\exp : \mathfrak{g} \rightarrow \mathcal{G}$, and a constant control input $\bar{\mathbf{u}}$. A trajectory $\varphi_u(\mathbf{x}_0, \cdot)$ is a relative equilibrium of the system (1) when there exists a Lie algebra element of $\xi \in \mathfrak{g}$ such that:

$$\begin{cases} \varphi_u(\mathbf{x}_0, t) = \Psi(\exp(\xi t), \mathbf{x}_0) \\ \mathbf{u}(t) = \bar{\mathbf{u}}. \end{cases} \quad (4)$$

$\forall t \geq 0$.

As it will be detailed later on this section, the relative equilibria constitute the FSA states. In particular, we consider the dynamical system remaining in the FSA states with null duration, i.e when reaching a relative equilibrium, the system instantaneously has to proceed with a smooth transition to another one. For that, we consider trajectories that start and end on a relative equilibrium, smoothly shifting from one to another, which we denominate as *transitions*.

Definition 3 (Transition). A transition is a finite-time trajectory that makes a passage between two relative equilibria and thus characterized by:

- (a) a time duration T ;
- (b) a control signal $\mathbf{u} : [0, T] \rightarrow \mathbb{R}^m$;
- (c) and $\mathbf{x}(0)$ and $\mathbf{x}(T)$ belonging to a relative equilibrium characterized by (4).

Transitions can be modeled arbitrarily as a customized trajectory. Alternatively, they can be computed with optimal control methods, e.g. minimizing an intrinsic energy of the system [30]. Transitions can also start and end on the same relative equilibrium, constituting self-loops on the FSA.

Invariance w.r.t. a group action leads us to define equivalence of trajectories, a fundamental concept to allow the concatenation of finite flows. Intuitively, two trajectories are equivalent if they can be exactly superimposed with time translation and the action of a symmetry group.

Definition 4 (Equivalence of trajectories). Given two trajectories

$$\begin{aligned}\pi_1 : t \in [t_{1,i}, t_{1,f}] &\rightarrow (\mathbf{x}_1(t), \mathbf{u}_1(t)) \text{ and} \\ \pi_2 : t \in [t_{2,i}, t_{2,f}] &\rightarrow (\mathbf{x}_2(t), \mathbf{u}_2(t)),\end{aligned}$$

they are equivalent if

- 1) $t_{1,f} - t_{1,i} = t_{2,f} - t_{2,i}$ and
- 2) There exists $g \in \mathcal{G}$, $T \in \mathbb{R}$ such that $(\mathbf{x}_1(t), \mathbf{u}_1(t)) = (\Psi_g(\mathbf{x}_2(t-T), \mathbf{u}_2(t-T))) \forall t \in [t_{1,i}, t_{1,f}]$.

A class of equivalent trajectories are called motion primitives. In particular, transitions are motion primitives and so are relative equilibria. As shown in Definition 4, a trajectory can easily be shifted via the symmetry action Ψ_g . This, together with the boundary conditions of transitions (see point c) in Definition 3) allows various concatenations of transitions. Lastly, a relative equilibrium (defined by $\bar{\mathbf{u}}$ and ξ) can be turned into a transition by fixing a duration T , defining the control signal to be constantly equal to $\bar{\mathbf{u}}$ and choosing a pair of start and end states, such that there exists a $\xi \in \mathfrak{g}$ with $x(T) = \Psi(\exp(\xi T), x_0)$. This will later become the self-loops in our FSA.

Denoting by $\mathcal{P}(\mathcal{S}, \mathcal{G})$ the set of all primitives π for a system $\mathcal{S}$ with a symmetry group $(\mathcal{G}, \mathcal{M}, \Psi)$, and given two primitives

$$\begin{aligned}\pi_1 : t \in [0, T_1] &\rightarrow (\mathbf{x}_1(t), \mathbf{u}_1(t)) \text{ and} \\ \pi_2 : t \in [0, T_2] &\rightarrow (\mathbf{x}_2(t), \mathbf{u}_2(t)),\end{aligned}$$

we can concatenate π_1 and π_2 , written as

$$\pi_1 \pi_2 : [0, T_1 + T_2] \rightarrow \mathcal{M} \times \mathbb{R}^m,$$

if there exists $g_c \in \mathcal{G}$ such that $\mathbf{x}_1(T_1) = \Psi(g_c, \mathbf{x}_2(0))$. Thus,

$$\pi_1 \pi_2(t) = \begin{cases} (\mathbf{x}_1(t), \mathbf{u}_1(t)), & \text{if } t \leq T_1 \\ (\Psi(g_c, \mathbf{x}_2(t - T_1)), \mathbf{u}_2(t - T_1)), & \text{otherwise.} \end{cases}$$

Remark. If $\pi_1, \pi_2 \in \mathcal{P}(\mathcal{S}, \mathcal{G})$ and can be concatenated to form $\pi_1 \pi_2$, then $\pi_1 \pi_2 \in \mathcal{P}(\mathcal{S}, \mathcal{G})$. The proof is given in [8].

Our special task is to select relative equilibria configurations and compute the transitions, such that we build a set of pieces of trajectories. Then, the system evolution can be driven by concatenating them.

B. Finite State Automaton

Given the needed mathematical tools, now we are able to define our FSA, which will be further used in the receding horizon graph search method in Section III.

Definition 5 (Finite state automaton). An FSA is a 5-tuple (Q, S, γ, q_0, Q_f) composed by:

- Q is a finite set of relative equilibria;
- S is a finite set of transitions;

- $\gamma : Q \times S \rightarrow Q$ is the update function defining the switch between two relative equilibria after the execution of a transition;
- $q_0 \in Q$ is the initial relative equilibrium;
- $Q_f \subseteq Q$ is the set of final relative equilibria.

The FSAs considered in this work are deterministic, i.e. given $q \in Q$ and $\pi \in S$, the result of the transition function $\gamma(q, \pi)$ can be computed deterministically.

C. Discrete System

For a set of transitions of the invariant system (1), the transition function γ defines a discretization of the state-space. In a specific case when all transitions present the same fixed duration T_s , we have a discrete-time nonlinear time-invariant system given by

$$\mathbf{x}_{k+1} = f_d(\mathbf{x}_k, \mathbf{u}_k) \quad (5)$$

with $k \in \mathbb{N}$, the vector field $f_d : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$, the state vector $\mathbf{x} \in \mathcal{M} \subseteq \mathbb{R}^n$ and the input vector $\mathbf{u} \in \mathbb{R}^m$. With a sample time T_s , the product $k \cdot T_s$ represents the corresponding continuous time. The input vector $\mathbf{u}_k(t)$ for $t \in [0, T_s]$ is given by the applied transition $\pi(k)$. Please note that the sequence $\mathbf{u} : [0, T_s] \rightarrow \mathbb{R}^m$ for each transition is pre-computed offline.

