Energy-Efficient Motion Planning for Autonomous Vehicles Using Uppaal Stratego

Chapter · July 2024

DOI: 10.1007/978-3-031-64626-3_21

Muhammad Naeem
Aalborg University
9 PUBLICATIONS **23** CITATIONS
SEE PROFILE

Rong Gu
Mälardalen University
18 PUBLICATIONS **108** CITATIONS
SEE PROFILE

Cristina Cerschi Seceleanu
Mälardalen University
159 PUBLICATIONS **990** CITATIONS
SEE PROFILE

Michele Albano
Aalborg University
128 PUBLICATIONS **908** CITATIONS
SEE PROFILE

Energy-Efficient Motion Planning for Autonomous Vehicles Using Uppaal Stratego [⋆]

Muhammad Naeem[1,2], Rong Gu[2], Cristina Seceleanu[2], Kim Guldstrand Larsen[1], Brian Nielsen[1], and Michele Albano[1]

[1] Department of Computer Science, Aalborg University, Aalborg, Denmark
{mnaeem, kgl, bnielsen, mialb}@cs.aau.dk
[2] Division of Networked and Embedded Systems, Mälardalen University, Sweden
{muhammad.naeem, rong.gu, cristina.seceleanu}@mdu.se

Abstract. Energy-efficient motion planning for autonomous battery-powered vehicles is crucial to increase safety and efficiency by avoiding frequent battery recharge. This paper proposes algorithms for synthesizing energy- and time-efficient motion plans for battery-powered autonomous vehicles. We use stochastic hybrid games to model an appropriate abstraction of the autonomous vehicle and the environment. Based on the model, we synthesize energy- and time-efficient motion plans using Q-learning in Uppaal Stratego. Via experiments, we show that pure Q-learning is insufficient when the problem becomes complex, e.g., Motion Planning (MOP) in large environments. To address this issue, we propose Concatenated Motion Planning (CoMOP), which divides the environment into several regions, synthesizes a motion plan in each region and concatenates the local plans into an entire motion plan for the whole environment. CoMOP enhances the applicability of Q-learning to large and complex environments, reduces synthesis time, and provides efficient navigation and precise motion plans. We conduct experiments with our approaches in an industrial use case. The results show that CoMOP outperforms MOP regarding synthesis time and the ability to deal with complex models. Moreover, we compare the energy- and time-efficient strategies and visualize their differences on different terrains.

1 Introduction

Autonomous vehicles are equipped with advanced sensors, cameras, and communication devices, which enable them to perceive the environment and perform various tasks without human intervention. In particular, using autonomous vehicles in construction sites can lead to increased productivity, safety, and reduced costs. Ensuring energy efficiency is a critical challenge in autonomous vehicle

[⋆] The research was supported by the ERC Advanced Grant LASSO and Villum Investigator Grant S4OS, and the Swedish Knowledge Foundation via the synergy project ACICS – Assured Cloud Platforms for Industrial Cyber-Physical Systems, grant nr. 20190038, and the profile project DPAC - Dependable Platform for Autonomous Systems and Control, grant nr: 20150022.

design, especially for battery-powered ones, where limited energy constrains operating range and duration. Therefore, an efficient motion-planning algorithm is crucial for real-world deployment. However, energy-efficient motion planning is not trivial. Intuitively, a time-efficient motion plan should be energy-efficient, which means a path-finding algorithm like A* [18] would suffice. However, our experiments show that these two kinds of motion plans are not necessarily the same on different terrains and that the energy efficiency of driving depends on the terrain type. For instance, on slopes, the fastest plan can be energy-consuming, while a curved path may cost less energy.

In recent years, reinforcement learning [20] gained popularity for motion planning [1][3]. Despite the simple idea of allowing the machine to learn and accumulate rewards from its actions, these algorithms are data-hungry, and tuning a reward function can be challenging and time-consuming, as our experiments show; synthesizing an energy-efficient motion plan with Q-learning [21] in a 20×20 environment with 2 obstacles takes nearly 2 hours. This motivates us to develop a new method that is able to solve real-world motion-planning problems in a reasonable time, and provide a systematic way of evaluating the learning results, such as the probability of finishing a job before the energy consumption exceeds a threshold.

In this paper, we propose a method (backed by experiments) to generate *energy-efficient motion planning* for battery-powered autonomous vehicles. First, we model the autonomous vehicles and the environment as a *Stochastic Hybrid Game* (SHG) [12]. By using this modeling language, we can construct the control logic as timed games (TG) [2], and encode the vehicle's kinematics and energy consumption as ordinary differential equations (ODE) in the TG. Next, we design *methods* for solving the SHG by using Q-learning, that is, synthesizing time- and energy-efficient motion plans of the model. We construct the model and conduct the synthesis in UPPAAL Stratego [6], which is a toolset of modeling, simulation, verification, and synthesis of timed games. As aforementioned, motion planning by Q-learning is restricted by the scale of the problem, such as the environment size and obstacle numbers. To scale the synthesis, we propose a *novel algorithm*, called *concatenation-based motion planning* (CoMOP), which splits the environment into sub-components, synthesizes a motion plan in each of the sub-components, concatenates the motion plans into a complete one, and verifies the complete plan by using statistical model checking [19]. When concatenating two motion plans, we must ensure the *validity* and *automation* of the concatenation, that is, the final set of states of a partial motion plan is equivalent to the initial set of states of its following motion plan, and the work must be done automatically by our tool, such that CoMOP does not break the autonomy of the system. To meet this ambition, we realize the concatenation via an external library that is invoked by UPPAAL STRATEGO, in which assertions are used for checking the validity of the concatenation. We evaluate CoMOP on an industrial case study, that is, an autonomous quarry. We design several groups of experiments that have different sizes of the quarry and obstacles, different numbers of obstacles, and various terrains. Via the experiments, we show the significant difference in performance between CoMOP and MOP (motion planning without concatenation), and the difference between time- and energy-efficient trajectories on various terrains. In summary, our contributions are as follows:

1. This paper proposes a novel *concatenation-based motion planning* (CoMOP) method, using reinforcement learning. We show that CoMOP is much more efficient than MOP when the scale of the problem is large.
2. We demonstrate the use of statistical model checking (SMC) in learning motion plans. Via experiments, we show the *qualitative evaluation of the synthesized motion plans* by using SMC, which exposes the correct rate of the resulting motion plan before testing it on an actual system implementation.
3. We demonstrate the difference between *time- and energy-efficient motion plans*. Our large-scale evaluation of CoMOP in an industrial case study shows the efficiency of obtaining these two kinds of motion plans.

The rest of the paper is structured as follows: We review the previous work in Section 2. In Section 3, we provide an introduction to Reinforcement Learning and Uppaal Stratego. Section 4 elaborates on the case study and presents the vehicle's kinematic and energy model. In Section 5, we describe the modeling of the case study in Uppaal Stratego and strategy synthesis. Furthermore, Section 6 outlines the motion planning concatenated motion planning. In Section 7, we conduct a thorough examination of the energy- and time-efficient strategies for the case study. Finally, we conclude our research results in Section 8.

2 Related Work

In recent years, there has been a growing interest in developing energy-efficient motion planning algorithms for autonomous vehicles. Energy consumption can be minimized by a parametric curve fit [14], by applying machine learning [16, 17], or by employing physics-based models of energy consumption [5, 10]. The energy prediction method used by Quann et al. [16, 17] abstracts from kinematic considerations such as velocity and acceleration. Physics-based modeling [10] that accounts for these assume mostly flat terrains. In Wallace et al. [5], the authors consider elevation and a detailed kinematic model. However, their model is analytical, being validated via simulation. In our work, we model both vehicle kinematics, although not as detailed as in the work of Wallace et al. [5], and we consider environments that include slopes and employ a combination of learning and model-checking techniques for the automatic synthesis of energy-efficient motion plans.

Compositional reinforcement learning is a promising approach for training policies to perform complex long-horizon tasks. Jothimurugan et al. [11] propose a novel framework based on two reinforcement learning algorithms by formulating the learning problem of choosing what tasks to perform from an existing set, as an adversarial reinforcement learning problem. Although close in spirit, our approach does not focus on robust task choices w.r.t. functionality, but rather w.r.t. optimizing time- and energy usage by combining task learning with assertion-based space decomposition within stochastic hybrid games.

As formal methods are capable of providing rigorous analysis for complex systems, a combination of formal methods and learning in the field of robotics has been a trend. Yang et al. [23] propose guided and safe reinforcement learning, and Pek et al. [15] employ Spatio-Temporal Logic in monitoring and planning complex robotic tasks. Mariano et al. [8] proposed an optimal motion planning approach by integrating cell-mapping techniques and reinforcement learning to improve trajectory generation. Xu et al. [22] present an autonomous vehicle motion planning and control system that integrates a unified behavior planner and robust trajectory generation. Our work approaches the motion planning of autonomous systems from another angle: (i) a compositional learning technique, to cope with complex environments, and (ii) quantitative evaluation of the results by using statistical model checking.

3 Preliminaries

In this section, we overview reinforcement learning, stochastic hybrid games, and Uppaal Stratego, which form the paper's necessary background information.

3.1 Reinforcement Learning

Reinforcement learning (RL) [20] is a machine-learning method that trains a machine's behaviour by letting it interact with the environment, collecting scores of the machine's actions at different states, and calculating the state-action pairs' values. *Q-learning* [21] is a classic RL algorithm that uses the Bellman-optimal equation to calculate the values of state-action pairs, shown as follows.

$$q^*(s,a) = \mathbb{E}[R(s,a) + \gamma \max_{a'} q^*(s',a')], \tag{1}$$

where $q^*(s,a)$ represents the expected value of performing action a at state s, $\mathbb{E}$ denotes the expected value function, $R(s,a)$ is the reward returned from the environment by taking a at s, $\gamma \in [0,1]$ is a discounting value indicating how much the future reward is evaluated in the calculation, s' is the next state originating from s by taking a, and $\max_{a'} q^*(s',a')$ is the maximum reward that can be obtained by any possible next state-action pair (s',a').

3.2 Uppaal Stratego

Uppaal Stratego allows for the modeling of Stochastic Hybrid Games (SHG) and facilitates the synthesis of nearly optimal strategies through Reinforcement Learning (RL). SHG are two-player games that are played on SHG [4][12]. We recall the formal definition of SHG as follows:

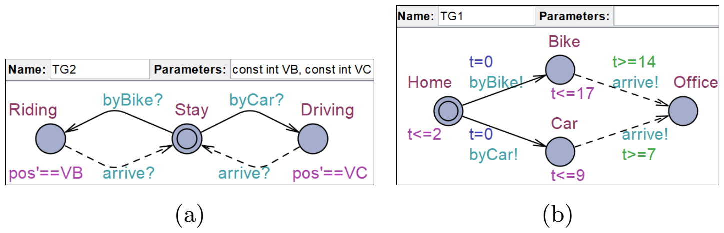


Fig. 1: An example of NSHG templates in UPPAAL STRATEGO

Definition 1. *A SHG $\mathcal{G} = \langle L, l_0, C, \Sigma, E, Inv, F_l, \mu, \gamma, l_f \rangle$ is a tuple, where L is a finite set of locations, l_0 is the initial location, C is a finite set of non-negative real-valued variables (including clocks that evolve at rate 1); $\Sigma = \Sigma_c \cup \Sigma_u$ is a finite set of actions (Σ_c denotes the set of controllable actions, and Σ_u denotes the uncontrollable ones), $E \subseteq L \times \mathcal{B}(C) \times \Sigma \times 2^C \times L$ is a finite set of edges, where $\mathcal{B}(C)$ is the set of guards over C, that is, conjunctive formulas of constraints of the form $c_1 \bowtie n$ or $c_1 - c_2 \bowtie n$, where $c_1, c_2 \in C$, $n \in \mathbb{N}$, $\bowtie \in \{<, \leq, =, \geq, >\}$, $Inv : L \to \mathcal{B}(C)$ is a partial function assigning invariants to locations, for each $l \in L$, $F_l : \mathbb{R}_{>0} \times \mathbb{R}^C \to \mathbb{R}^C$ is the flow-function that describes the evolution of the continuous variables over time, typically represented by Ordinary Differential Equations (ODE_s), μ is a family of delay density functions (can be uniform or exponential distributions), $\mu_l : \mathbb{R}_{\geq 0} \times L \to \mathbb{R}_{\geq 0}$ that determine the time point for the next discrete jump, for each location $l \in L$, respectively, γ is the set of probability functions $\gamma_l : L \times L \to \mathbb{R}_{\geq 0}$ determining the next location, and l_f is the set of goal locations.*