III. RECEDING HORIZON CONTROL USING GRAPH SEARCH

This section presents the combination of graph search with a receding horizon control (RHC) concept. Particularly, the recursive feasibility property known from RHC is transferred to the graph search algorithm.

A. Problem Statement

Consider the problem of controlling the discrete-time nonlinear time-invariant system (5), subject to constraints

$$\mathbf{x}(k) \in \mathcal{X}, \mathbf{u}(k) \in \mathcal{U}, \forall k \in \mathbb{N} \quad (6)$$

where $\mathcal{X} \subseteq \mathcal{M} \subseteq \mathbb{R}^n$, $\mathcal{U} \subseteq \mathbb{R}^m$. Assume a full measurement or estimate of the state $\mathbf{x}(k)$ is available at the current time k .

We define the cost function

$$J_{k \rightarrow k+N|k} = p(\mathbf{x}_{k+N|k}) + \sum_{i=0}^{N-1} q(\mathbf{x}_{k+i|k}, \mathbf{u}_{k+i|k}), \quad (7)$$

where N is the horizon length of the optimal control problem (OCP) and p and q are penalty functions for the state and input.

The corresponding OCP is formulated as

$$\begin{aligned} &\text{minimize} && J_{k \rightarrow k+N|k} \\ &U_{k \rightarrow k+N|k} && \end{aligned} \quad (8a)$$

subject to

$$\mathbf{x}_{k+i+1|k} = f_d(\mathbf{x}_{k+i|k}, \mathbf{u}_{k+i|k}), \quad i = 0, \dots, N-1, \quad (8b)$$

$$\mathbf{u}_{k+i|k} \in \mathcal{U}, \quad i = 0, \dots, N-1, \quad (8c)$$

$$\mathbf{x}_{k+i|k} \in \mathcal{X} \quad i = 0, \dots, N-1, \quad (8d)$$

$$\mathbf{x}_{k+N|k} \in \mathcal{X}_f, \quad (8e)$$

$$\mathbf{x}_{k|k} = \mathbf{x}(k). \quad (8f)$$

Here, $U_{k \rightarrow k+N|k}$ is the vector of stacked control inputs $(\mathbf{u}_{k|k}, \mathbf{u}_{k+1|k}, \dots, \mathbf{u}_{k+N|k})$.

The OCP is solved repeatedly after a time step duration T and with updated values for the states and constraints, which establishes the RHC. Choosing the time step equal to the motion primitive duration means that exactly one motion primitive is executed between consecutive values of $\mathbf{x}$ and $\mathbf{u}$.

B. Recursive Feasibility

A set $\mathcal{C} \subseteq \mathcal{X}$ is a control invariant set for the system (5) subject to constraints (6) if

$$\begin{aligned} \mathbf{x}(k) \in \mathcal{C} &\implies \exists \mathbf{u}(k) \in \mathcal{U} \text{ such that} \\ \mathbf{x}(k+1) &\in \mathcal{C}, \forall k \in \mathbb{N}. \end{aligned} \quad (9)$$

Lemma 1. *If $\mathcal{X}_f$ is a control invariant set of the system with constraints (8) with $N > 1$, then (8e) ensures recursive feasibility of the RHC.*

Proof. The proof is given in [31]. $\square$

In a representation of the system as an FSA, the transitions determine the control input and the change of the state variables of the system (5). The available transitions and initial state variable depend on the state. We can reformulate the condition of control invariant sets for FSAs as follows.

Definition 6 (Control invariant set for an FSA). A set $\mathcal{C} \subseteq \mathcal{X}$ is a control invariant set for the system (5) if

$$\begin{aligned} \mathbf{x}(k) \in \mathcal{C} \text{ s.t. } q(k) \in Q &\implies \exists \pi \in S \text{ such that} \\ \mathbf{x}(k+1) \in \mathcal{C} \text{ with } q(k+1) \in Q &\text{ and} \\ \gamma(q(k), \pi) = q(k+1), &\forall k \in \mathbb{N}. \end{aligned} \quad (10)$$

Note that the relative equilibrium q follows from the state vector $\mathbf{x}$.

Theorem 1. *RHGS achieves recursive feasibility if the generated sequence of transitions ends in an automaton state and a system state that form a control invariant set.*

Proof. Follows directly from Lemma 1 with Definition 6 of control invariant sets for FSAs. $\square$

A case that is especially simple is if the generated sequence of transitions ends in an automaton state from where there exists a transition which keeps the state vector $\mathbf{x}(k)$ at an equilibrium of the system. In an equilibrium of the system $f_d(\mathbf{x}(k), \mathbf{u}(k)) = \mathbf{x}(k)$, so $\mathbf{x}(k)$ clearly represents a control invariant set.

C. Graph Search Loop

This subsection presents the RHGS approach to solve the RHC problem (8). Mapping the RHC problem to graph search based on an FSA, we have the following correspondences. As mentioned before, the motion primitive duration and the duration of a time step are chosen to be equal to T , so exactly one motion primitive is executed between consecutive states and control inputs at time $k+i$ and $k+i+1$. The FSA incorporates the constraints on system dynamics (8b), on control inputs (8c), and on the part of the state vector that is

invariant w.r.t. group actions Ψ (8d), (8e). The graph search incorporates the part of the state vector that is affected by group actions Ψ (8d), (8e), and the cost function (8a). In the OCP (8), the input $\mathbf{u}(k+i)$ is kept constant during every time step. In the transitions of the FSA, the control inputs can change over the duration of a primitive as explained in subsection II-C.

Generally, the FSA represents the discrete system model and therefore the discrete system trajectories that can be followed when performing graph search. In order to explore the search space, a search tree $\mathcal{T}$ is constructed. A vertex in the tree τ consists of the tuple $(q, \mathbf{x}, i, J)$, whose elements are the corresponding relative equilibrium of the automaton $q \in Q$, the location in the search space $\mathbf{x} \in \mathcal{X}$, the level i , and the value of the cost function J respectively. Please note that a level i in the tree directly corresponds to the time step i in the OCP (8). The relative equilibrium q of a tree vertex τ together with the transition function γ determine the possible child vertices.

Algorithm 1 shows the main steps of an informed search algorithm loop. An informed search is characterized by an estimate of the remaining cost-to-go (CTG), which supports more efficient searches. The algorithm ensures optimality of the returned path with an admissible and underestimating cost estimation, similar to A^* . In contrast to A^* , the algorithm does not keep track of a closed list. Due to the limited planning horizon, the algorithm will still not get stuck in endless loops. The following paragraphs explain the algorithm.

The algorithm initializes the open list with the root vertex τ_0 (line 1). The root vertex contains the current state $\mathbf{x}(k)$ of the system at the start of each iteration, which determines the relative equilibrium of the FSA as well as the initial location in the search space.

The main loop starts by sorting the vertices in the open list ascending by their cost function values (line 3). The vertex τ_p with the lowest cost function value is inspected further and removed from the open list (line 5).

We assume that evaluating edges to a vertex can be separated into calculating the cost function (8a) value and checking for constraints (8d). Further we assume that checking for constraints is computationally more demanding. The constraints are checked only on vertices selected for further inspection (line 6). If the vertex does not satisfy all constraints, the algorithm skips to the next vertex in the open list.

If the vertex satisfies all constraints, the algorithm checks if it has found the optimum. The condition that must be met is that the lowest-cost vertex that fulfills the constraint (8e) is found at the planning horizon N (line 9). This results in a limited planning horizon which reduces computational effort. The algorithm consequently expands the root node only up to a tree depth N , and the path to the lowest-cost vertex at depth N is returned (line 9).

If the selected vertex is not the optimum, the algorithm generates child vertices based on the relative equilibrium q , the transition function γ , and the state $\mathbf{x}$ of the parent vertex (line 12). When vertices are expanded, the possible transitions

and therefore successors are limited such that an equilibrium state is reached within the horizon N . The distance between two vertices in a graph is the number of edges in a shortest connecting path. Let i_{τ_c} be the distance from the root vertex of the search tree to the child vertices of the expanded vertex, and $d_{\text{FSA}}: Q \times Q \rightarrow \mathbb{N}$ the distance between two states in the FSA. Then, only vertices are returned as successors for which it holds that

$$d_{\text{FSA}}(q_{\tau_c}, q_{\tau_{\text{eq}}}) \leq N - i_{\tau_c}, \quad (11)$$

where q_{τ_c} is the state of the child vertex, and $q_{\tau_{\text{eq}}}$ is the equilibrium state closest to the child vertex. This ensures recursive feasibility.

For each successor vertex, a lazy cost estimate is computed. In informed search algorithms, the cost of a vertex consists of two parts: the cost-to-come (CTC) $g: \tau \in \mathcal{T} \rightarrow \mathbb{R}$, which is the cost of the path to the current vertex and the CTG, estimated by the heuristic $h: \tau \in \mathcal{T} \rightarrow \mathbb{R}$. The cost chosen here determines the functional that is minimized by the graph search which corresponds to (8a). In applications such as robotic trajectory planning, computing the CTC, also known as evaluating the edges leading to a vertex, includes checking for collision constraints, which has a high computational cost [5]. The number of costly edge evaluations can be reduced if a lazy estimate of the cost is computed here, and only substituted with the real cost if it is necessary [17], [18], [19].

Adding the successor vertices to the open list wraps up the graph search loop (line 15).

Algorithm 1: Receding horizon graph search

Input : initial vertex τ_0 , FSA, goal set $\mathcal{X}_f$
Output: path to best vertex

```

1  $L_{\text{open}} \leftarrow \tau_0$ 
2 while  $L_{\text{open}}$  not empty do
3   Sort  $L_{\text{open}}$  ascending by  $J(\tau) = g(\tau) + h(\tau)$ 
4    $\tau_p \leftarrow L_{\text{open}}[0]$ 
5    $L_{\text{open}} \leftarrow L_{\text{open}} \setminus \tau_p$ 
6   if not  $\text{is\_valid}(\tau_p)$  then
7     continue
8   end
9   if  $i_{\tau_p} = N$  and  $x_{\tau_p} \in \mathcal{X}_f$  then
10    return path from  $\tau_0$  to  $\tau_p$ 
11  end
12   $\text{successors} \leftarrow \text{expand}(\tau_p)$ 
13  forall  $\tau \in \text{successors}$  do
14     $\text{lazy\_eval}(\tau)$ 
15     $L_{\text{open}} \leftarrow L_{\text{open}} \cup \tau$ 
16  end
17 end
18 return failure

```

Example 1. We illustrate the RHGS with an example. Figure 1a shows an example FSA consisting of two states. State 2 is the current state of the system and state 1 an accepting state, i.e. an equilibrium of the underlying dynamical system. The

(a) Exemplary FSA with initial state 2 and accepting state 1 unrolled over the horizon $N = 3$.

(b) An example of a search tree resulting from RHGS with horizon $N = 3$ on the FSA in Figure 1a. The vertex' automaton state is indicated with the number in superscript.

Fig. 1: Example FSA and RHGS.

states are unrolled over time to show that possible transitions change over the horizon. There exist transitions between the states as well as transitions that loop back to a state. Figure 1b shows an exemplary search tree resulting from a receding horizon graph search with horizon length $N = 3$. Vertices are numbered consecutively with their index, and a subscript indicates the corresponding state of the FSA.

IV. TRAJECTORY PLANNING FOR CAR-LIKE ROBOTS

This section applies the general approach of recursively feasible RHGS on systems with symmetry to networked trajectory planning for car-like robots. In subsection IV-A a system model is shown to exhibit the symmetry property. In subsection IV-B the FSA for multiple car-like robots, which we call MPA, is introduced. In subsection IV-C the RHGS algorithm is adjusted to the trajectory planning task.

For this section, let I and $\mathbf{0}$ denote, respectively, the identity matrix and the null matrix (or vector) with appropriate dimension.

A. Kinematic Single-Track Model

The nonlinear kinematic single-track model according to [32] is more precise than the linear point-mass model (see e.g. [33]). It is suitable for small velocities. Figure 2 shows a general overview of the vehicle model's variables and their

Fig. 2: Kinematic bicycle model of a vehicle.

relations. Following [34], we assume that no slip occurs on the front and rear wheels, and no forces act on the vehicle. The resulting equations are

$$\begin{cases} \dot{x}(t) = v(t) \cdot \cos(\psi(t) + \beta(t)), \\ \dot{y}(t) = v(t) \cdot \sin(\psi(t) + \beta(t)), \\ \dot{\psi}(t) = v(t) \cdot \frac{1}{L} \cdot \tan(\delta(t)) \cos(\beta(t)), \\ \dot{v}(t) = u_v(t), \\ \dot{\delta}(t) = u_\delta(t), \end{cases} \quad (12)$$

with

$$\beta(t) = \tan^{-1} \left(\frac{\ell_r}{L} \tan(\delta(t)) \right), \quad (13)$$

where $x \in \mathbb{R}$ and $y \in \mathbb{R}$ describe the position of the center of gravity (CG), $\psi \in [0, 2\pi)$ is the orientation, $\beta \in [0, 2\pi)$ is the side slip angle, $\delta \in [0, 2\pi)$ and $u_\delta \in \mathbb{R}$ are the steering angle and its derivative respectively, $v \in \mathbb{R}$ and $u_v \in \mathbb{R}$ are the velocity and acceleration respectively, L is the wheelbase length and ℓ_r is the length from the rear axle to the CG.