Intuitively, *controllable* actions are performed by the system, e.g., autonomous vehicles in this paper, and *uncontrollable actions* are performed by the environment. The semantics of the SHG is defined over a timed transition system, whose states are pairs $s = (l, v) \in L \times \mathbb{R}^C$, with $v \models Inv(l)$, and transitions are defined as: (i) delay transitions $(l, v) \xrightarrow{d} (l, v')$ with $d \in \mathbb{R}_{\geq 0}$ and $v' = v + d$ such that $v' \models Inv(l)$, and (ii) discrete transitions $(l, v) \xrightarrow{a} (l', v')$ if there is an edge (l, g, a, Y, l'), where $a \in \Sigma$ and $Y \subseteq C$, such that $v \models g$, $v \models Inv(l)$, $v' = v[Y]$, and $v[Y]$ is a clock valuation by assigning 0 to $x \in Y$ such that $v' \models Inv(l')$. We write $\rho = (l, v) \rightsquigarrow (l', v')$, where ρ is a finite sequence of delay and discrete transitions from (l, v) to (l', v').

A solution of an SHG is a *winning strategy*, which is a function $\pi : \rho \to \Sigma_c' \subseteq \Sigma_c$. As the strategies in this paper are *memoryless*, our winning strategy can be formulated as $\pi : s \to \Sigma_c' \subseteq \Sigma_c$, where $s = last(\rho)$ is the last state of ρ. Intuitively, strategy π indicates the "system" player what controllable actions to choose at each state, such that it can eventually reach the desired goal and avoid obstacles with the minimum cost, where cost can be time and energy consumption modelled by the ODEs in Definition 1.

UPPAAL STRATEGO [6] is a powerful tool for modeling, strategy synthesis, and (statistical) model checking of SHG, in which SHG can be composed in parallel as a *network* of SHG (NSHG) synchronized via *channels*. Fig. 1 depicts SHG templates that can be instantiated into NSHG models by assigning concrete values to the parameters of the template, such as `VB` in TG2 (Fig. 1a). Blue circles are *locations* that are connected by directional *edges*. Double-circled locations are the *initial* locations (e.g. `Stay`). In Uppaal Stratego, there are two special kinds of locations: *urgent* locations (i.e., encircled "u"), and *committed* locations (i.e., encircled "c"). A Uppaal Stratego SHG requires that time does not elapse in those two kinds of locations, with committed locations being even stricter, that is, the next edge to be traversed must start from one of them. On the edges, there are assignments resetting clocks (e.g., `t=0`) and updating data variables, guards (e.g., `t>=14`), and synchronization channels (e.g., `arrive!` and `arrive?`). On location `Bike` in Fig. 1b, an invariant `t<=17` means that clock `t` must never exceed 17. Derivatives of clocks can be specified by Ordinary Differential Equations (ODE), such as `pos'==VB` in Fig. 1a.

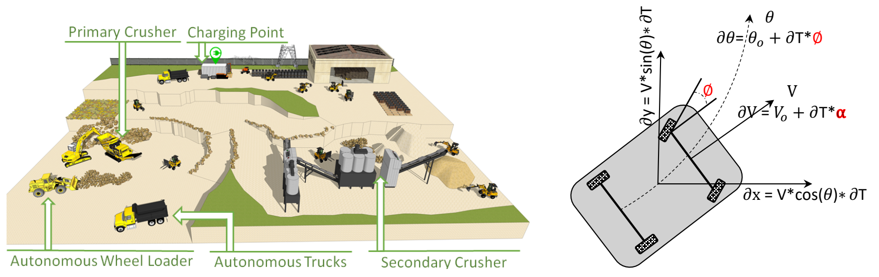
In Uppaal Stratego, the SHG have a stochastic interpretation based on: (i) the probabilistic choices between multiple enabled discrete transitions (that assume a uniform distribution by default), and (ii) the non-deterministic time delays that are refined based on probability distributions, either uniform distributions for time-bounded delays, or user-defined exponential distributions for unbounded delays. In this paper, we use only the default uniform distributions for discrete transitions and time-bounded delays. For instance, Fig. 1 models a driver taking different transportation tools to visit his office. In Uppaal Stratego, one can synthesize a strategy that guides the system player (e.g., the driver) to reach its goal (e.g., the office) by using the following query:

$$\texttt{strategy s} = \texttt{minE(cost)}\,[\leq \texttt{T}]\,\{\texttt{ExpList1}\} \;\rightarrow\{\texttt{ExpList2}\} :<> \texttt{goal} \quad (2)$$

where `minE(exp)` means that strategy `s` is to minimize the value of expression `cost` within T time-units or when the goal predicate becomes true. `ExpList1` and `ExpList2` are lists of observable state expressions used in Q-learning [21]. `ExpList1` are discrete variables and `ExpList2` are continuous variables. Uppaal Stratego simulates the model and samples traces for the learning algorithm, with `<> goal` specifying criteria for trace selection. The learning algorithm receives controllable actions in these traces, representing partial observation of the state space [9]. Additionally, Uppaal Stratego supports calling external C-libraries, an important feature used in this work.

4 Use Case and Models

In this paper, we demonstrate the applicability of our method on an industrial case study provided by Volvo Construction Equipment, Sweden. The use case focuses on reducing the energy consumption of autonomous wheel loaders that transport materials within a construction site. An example is shown in Fig 2a, where the autonomous wheel loader needs to transport stones to a primary crusher to crush them into smaller pieces. After that, it transports the crashed



(a) Working environment (b) Kinematic vehicle model

Fig. 2: An example of an autonomous vehicle

stones to the secondary crusher, which is the destination of the stones. We aim to develop time- and energy-efficient motion planning for battery-powered autonomous vehicles to extend the vehicle's operating time on a single charge.