The inputs are chosen to be the change in the steering angle u_δ and the acceleration u_v . For the sake of simplicity, the input effects are assumed to influence the dynamics instantly without delay.

The nonlinear dynamical control system $\mathcal{S}$ defined in (12) is compactly written as (1) with the state vector

$$\mathbf{x} = \begin{pmatrix} x & y & \psi & v & \delta \end{pmatrix}^T \in \mathbb{R}^5, \quad (14)$$

the control input

$$\mathbf{u} = \begin{pmatrix} u_v & u_\delta \end{pmatrix} \in \mathbb{R}^2 \quad (15)$$

and the vector field f in terms of (12).

In the following, we refer to the pose of the system, which is described by the first three components of the state vector, as

$$\mathbf{p} = \begin{pmatrix} x & y & \psi \end{pmatrix}^T. \quad (16)$$

The results can be easily extended to the complete state vector $\mathbf{x}$, as the other components are decoupled from the first three and are not affected by the following transformations. For this system, the symmetry group $\mathcal{G}$ is a subgroup of $SE(3)$, i.e. the class of combined rotations and translations

about a vertical axis. It can be nicely represented by matrix multiplication when switching to homogeneous coordinates:

$$\mathcal{G} = \left\{ g \in SE(3) : g = \begin{pmatrix} R_{SO(3)} & \Delta \mathbf{p} \\ \mathbf{0} & 1 \end{pmatrix} \right\}, \quad (17)$$

where

$$R_{SO(3)} = \begin{pmatrix} \cos(\Delta\psi) & -\sin(\Delta\psi) & 0 \\ \sin(\Delta\psi) & \cos(\Delta\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \in SO(3), \quad (18)$$

is a rotation of an angle $\Delta\psi$ and

$$\Delta \mathbf{p} = \begin{pmatrix} \Delta x & \Delta y & \Delta\psi \end{pmatrix}^T \in \mathbb{R}^2 \times S^1 \quad (19)$$

is a translation.

Henceforth on this paper, consider the notation $g := g(\Delta \mathbf{p})$.

Proposition 1. The group action Ψ_g for the kinematic single-track model (12) can be given as

$$\Psi_g(\mathbf{x}) = \begin{pmatrix} R_{SO(3)} \cdot \mathbf{p} + \Delta \mathbf{p} \\ v \\ \delta \end{pmatrix}, \quad (20)$$

which is equivalent to

$$\Psi_g(\mathbf{x}) = R\mathbf{x} + \Delta \mathbf{p}, \quad (21)$$

considering

$$R = \begin{pmatrix} R_{SO(3)} & \mathbf{0} \\ \mathbf{0} & I \end{pmatrix} \in SO(5), \quad (22)$$

$$\Delta \mathbf{p} = \begin{pmatrix} \Delta \mathbf{p} \\ \mathbf{0} \end{pmatrix} \in \mathbb{R}^2 \times S^1 \times \{0\}^2. \quad (23)$$

Proof. Let $f(\mathbf{x}, \mathbf{u}) : \mathcal{M} \rightarrow T\mathcal{M}$, where $T\mathcal{M}$ is the tangent bundle of a differentiable manifold $\mathcal{M}$. Then, $\Psi : \mathcal{G} \times \mathcal{M} \rightarrow \mathcal{M}$ can be lifted to $T_{\mathbf{x}}\mathcal{M}$ for $\mathbf{x} \in \mathcal{M}$, such that $\Psi^{T_{\mathbf{x}}\mathcal{M}} : \mathcal{G} \times T_{\mathbf{x}}\mathcal{M} \rightarrow T_{\mathbf{x}}\mathcal{M}$, via [29]:

$$\Psi^{T_{\mathbf{x}}\mathcal{M}}(g, f) = \frac{d\Psi_g(\mathbf{x})}{d\mathbf{x}} \cdot f. \quad (24)$$

The relation (3) can be proven in terms of the equivariance of the vector field f . The vector field f is equivariant w.r.t. the symmetry action Ψ if [29]:

$$f(\Psi_g(\mathbf{x}), \mathbf{u}) = \Psi_g^{T_{\mathbf{x}}\mathcal{M}}(f(\mathbf{x}, \mathbf{u})), \quad \forall \mathbf{x} \in \mathcal{M}. \quad (25)$$

Deriving $\Psi_g(\mathbf{x}) = R\mathbf{x} + \Delta \mathbf{p}$ in (21) w.r.t. $\mathbf{x}$ gives $\frac{d\Psi_g(\mathbf{x})}{d\mathbf{x}} = R$. Replacing this relation on the definition (24), we get

$$\Psi_g^{T_{\mathbf{x}}\mathcal{M}}(f(\mathbf{x}, \mathbf{u})) = R \cdot f(\mathbf{x}, \mathbf{u}). \quad (26)$$

Note that $\frac{d}{dt}(\psi + \Delta\psi) = \dot{\psi}$. Then, from $f(\mathbf{x}, \mathbf{u})$ given by (12) and the from (26), we have

$$\begin{aligned} f(\Psi_g(\mathbf{x}), \mathbf{u}) &= \begin{pmatrix} v \cos(\psi + \Delta\psi + \beta) \\ v \sin(\psi + \Delta\psi + \beta) \\ \begin{pmatrix} \dot{\psi} & \dot{v} & \dot{\delta} \end{pmatrix}^T \end{pmatrix} \\ &= \begin{pmatrix} v \cos(\psi + \beta) \cos(\Delta\psi) - v \sin(\psi + \beta) \sin(\Delta\psi) \\ v \sin(\psi + \beta) \cos(\Delta\psi) + v \cos(\psi + \beta) \sin(\Delta\psi) \\ \begin{pmatrix} \dot{\psi} & \dot{v} & \dot{\delta} \end{pmatrix}^T \end{pmatrix} \\ &= R \cdot f(\mathbf{x}, \mathbf{u}) = \Psi_g^{\mathcal{T}_\infty \mathcal{M}}(f(\mathbf{x}, \mathbf{u})), \end{aligned}$$

proofing the equivariance of the vector field by satisfying (25). $\square$

B. Motion Primitive Automaton

We apply the concept of modeling the system dynamics as an FSA following the approach presented in section II to car-like robots. We refer to this specific automaton as an MPA.

An MPA consists of two main types of components, the relative equilibria $q \in Q$ and the motion primitives $\pi \in S$ with duration T_s . The constraints of the system (8b) and (8c) need to be considered when creating motion primitives. These motion primitives can be precomputed by solving a corresponding optimization problem.

If vehicles plan their trajectories cooperatively, they need to create a representation of the dynamical system consisting of all vehicles who should cooperate. We obtain the MPA of all cooperating vehicles by using synchronous side-by-side composition [35]. Suppose two MPA A and B are given by the five-tuples

$$\begin{aligned} A &= (Q_A, S_A, \gamma_A, q_{0,A}, Q_{f,A}) \\ B &= (Q_B, S_B, \gamma_B, q_{0,B}, Q_{f,B}). \end{aligned} \quad (27)$$

The synchronous side-by-side composition C is given by

$$\begin{aligned} Q_C &= Q_A \times Q_B \\ S_C &= S_A \times S_B \\ q_{0,C} &= (q_{0,A}, q_{0,B}) \\ Q_{f,C} &= Q_{f,A} \times Q_{f,B} \end{aligned} \quad (28)$$

and the update function is given by

$$\gamma_C((q_A, q_B), (s_A, s_B)) = ((q'_A, q'_B)) \quad (29)$$

where

$$\begin{aligned} q'_A &= \gamma_A(q_A, s_A) \\ q'_B &= \gamma_B(q_B, s_B) \end{aligned} \quad (30)$$

$\forall q_A \in Q_A, q_B \in Q_B, s_A \in S_A, s_B \in S_B$. Figure 3 illustrates an example of a synchronous side-by-side composition of two MPAs.

During trajectory planning, we exploit the symmetry of the system and shift the precomputed motion primitives to the current state of the system. Trajectory planning includes collision checks when a transition is taken. Since the underlying trajectory carries information about time, the transitions that

Fig. 3: Synchronous side-by-side composition C of two MPAs A and B .

connect relative equilibria need to share the same duration T_s . Then, the time for collision checks between primitives of two vehicles matches.

C. Trajectory Planning

The goal of trajectory planning in this paper is the search for a trajectory that (i) keeps a vehicle as close as possible to a reference trajectory (ii) while avoiding obstacles (iii) under consideration of the vehicle dynamics (e.g. (12)). The first subgoal is addressed by minimizing a suitable cost function, the second subgoal is considered during edge evaluation and the last subgoal is achieved by construction of the MPA. This section presents the specific implementation of the RHGS as shown in algorithm 1 in section III.

During graph search, a search tree is constructed from a root vertex τ_0 . The information contained in each vertex τ of the search tree $\mathcal{T}$ is a tuple $(q, \mathbf{p}, i, J)$, whose elements are the corresponding state of the automaton, the vehicle pose, the distance to the root vertex, and the cost function, respectively.

At the beginning of the control loop, the vehicle's states are measured and predicted using the vehicle model until the time when the current control input takes effect. The speed v and the steering angle δ determine the state q . The root vertex τ_0 of the search tree is initialized with this state and the pose $\mathbf{p}$.

During the graph search iterations, the vertex with the lowest cost is expanded. The cost of a vertex consists of the CTC and the CTG. The cost function to minimize in the given trajectory planning problem is

$$J = \sum_{i=0}^N (\mathbf{x}_{k+i|k} - \mathbf{x}_{\text{ref},k+i|k})^T \mathbf{Q} (\mathbf{x}_{k+i|k} - \mathbf{x}_{\text{ref},k+i|k}). \quad (31)$$

With n_{veh} as the number of vehicles, $\mathbf{x} \in \mathbb{R}^{5n_{\text{veh}}}$ is the state vector, $\mathbf{Q} \in \mathbb{R}^{5n_{\text{veh}} \times 5n_{\text{veh}}}$ is a positive semi-definite, block diagonal matrix with n_{veh} times repeated entries $\mathbf{q}$

$$\mathbf{q} = \begin{pmatrix} \mathbf{I}_2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \quad (32)$$

and $\mathbf{x}_{\text{ref},k+i|k} \in \mathbb{R}^{5n_{\text{veh}}}$ is the reference trajectory sampled at the time steps in the horizon $k+i$. The reference trajectory is

generated as follows. First, a reference path is chosen, e.g. a line connecting two points. Then, given a reference velocity, points corresponding to the time step on this line are sampled. In urban environments, one could imagine the reference path as the lane a vehicle should follow and the reference velocity as the speed limit.

Trajectory planning for ground vehicles using graph search belongs to a class of graph search problems with expensive-to-evaluate edges [5]. Computationally most demanding are collision checks, either between two vehicles or a vehicle and an obstacle. Therefore, we employ a lazy approach as inspired by [18].

The lazy edge evaluation corresponds to an evaluation of the cost function (31) without collision checks. In RHGS, the cost at a vertex is comprised of two parts. The first part is the CTC. Given a vertex τ at depth i_τ , the CTC is the euclidean distance squared from the position to the reference position at time $k + i$

$$J_{\text{CTC}} = \sum_{i=0}^{i_\tau} (\mathbf{x}_{k+i|k} - \mathbf{x}_{\text{ref},k+i|k})^T \mathbf{Q} (\mathbf{x}_{k+i|k} - \mathbf{x}_{\text{ref},k+i|k}). \quad (33)$$

The second part of the cost is the CTG, which must underestimate the remaining cost from a vertex in the search tree. In our case of trajectory planning with a reference, the CTG is computed as the distance from every future position to the reference position assuming the vehicle would move towards the reference position with maximum velocity:

$$J_{\text{CTG}}(i_\tau) = \sum_{i=i_\tau}^N \max \left(0, (\mathbf{x}_{k+i|k} - \mathbf{x}_{\text{ref},k+i|k})^T \mathbf{Q} (\mathbf{x}_{k+i|k} - \mathbf{x}_{\text{ref},k+i|k}) - i \cdot v_{\max} \cdot T_s \right). \quad (34)$$

The exact edge evaluation requires collision checks on the generated trajectories with both other vehicles and obstacles. A collision between two vehicles occurs if the areas occupied by each vehicle during a simultaneously executed motion primitive overlap. In order to check this condition, vehicles are approximated with rectangles of length L and width W . Furthermore, the occupied area during a motion primitive is approximated using a polygon as shown in Figure 4. On the left, the occupied space during a primitive of straight motion is depicted, where the polygonal approximation does not introduce an error. On the right, the occupied space during a primitive of circular motion is depicted. The five-sided polygon overapproximates the actual space occupied, but simplifies checking the occupied areas of multiple vehicles for overlaps. Inter-vehicle collisions and obstacle collisions can efficiently be identified using the separating axis theorem [4]. Both vehicles and obstacles are approximated using convex polygons, so a separating axis exists iff an axis through one of the polygon sides represents a separating axis. Consequently, we iterate over all sides of the polygons of the vehicles or obstacles in question until we find a separating axis. An axis separates two polygons if the projection of polygon corner points onto a line perpendicular to the axis results in the

Fig. 4: Occupied space in green during a primitive of straight motion (left) and of circular motion (right) with polygonal approximation in orange. The vehicle geometry is approximated by a rectangle.