4.1 Vehicle's kinematic Model

In Fig. 2b, one can see a simple kinematic model that illustrates the configuration of a vehicle using seven parameters: (x, y, ϕ, α, V, θ, T), where x and y model the vehicle's current position in a 2D environment, ϕ the steering angle, α the acceleration, V the velocity, θ the vehicle's orientation, and T the time.

The vehicle's trajectory must satisfy a reach-avoid requirement, that is, eventually reaching the destination while always avoiding obstacles. It can be seen that by controlling acceleration α and steering angle ϕ, one can determine the vehicle's position, that is, the values of X and Y. Therefore, our motion planning is about controlling parameters α and ϕ. In Section 4.2, we introduce how energy optimization is considered in our motion planning.

4.2 Energy Model

The tractive power model described by Lascurain et al. [13] and Gao et al. [7] is widely used to calculate the power required by a vehicle powertrain. The tractive power is the amount of energy required to overcome the vehicle's resistance to motion, and it considers various factors that affect the vehicle's performance, including the environment. The following equation can be used to calculate the vehicles' tractive power:

$$W_{tract} = m \cdot V \cdot \frac{dV}{dt} + m \cdot g \cdot C_{rr} \cdot V + m \cdot g \cdot V \cdot sin(\Theta) \tag{3}$$

where W_{tract} is the tractive power, m is the vehicle's mass, V is the velocity, g is the gravitational acceleration constant, C_{rr} is a coefficient of rolling resistance,

Table 1: Energy Parameters

Name	*Value*	*Name*	*Value*	*Name*	*Value*	*Name*	*Value*
m	$7-22Ton$	C_{rr}	$2-3\%$	η_{mot}	0.88	Θ	12%
W_{acc}	3.75	η_{batt}	0.98	η_{wh}	0.99	η_{fd}	0.98

Θ is the road's slope. Parameter values obtained from the Volvo project are shown in Table 1, which contains other parameters that we introduce later.

In this study, we use the energy model presented in the literature [13]. The authors propose a model for estimating the electric power output of an electric vehicle (EV), which considers the efficiencies of the electric components, such as the motor and battery, and the related drive-train components, such as the final drive and wheel efficiencies shown in Eq 4. The equations for calculating the driving power and braking power of an EV are as follows.

$$\begin{aligned} &Driving\ Power : W_{drive} = W_{acc} + \frac{W_{tract}}{\eta_{wh} \cdot \eta_{fd} \cdot \eta_{mot} \cdot \eta_{batt}} \\ &Braking\ Power : W_{Braking} = W_{acc} \\ &TotalPower : W = W_{drive} + W_{Braking} \end{aligned} \tag{4}$$

where W_{acc} is a conventional vehicle's accessory load (kW), η_{wh} is the wheel efficiency, η_{fd} is the final drive efficiency, η_{mot} is the motor efficiency, and η_{batt} is the battery efficiency. Table 1 depicts the coefficient values used in this paper.

4.3 Abstract View of Vehicle Model

We develop a framework for simulating a vehicle's movement to find a path to the destination while avoiding obstacles, which includes a dynamic decision-making process based on the Q-learning algorithm. Specifically, we construct an SHG model of an autonomous wheel loader in UPPAAL STRATEGO, which can navigate along the x and y axes. The vehicle's movement is controlled by selecting the right steering angle (ϕ) and acceleration (α), chosen from a strategy synthesized by the Q-learning algorithm. To illustrate our SHG model in an abstract way, Fig. 3 depicts a graphical representation of the vehicle model.

The decision-making process of the vehicle is controlled by the Q-learning block, which determines the appropriateness of the chosen ϕ and α values for each time unit during the vehicle's movement, that is, the evolution of x and y. Additionally, when a large steering angle (ϕ) is chosen, and the vehicle's current velocity exceeds a predefined safety threshold or comfortableness at that specific ϕ, the model enters the `Braking` block. In this block, the vehicle can move to the `Drive` block when the velocity is reduced to a required level or return to its decision-making block to reconsider whether the destination is close. Alternatively, the model also enters the `Braking` block as it approaches the destination, where it keeps braking until the vehicle fully arrives and stops. Furthermore, if

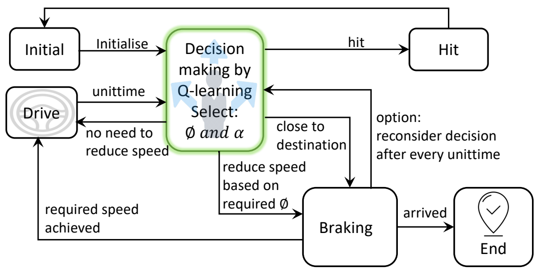

Initial
Initialise
Decision making by Q-learning Select: $\emptyset$ and α
hit
Hit
Drive
unittime
no need to reduce speed
close to destination
option: reconsider decision after every unittime
reduce speed based on required $\emptyset$
required speed achieved
Braking
arrived
End

Fig. 3: Abstract View of Vehicle Model in UPPAAL STRATEGO

the vehicle hits an obstacle or crosses a boundary of the environment, it enters the `Hit` block to indicate a collision or boundary violation.

Now, we define the problem of motion planning as follows:

Definition 2 (Motion Planning). *Given an autonomous vehicle whose kinematics and energy consumption are described by ordinary differential equations, e.g., equations (2b) and (4), respectively, a motion plan for the vehicle is a function* $(\theta, V) = f(x, y, t)$, *where* $\theta, v, x,$ *and* y *are the state variables of the vehicle in equation (2b),* $t \in [0, T]$ *is a time point within* 0 *and* T, *and* $T \in \mathbb{R}^+$ *is the time of reaching the destination. The goal of motion planning is to synthesize motion plans that are time-efficient, i.e.,* T *is minimal, or energy-efficient, i.e., total power* W *is minimal.*

5 Vehicle Model In UPPAAL STRATEGO

This section demonstrates how we use UPPAAL STRATEGO to model the vehicle's actions. Fig. 4 depicts an SHG model that implements the abstract vehicle model in Fig. 3. UPPAAL STRATEGO handles this motion planning problem as an SHG. The model contains two types of elements (locations and edges), and the edges are further divided into controllable (solid line) and uncontrollable ones (dotted line). The locations of the model represent the system's processes like *Braking* and *Driving*, and edges are the actions controlling these processes. Table 2 presents some important variables used in the model.