TABLE I: Parameters used in simulation

Parameter	Value	Unit
ℓ_τ	0.075	[m]
L	0.15	[m]
T_s	0.4	[s]
N	5	[1]

projected corner points of one polygon being on the left and the others being on the right of the axis.

As explained in section III, the proposed method returns the path to the cheapest vertex at the desired depth N .

The worst-case time complexity of the algorithm is $\mathcal{O}(b_{\max}^N)$, where $b_{\max} \in \mathbb{N}$ is the maximum branching factor of the MPA. The best-case time complexity is $\mathcal{O}(b_{\max} \cdot N)$. The average time complexity depends significantly on the quality of the heuristic and the specific scenario. The space complexity is similar to the time complexity since the algorithm keeps track of all expanded vertices in the tree. The number of expanded vertices is therefore a reasonable indication on both time and space complexity. The branching factor b_C of a vertex in the synchronous composition of two MPA A and B is the product of branching factors of the original vertices $b_C = b_A \cdot b_B$. This means that the complexity of the algorithm grows exponentially with the number of vehicles involved in the scenario.

V. EVALUATION

This section details the simulation and experiment setup with which we validate our approach. The section first highlights the importance of the recursive feasibility property in simulation. Then, the effect of the finite planning horizon is analyzed. Lastly, we run the algorithm in an experiment in the CPM Lab Lab [6]. It is an 1:18 model-scale, open source platform for networked and autonomous vehicles.

We evaluate the presented framework in simulation and experiments. The computational hardware is an off-the shelf desktop PC. Simulations and experiments were conducted on multiple computers operating in Windows, Ubuntu and MacOS. The software is implemented in MATLAB and is available online [2], [3]. The performance of the code could be improved with an implementation with a more performing programming language like C++, but still we could achieve reasonable real-time results. Table I shows the parameters used in simulation. The parameters correspond to the model-scale experimental platform CPM Lab Lab. The MPA used in simulation and experiments is based on the kinematic

Fig. 5: An MPA with two speeds and three steering angles. The speed v is shown at the top, the steering angle δ at the bottom of each state. For clarity, the time dependency to ensure recursive feasibility is omitted.

bicycle model (12). The relative equilibria are chosen such that transitions between them are possible in the given sample time T_s for the model-scale vehicles used in the experiments. The transitions in the MPAs in Figure 5, Figure 7a and Figure 7b use either constant control inputs if it is a transition looping back to the same relative equilibrium, or a linear change of control inputs over time if it is a transition connecting to different relative equilibria.

A video of the simulations and experiments is available online [1].

A. Trajectory planning

We demonstrate the effectiveness of trajectory planning with RHGS in simulation.

The simulation uses the MPA shown in Figure 5 that consists of six states with two speeds and five steering angles.

We test the trajectory planner in a scenario where it needs to find a trajectory that keeps the vehicle close to the reference, which is a line from left to right with $y = 2$ m. The planner needs to avoid obstacles that travel from the top downwards and from the bottom upwards with a speed of $v = 1$ m s⁻¹.

Figure 6 shows the resulting traveled and planned trajectory at four key time steps. The planner finds a trajectory from left to right that avoids the dynamic obstacles.

B. Recursive Feasibility

We demonstrate the importance of ensuring recursive feasibility during RHGS in simulation.

The simulations use simple MPAs with only three speeds. The MPAs and their transitions over a horizon of $N = 2$ are shown in Figure 7a and Figure 7b. The figures showcase the effect of restricting the transitions such that the automaton ends on an equilibrium state at the end of the horizon. Note that with the restriction in Figure 7b, the state with the highest speed $v = 3/2$ m s⁻¹ cannot be reached.

Fig. 6: The planned trajectory avoids moving obstacles to follow a straight reference trajectory from left to right. Obstacles on the left move upwards, while obstacles on the right move downwards.

Both MPAs are evaluated in the scenario shown in Figure 8a. The scenario is chosen such that it illustrates possible unsafe behavior of Figure 7a, even though a full stop can technically be reached during the horizon. In the worst case, the vehicle has a high speed very close to the obstacle. This is the case if the planner chooses the state sequence "3, 2" at time k , which results in the vehicle being right in front of the obstacle at the prediction horizon. Then, at time $k + 1$, the remaining distance to the obstacle is

$$d_o = T_s \cdot 0.5 \cdot (3/2 \text{ m s}^{-1} + 3/4 \text{ m s}^{-1}). \quad (35)$$

In order to avoid the collision, the vehicle needs to stop before traveling the distance d_o . Since the input u_v corresponds to the acceleration and is constant per time step, according to the model (12) the distance traveled at time k of the horizon is

$$d(k+i) = d(k) + T_s \cdot i \cdot v(k) + T_s^2 \cdot \sum_{j=k}^{k+i-1} \left(k+i - \frac{2j+1}{2} u_v(j) \right). \quad (36)$$

According to the MPA in Figure 7a, the minimum acceleration which reaches the speed $v(k+N) = 0$ from the maximum velocity $v_{\max}$ over the horizon of N steps is

$$u_{\min} = -\frac{v_{\max}}{N \cdot T_s} \quad (37)$$

(a) Recursive feasibility cannot be guaranteed.

(b) Guarantees recursive feasibility.

Fig. 7: Two MPAs over a receding horizon with $N = 2$.

By inserting (37) into (36), the distance traveled until stop becomes

$$d(k+N) = d(k) + \frac{1}{2} \cdot N \cdot v_{\max} \cdot T_s. \quad (38)$$

For collision avoidance, the distance traveled over the horizon $d(k+N) - d(k)$ must be smaller than the remaining distance d_o

$$\begin{aligned} d(k+N) - d(k) &\stackrel{!}{\leq} d_o \\ \Leftrightarrow 3/2 \text{ m s}^{-1} &\stackrel{!}{\leq} 3/4 \text{ m s}^{-1} \cdot \frac{1}{2}. \end{aligned} \quad (39)$$

Figure 8b shows the simulated trajectory of the position in x -direction for both MPAs. Without enforcing the final state to be in an equilibrium, the planner maneuvers the vehicle in such a way that it cannot find a feasible solution at $t = 1.2$ s. On the other hand, when the final state is always an equilibrium, the vehicle is able to stop in front of the obstacle.

In this specific MPA, the planner would stay feasible if the maximum shortest path length of all states q to the equilibrium was $d_{\text{FSA}}(q, q_{\text{eq}}) = N - 1$, but this behavior can not be generalized to any FSA.

C. Receding Horizon

This subsection analyzes the implications of implementing a receding horizon to a graph search algorithm based on

(a) Scenario in which a vehicle's reference trajectory is obstructed by an obstacle.

(b) Resulting trajectory of RHGS with (red) and without (blue) restricting the final state to be an equilibrium state in the scenario shown above. The grey line indicates the maximum allowed distance given by the obstacle.

Fig. 8: Recursive feasibility simulation setup and result.

simulations. The focus lies on the behavior of the computation time depending on the scenario.

The following simulations use the MPA shown in Figure 5. The three upper states specify the speed $v = 3/4 \text{ m s}^{-1}$ and allow transitions between three steering angles. The lower state is the equilibrium state of the system.

The structure of the scenario is shown in Figure 8a. A vehicle's reference trajectory is obstructed by an obstacle, so the planner avoids the obstacle by steering around it. In the following simulations, the initial distance between the vehicle and the obstacle vary from 1 to 3 m and the horizon N varies from 5 to 30.

The number of expanded vertices during search gives an indication on the necessary computational burden. This number of expanded vertices is shown in Figure 9 depending on the horizon length N and the initial distance between vehicle and obstacle. As presented in subsection IV-C, the worst case computational effort grows exponentially with the horizon N , and the best case computational effort grows linearly with the horizon N . The tendency of increasing computational effort with horizon length is observable in Figure 9. There are some cases where the number of expanded vertices decrease with increasing N . In these cases, the planner finds a trajectory which is clearly better than other alternatives judged by the heuristic function. This is the case if the planned trajectory lies on the reference trajectory. If the vehicle moves in a slight angle to the reference trajectory, many options are explored in order to find a better solution. The computational effort also increases with increasing initial distance between vehicle and obstacle. When the obstacle is encountered, the value of the cost function increases. Therefore, vertices before the obstacle

Fig. 9: Expanded vertices depending on the horizon length N and the initial distance between vehicle and obstacle.

are expanded. The larger the distance to the obstacle when it appears in the horizon, the larger is the number of vertices that need to be considered.

The choice of the control horizon affects the quality of the obtained solution. In the trajectory planning framework presented in section IV, the reference trajectory depends on the control horizon as well. Therefore, there is no fair method to compare the solution quality for different control horizons.

D. Multi-Agent Experiment

This subsection presents results of trajectory planning for multiple model-scale vehicles in simulation and in experiment in the CPM Lab Lab.

The simulations and experiments are based on an MPA which is the synchronous composition of two of the MPA shown in Figure 5.

Figure 10a shows the testing scenario. Two vehicles start opposite of each other and follow reference trajectories to the other's position. This provokes a collision in the center between the vehicles, which the planner has to avoid.

The vehicles successfully avoid the collision in a cooperative fashion by solving the RHC problem for the system consisting of both vehicles. The maximum computation time ranges between 100 ms to 150 ms depending on the hardware. The maximum occurs when the collision is avoided. The median of all 14 control loop iterations of the scenario is 8 ms. The maximum number of expanded vertices during one control loop is 2928. This number of course strongly depends on the MPA and therefore also on the number of vehicles, on the scenario and on the planning horizon N as seen in subsection V-C.

For comparison, if a single vehicle plans a trajectory, the maximum computation time is only 3 ms. On the other hand, when three vehicles are positioned in the corners of an equilateral triangle and plan their trajectory towards the opposite side, the maximum computation time is around 364 s, while the median is at around 50 ms.

(a) Scenario in which two vehicles drive toward each other.

(b) Multiple exposure image taken from the video of a scenario where two vehicles plan trajectories using RHGS to follow their reference to the opposite side of the room.

Fig. 10: Mutli-agent experiment setup and result.

VI. CONCLUSION

This paper presented a trajectory planning approach using motion primitives for multiple vehicles that avoids collisions. By introducing a receding horizon to a graph search algorithm, we reduced the computational load while keeping the important property of recursive feasibility. Using motion primitives can also reduce computational load compared to approaches that consider the complete action space. In contrast to many graph-based approaches, we extended the strategy to reach a single goal by planning a trajectory to follow a reference trajectory. The approach can plan collision-free trajectories for multiple vehicles simultaneously. Experimental results for trajectory planning for two vehicles, and numerical results for trajectory planning for three vehicles were presented.

Abstracting the system dynamics to a motion graph yields the advantage that detailed nonlinear models can be used without significantly increasing the computation time. This paper investigated a limited set of motion primitives. Using more motion primitives might result in a smoother trajectory, but also increases the computation time. The search algorithm could be developed further in future work. State-of-the-art strategies like an incremental search using D* Lite [36], or a Hybrid A* variant which keeps track of a closed list, can be combined with the presented receding horizon graph search. The presented work is real-time capable for systems of up to two car-like robots and a crude motion primitive automaton. Of course, it can be accelerated by increasing the efficiency of the implementation. Apart from that, the source for high computational load is the centralized problem formulation. In future work, the authors plan to investigate the presented control algorithm regarding possible distribution approaches [37].

REFERENCES

- [1] P. Scheffe, M. V. A. Pedrosa, K. Flaßkamp, and B. Alrifae. (2021) Receding Horizon Control Using Graph Search for Multi-