Locations `Braking` and `Drive` are the main locations in our model. All other locations are used to help in taking different actions between or on these locations. The edge associated with the `Initial` location invokes the `Initialise()` function to initialize variables such as the initial position and angle. The edge from `T7` is controllable and includes a statement of *selections*, which allows the vehicle model to select an acceleration from a defined range from 0 to 10. Within the `Drive` location, continuous variables evolve according to the differential equations defined in the model, e.g., `agl'==ar` define the vehicle's orientation.

Now, we describe the vehicle's movement defined at the `Drive` location. First, the coordinates of the vehicle, x, and y, begin to change in the direction specified by `agl` (the orientation of the vehicle) while consuming energy. Second, the loops

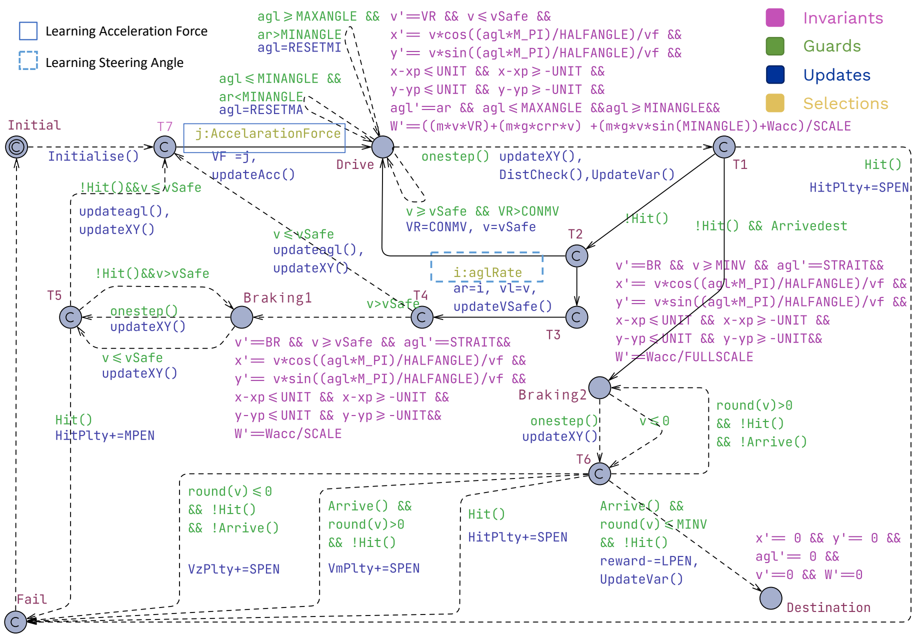


Fig. 4: Vehicle's SHG model in Uppaal Stratego

in the Drive location control the value of agl between 0 and 360 and restrict the acceleration when it reaches the maximum safe speed (vSafe). Third, the model should leave the Drive location and move to T1 when the x or y value evolves a step on the map. The duration spent in a location is determined by the location's invariants and the guard on the outgoing edge. Therefore, the invariants in location Drive, such as x-xp≤UNIT, and the guard onestep() regulate the duration in Drive to be *one step*. The length of the step is a constant defined in the model.

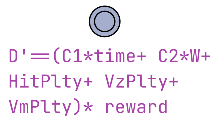


Fig. 5: Reward function

The model moves from location T1 to Fail, if an obstacle or boundary is encountered. Otherwise, it offers two controllable actions: continuing to drive or moving to the Braking2 location to check if the destination has been reached. The edge from T1 to Braking2 is guarded by the Arrivedest variable, enabling it only when the vehicle is close to the destination, thus avoiding unnecessary execution towards the final braking.

Two controllable actions towards the Drive location are changing the steering angle or moving to Drive with the same steering angle. En route to T3, another controllable edge involves selecting a steering angle from -45 to 45. The model assumes instantaneous steering angle changes. Updating the angle requires adjusting the maximum safe speed (vSafe) based on the new angle. If the current velocity exceeds the new maximum safe velocity, the model transi-

Table 2: Vehicle's variables used in the model.

Variable	Type	Description
`X,Y`	Clock	Position on the x and y axes.
`V`	Clock	Velocity.
`agl`	Clock	Orientation, denoted as θ in Figure 2b.
`W`	Clock	Energy consumption.
`ar`	Int	Steering angle, labelled as ϕ in Figure 2b.
`VR`	Int	Acceleration, labelled as α in Figure 2b.
`vSafe`	Real	Safety threshold of velocity.

tions to `Braking1` to reduce speed before moving to `Drive`; otherwise, it directly proceeds to `Drive`.

Penalties in our model training include `HitPlty` for reaching the `hit` location, `VmPlty` for arriving with non-zero vehicle speed, and `VzPlty` for stopping without reaching the destination. Conversely, a reward is granted for the successful arrival of the destination. The model learns to minimize penalties and maximize rewards, refining strategies through these feedback mechanisms.

The automaton shown in Figure 5 models the objective function. `D` is a continuous variable, and its derivative depends on a set of variables, that is, `Time` , `W`, `HitPlty`, `VzPlty`, `VmPlty`, and `reward`. The variable `time` is used for the time-efficient strategy, and `W` is used for the energy-efficient one. `C1` and `C2` are coefficients for balancing `time` and `W` in developing strategies. Variable `reward` is updated with a negative value when it reaches the goal and starts reducing the objective function. We run a query as Query 2 in Uppaal Stratego, which minimizes this objective function to generate an efficient strategy.