- Agent Trajectory Planning. YouTube. [Online]. Available: <https://youtu.be/7LB71SSOpQE>
- [2] P. Scheffe, M. V. A. Pedrosa, K. Flaßkamp, and B. Alrifae. (2021) Receding Horizon Control Using Graph Search for Multi-Agent Trajectory Planning. GitHub. [Online]. Available: <https://github.com/embedded-software-laboratory/receding-horizon-graph-search>
 - [3] P. Scheffe, M. V. A. Pedrosa, K. Flaßkamp, and B. Alrifae. (2021) Receding Horizon Control Using Graph Search for Multi-Agent Trajectory Planning. Code Ocean. [Online]. Available: <https://codeocean.com/capsule/7778016/tree/v2>
 - [4] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
 - [5] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.
 - [6] M. Kloock, P. Scheffe, J. Maczajewski, A. Kampmann, A. Mokhtarian, S. Kowalewski, and B. Alrifae, "Cyber-physical mobility lab: An open-source platform for networked and autonomous vehicles," *arXiv preprint arXiv:2004.10063*, 2020.
 - [7] D. Delchamps, "Stabilizing a linear system with quantized state feedback," *IEEE Transactions on Automatic Control*, vol. 35, no. 8, pp. 916–924, 1990.
 - [8] E. Frazzoli, M. A. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE transactions on robotics*, vol. 21, no. 6, pp. 1077–1091, 2005.
 - [9] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, no. 3, pp. 407–427, 1999.
 - [10] B. Alrifae, M. G. Mamaghani, and D. Abel, "Centralized non-convex model predictive control for cooperative collision avoidance of networked vehicles," in *2014 IEEE International Symposium on Intelligent Control (ISIC)*, 2014, pp. 1583–1588.
 - [11] D. Ioan, I. Prodan, S. Olaru, F. Stoican, and S.-I. Niculescu, "Mixed-integer programming in motion planning," *Annual Reviews in Control*, 2020.
 - [12] A. Gray, Y. Gao, T. Lin, J. K. Hedrick, H. E. Tseng, and F. Borrelli, "Predictive control for agile semi-autonomous ground vehicles using motion primitives," in *2012 American Control Conference (ACC)*. IEEE, 2012, pp. 4239–4244.
 - [13] E. Frazzoli, "Maneuver-based motion planning and coordination for multiple UAVs," in *Proceedings. The 21st Digital Avionics Systems Conference*, vol. 2. IEEE, 2002, pp. 8D3–8D3.
 - [14] J. Eilbrecht and O. Stursberg, "Optimization-Based Maneuver Automata for Cooperative Trajectory Planning of Autonomous Vehicles," in *2018 European Control Conference (ECC)*. IEEE, Jun. 2018, pp. 82–88. [Online]. Available: <https://ieeexplore.ieee.org/document/8550422/>
 - [15] J. Alonso-Mora, P. Beardsley, and R. Siegwart, "Cooperative collision avoidance for nonholonomic robots," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 404–420, 2018.
 - [16] N. Richards, M. Sharma, and D. Ward, "A hybrid A* automaton approach to on-line path planning with obstacle avoidance," in *AIAA 1st Intelligent Systems Technical Conference*, 2004, p. 6229.
 - [17] B. Cohen, M. Phillips, and M. Likhachev, "Planning Single-arm Manipulations with n-Arm Robots," in *Robotics: Science and Systems X*. Robotics: Science and Systems Foundation, Jul. 2014. [Online]. Available: <http://www.roboticsproceedings.org/rss10/p33.pdf>
 - [18] C. M. Dellin and S. S. Srinivasa, "A Unifying Formalism for Shortest Path Problems with Expensive Edge Evaluations via Lazy Best-First Search over Paths with Edge Selectors," in *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS)*, 2016, p. 9.
 - [19] A. Mandalika, O. Salzman, and S. Srinivasa, "Lazy Receding Horizon A* for Efficient Path Planning in Graphs with Expensive-to-Evaluate Edges," in *International Conference on Automated Planning and Scheduling*, 2018, p. 9.
 - [20] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," *Ann Arbor*, vol. 1001, no. 48105, pp. 18–80, 2008.
 - [21] Z. Ajanovic, B. Lacevic, B. Shyrokau, M. Stolz, and M. Horn, "Search-Based Optimal Motion Planning for Automated Driving," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2018, pp. 4523–4530, ISSN: 2153-0866.
 - [22] B. Mettler and Z. Kong, "Receding horizon trajectory optimization with a finite-state value function approximation," in *American Control Conference*, Seattle, WA, USA, 2008.
 - [23] O. Andersson, O. Ljungqvist, M. Tiger, D. Axehill, and F. Heintz, "Receding-Horizon Lattice-Based Motion Planning with Dynamic Obstacle Avoidance," in *2018 IEEE Conference on Decision and Control (CDC)*. Miami Beach, FL: IEEE, Dec. 2018, pp. 4467–4474. [Online]. Available: <https://ieeexplore.ieee.org/document/8618964/>
 - [24] D. Schleich and S. Behnke, "Search-based planning of dynamic mav trajectories using local multiresolution state lattices," *arXiv preprint arXiv:2103.14607*, 2021.
 - [25] K. Kurzer, C. Zhou, and J. M. Zöllner, "Decentralized cooperative planning for automated vehicles with hierarchical monte carlo tree search," in *2018 IEEE intelligent vehicles symposium (IV)*. IEEE, 2018, pp. 529–536.
 - [26] C. Frese, J. Beyerer, and M. Huber, "Cooperative motion planning using branch and bound methods," in *Proc. of the 2009 Joint Workshop of Fraunhofer IOSB and Institute for Anthropomatics, Vision and Fusion Laboratory (J. Beyerer and M. Huber, eds.)*, no. IES-2009-13, 2010, pp. 187–201.
 - [27] C. Frese and J. Beyerer, "Planning cooperative motions of cognitive automobiles using tree search algorithms," in *Annual Conference on Artificial Intelligence*. Springer, 2010, pp. 91–98.
 - [28] E. Frazzoli, M. A. Dahleh, and E. Feron, "Robust hybrid control for autonomous vehicle motion planning," in *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No.00CH37187)*, vol. 1, 2000, pp. 821–826 vol.1.
 - [29] K. Flaßkamp, S. Ober-Blöbaum, and K. Worthmann, "Symmetry and motion primitives in model predictive control," *Mathematics of Control, Signals, and Systems*, vol. 31, no. 4, pp. 455–485, 2019.
 - [30] K. Flaßkamp, S. Ober-Blöbaum, and M. Kobilarov, "Solving optimal control problems by exploiting inherent dynamical systems structures," *Journal of Nonlinear Science*, vol. 22, pp. 599–629, 2012.
 - [31] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.
 - [32] R. Rajamani, *Vehicle dynamics and control*. Springer Science & Business Media, 2011.
 - [33] M. Althoff, M. Koschi, and S. Manzi, "Commonroad: Composible benchmarks for motion planning on roads," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017.
 - [34] B. Alrifae, "Networked model predictive control for vehicle collision avoidance," Ph.D. dissertation, RWTH Aachen University, 2017.
 - [35] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*, 2nd ed. MIT Press, 2017.
 - [36] S. Koenig and X. Sun, "Comparing real-time and incremental heuristic search for real-time situated agents," *Autonomous Agents and Multi-Agent Systems*, vol. 18, no. 3, pp. 313–341, 2009.
 - [37] B. Alrifae, F.-J. Heßeler, and D. Abel, "Coordinated non-cooperative distributed model predictive control for decoupled systems using graphs," *IFAC-PapersOnLine*, vol. 49, no. 22, pp. 216–221, 2016.