5.1 Strategy Synthesis

This section details strategy queries (shown in equation 5) for motion-plan synthesis, following the structure described in Section 3.2. Here, `D` is the objective function, and `minE` means the goal of synthesis is to minimize `D`. The query passes the set of discrete and continuous variables (described in section 3.2) to the Q-learning algorithm embedded in Uppaal Stratego to develop a strategy.

$$\begin{gathered}\texttt{strategy GoFast} = \texttt{minE(D)}\,[\texttt{time} \le \texttt{T}]\,\{\texttt{location}, \texttt{ar}, \texttt{VR}, \texttt{VzPlty}, \texttt{VmPlty}, \\ \texttt{HitPlty}, \texttt{reward}\} \rightarrow \{\texttt{agl},\ \texttt{x},\ \texttt{y},\ \texttt{v},\ \texttt{W}\} : <> \texttt{shg.Destination} \,\&\&\, \texttt{time} <= \texttt{T}\end{gathered} \tag{5}$$

$$\texttt{simulate}\ [\le \texttt{T}; 1]\{\texttt{x}, \texttt{y}\}\ \texttt{under GoFast} \tag{6}$$

After constructing the strategy, simulation queries (shown in equation 6), are employed to simulate the model under the synthesized strategy. The query prints the values of `x` and `y`, representing the vehicle's moving trajectory.

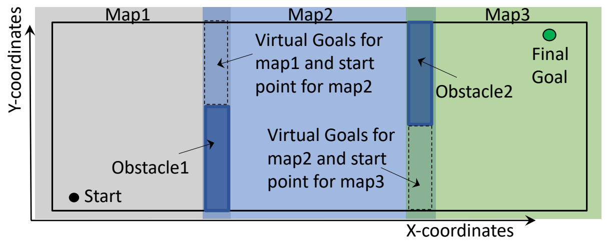

Map1
Map2
Map3
Y-coordinates
X-coordinates
Virtual Goals for map1 and start point for map2
Obstacle2
Final Goal
Virtual Goals for map2 and start point for map3
Obstacle1
Start

Fig. 6: Abstract view of a map and sub-maps

6 Assertion-Based Concatenated Motion Planning

This section introduces our new method *CoMOP*. First, we implement the vehicle model in Uppaal Stratego. Next, we use two methods to synthesize motion plans (a.k.a., strategies in Uppaal Stratego) to reach the destination.
Method 1: MOtion Planning (MOP). MOP considers the entire environment (a.k.a., map) as a single entity and learns a holistic motion plan from the starting point to the destination while taking into account time or energy consumption.
Method 2: Concatenated MOtion Planning (CoMOP). CoMOP also aims to synthesize efficient motion planning. The difference is that CoMOP divides the map into sub-maps. Each sub-map represents a distinct area of the environment. Fig. 6 depicts the subdivision of a map into sub-maps. As shown in the figure, overlapped regions exist between two sub-maps, and such regions serve as virtual destinations for one sub-map, and starting regions for the next, respectively.

To synthesize a motion-planning strategy, we use a **reverse calculation** approach, starting from *Map3*, determining an efficient starting point within the overlapped region for a precise virtual goal point. We also analyze the suitable vehicle orientation for the forward path strategy in *Map3*. A similar approach is employed for *Map2*, ensuring the model concludes at the starting points determined in *Map3*. This sequential process establishes a motion-planning strategy considering both sub-maps with consistent starting points and orientations.

In each sub-map, we employ a sequential execution approach for forward motion planning. Illustrated in Fig. 7, we develop a strategy for a sub-map, followed by simulating the vehicle model and obtaining variable trajectories (`x`

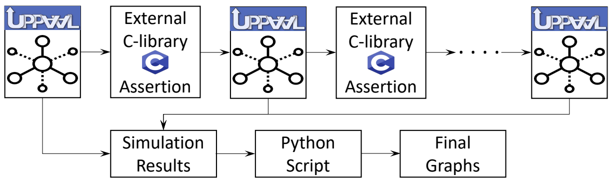
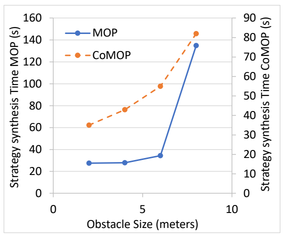
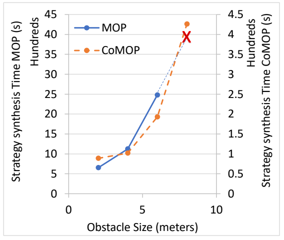


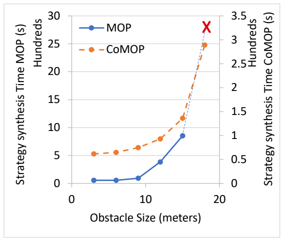
Fig. 7: Cascaded execution of model for CoMOP simulation



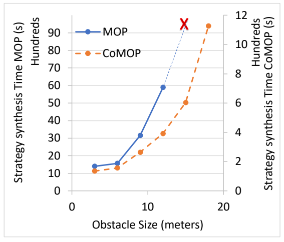
(a) 10x10 & 1 Obstacle



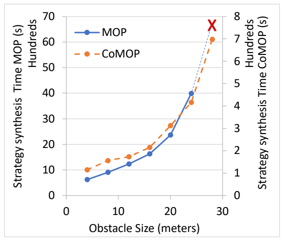
(b) 10x10 & 2 Obstacles



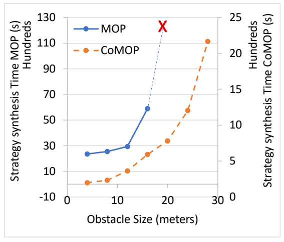
(c) 20x20 & 1 Obstacle



(d) 20x20 & 2 Obstacles



(e) 30x30 & 1 Obstacle



(f) 30x30 & 2 Obstacles

Fig. 8: Strategy Synthesis time with different map sizes, obstacle heights, and number of obstacles

`, y, agl, W, V, T`). The final values are then forwarded to the next strategy using an external C library, ensuring a continuous information flow between consecutive sub-maps. **Assertions** in the external C-library guarantee smooth transitions between the **concatenated** strategies, validating that final variable values in one sub-map are within valid regions for the subsequent sub-map.

7 Experimental Evaluation

In this section, we introduce the experiments that we have conducted to evaluate the performance of our method. Next, we explore and answer the following research questions according to the experimental results.

- **RQ1**: Considering map size, slope, obstacle size, and obstacle number, what is their impact on the performance of our synthesis methods?
- **RQ2**: Does CoMOP outperform MOP?
- **RQ3**: Is the time-efficient motion plan significantly different from the energy-efficient motion plan of the same problem?

7.1 RQ1: Impact Factors on the Synthesis Methods' Performance

Based on the industrial use case, we explore the influential factors in this series of experiments by systematically varying three key parameters: map size, number of obstacles, and obstacle size. The experimental design encompasses three different

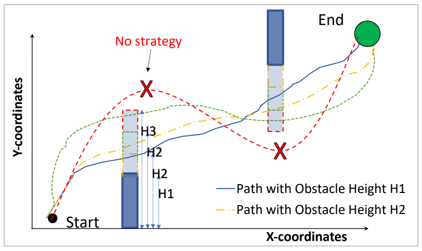


Fig. 9: Path trajectories of MOP

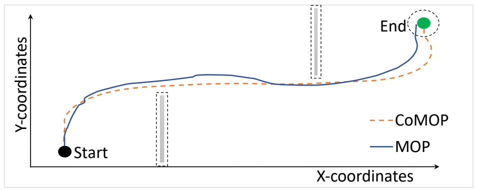
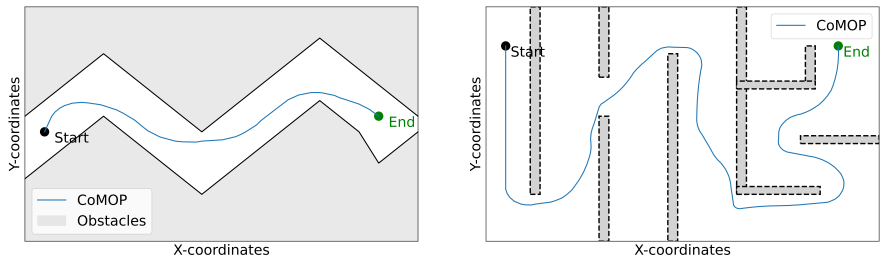


Fig. 10: Comparison of successful strategies developed by MOP and CoMOP

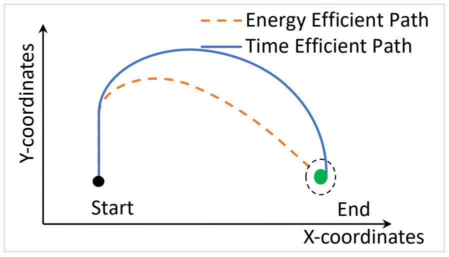
map sizes, namely 10, 20, and 30 units, with each map configuration simulated under two obstacle conditions: one obstacle and two obstacles. Additionally, we examine the effect of obstacle size. Note that the unit of a map is not specified because one step in a map (i.e., the guard `onestep()` in Fig.4) represents the granularity of decision making.

The results in Fig. 8 offer insights into dominant factors impacting learning time for motion plan synthesis. By analyzing the results, we can discern the influence of each factor on the simulation outcomes. Increasing the complexity in terms of obstacle size, map size, and the number of obstacles generally resulted in longer simulation times for both the MOP and CoMOP models. Comparatively, CoMOP needs less time to find a strategy, except in the experiment shown in Fig. 8a, having a small map with a small obstacle. However, it is essential to note that in certain experiments, the MOP algorithm encounters difficulties in finding a strategy. For example, in the experiment conducted on a 20-size map with a single obstacle of size 18, MOP fails to generate any result (see the red cross in Fig. 8c). CoMOP, on the other hand, demonstrates its ability to address complex motion-planning problems by employing the sub-map division approach.

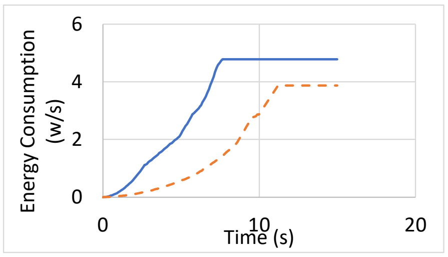


(a) (b)

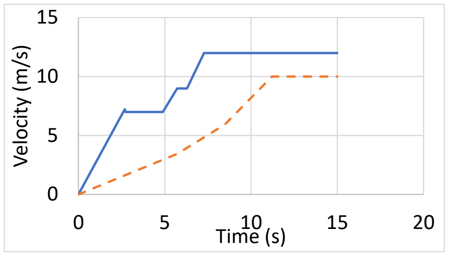
Fig. 11: Energy efficient path trajectories developed by CoMOP



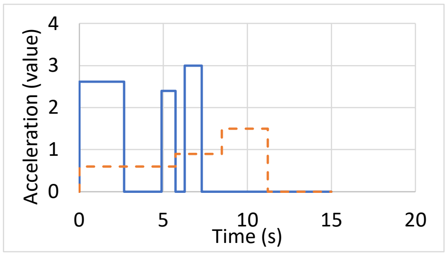
(a) Path Trajectory



(b) Energy Consumption



(c) Vehicle's Velocity



(d) Vehicle's Acceleration

Fig. 12: Energy- and time-efficient path trajectories and behavior of velocity, Acceleration, and Energy consumption

7.2 RQ2: Performance Improvement of Learning with CoMOP

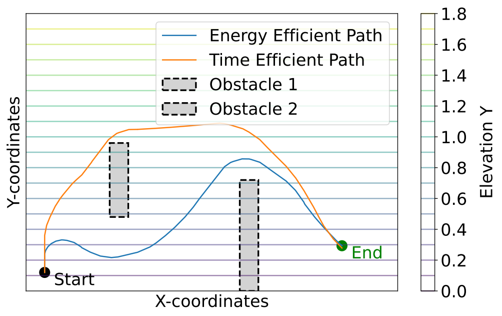
Figure 9 shows a subset of the experiments of Figure 8, highlighting the impact of obstacle height and the number of obstacles on path trajectories. Increasing obstacle height complicates strategy synthesis, particularly for two obstacles; learning a second sharp turn to reach the goal is more challenging. Although there is a significant margin between the obstacles and the map edges, MOP failed to generate any results as shown in the figure. We conducted simulations on a 20-size map with two obstacles to compare the accuracy of CoMOP and standalone MOP. CoMOP generates a holistic strategy by combining multiple strategies generated by MOP for the entire-map.

Figure 10 depicts the path trajectories of executing motion plans. It can be observed that CoMOP produces smoother trajectories toward the destination with less dramatic turnings compared to the trajectories generated by MOP alone. Figure 11 illustrates the versatility of CoMOP in generating path trajectories for intricate environments with multiple obstacles.

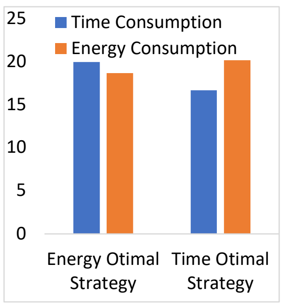
In summary, CoMOP improves overall motion-planning performance. It can even solve problems that cannot be addressed by MOP alone.

7.3 RQ3: Time-efficient and Energy-efficient Motion Plans

We synthesize energy- and time-efficient strategies and generate path trajectories to the destination by simulating the strategies. In Figure 12a, it can be seen



(a) Path trajectories with 12% slope on the y-axis.



(b) Energy and time

Fig. 13: Syntheses of time- and energy-efficient strategies to reach the destination by avoiding obstacles.

that there are distinct trajectories for time and energy. Figure 12b illustrates that the time-efficient strategy is fast but consumes more energy than that of the energy-efficient strategy. Figures 12c and 12d show that the energy-efficient strategy involves choosing a lower acceleration value and driving at a low speed to conserve energy. Driving at a low speed allows the vehicle to take sharp turns for an energy-efficient strategy comparatively, which can be seen in figure 12a. We further run an experiment in an environment having an obstacle and a 12% slope on the x-axis. Figure 13a depicts that CoMOP develops different trajectories for time and energy, and Figure 13b represents the cost of energy and time for both strategies.

By considering time and energy factors in strategy synthesis, we can generate balanced trajectories that optimize both time and energy usage by controlling the constant factors (`C1 and C2` shown in Figure 5) in the optimization function.

8 Conclusions and Future Work

In this paper, we propose automatic synthesis strategies for time- and energy-efficient motion planning for autonomous vehicles, using stochastic hybrid games in Uppaal Stratego. We present two techniques for motion planning based on the model: regular MOtion Planning (MOP) and Concatenated MOtion Planning (CoMOP). For CoMOP, we develop an assertion-based method to execute the Uppaal Stratego model in a cascaded fashion. The results show that the CoMOP performs better than MOP for complex models. We demonstrate the method's applicability on an industrial use case: energy-efficient path planning for an autonomous vehicle provided by Volvo CE, Sweden. The results show that our method can synthesize time-efficient and energy-efficient path trajectories.

In future work, we plan to extend the model to manage multiple operational vehicles in the environment and concurrent execution of each map's sub-part.

References

1. Aradi, S.: Survey of deep reinforcement learning for motion planning of autonomous vehicles. IEEE Transactions on Intelligent Transportation Systems **23**(2), 740–759 (2020)
2. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: Uppaal-tiga: Timed games for everyone. In: Nordic Workshop on Programming Theory (NWPT'06) (2006)
3. Bouton, M., Karlsson, J., Nakhaei, A., Fujimura, K., Kochenderfer, M.J., Tumova, J.: Reinforcement learning with probabilistic guarantees for autonomous driving. arXiv preprint arXiv:1904.07189 (2019)
4. Brorholt, A.H., Jensen, P.G., Larsen, K.G., Lorber, F., Schilling, C.: Shielded reinforcement learning for hybrid systems. arXiv preprint arXiv:2308.14424 (2023)
5. D. Wallace, N., Kong, H., J. Hill, A., Sukkarieh, S.: Energy Aware Mission Planning for WMRs on Uneven Terrains. IFAC-PapersOnLine **52**(30), 149–154 (2019), 6th IFAC Conference on Sensing, Control and Automation Technologies for Agriculture AGRICONTROL 2019
6. David, A., Jensen, P.G., Larsen, K.G., Mikučionis, M., Taankvist, J.H.: Uppaal Stratego. In: Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS-2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS-2015, London, UK, April 11-18, 2015, Proceedings 21. pp. 206–211. Springer (2015)
7. Gao, Z., Lin, Z., LaClair, T.J., Liu, C., Li, J.M., Birky, A.K., Ward, J.: Battery capacity and recharging needs for electric buses in city transit service. Energy **122**, 588–600 (2017)
8. Gómez, M., González, R., Martínez-Marín, T., Meziat, D., Sánchez, S.: Optimal motion planning by reinforcement learning in autonomous mobile vehicles. Robotica **30**(2), 159–170 (2012)
9. Gu, R., Jensen, P.G., Seceleanu, C., Enoiu, E., Lundqvist, K.: Correctness-guaranteed Strategy Synthesis and Compression for Multi-agent Autonomous Systems. Science of Computer Programming **224**, 102894 (2022)
10. Henkel, C., Bubeck, A., Xu, W.: Energy Efficient Dynamic Window Approach for Local Path Planning in Mobile Service Robotics. IFAC-PapersOnLine **49**(15), 32–37 (2016), 9th IFAC Symposium on Intelligent Autonomous Vehicles IAV 2016
11. Jothimurugan, K., Hsu, S., Bastani, O., Alur, R.: Robust Subtask Learning for Compositional Generalization. In: 40th International Conference on Machine Learning (ICML2023) (2023)
12. Kamgarpour, M., Ding, J., Summers, S., Abate, A., Lygeros, J., Tomlin, C.: Discrete time stochastic hybrid dynamical games: Verification & controller synthesis. In: 2011 50th IEEE Conference on Decision and Control and European Control Conference. pp. 6122–6127. IEEE (2011)
13. Lascurain, M.B., Franzese, O., Capps, G., Siekmann, A., Thomas, N., LaClair, T., Barker, A., Knee, H.: Medium truck duty cycle data from real-world driving environments: project final report. ORNL/TM-2012/240. Oak Ridge, TN: Oak Ridge National Laboratory (2012)
14. Mei, Y., Lu, Y.H., Hu, Y., Lee, C.: Energy-efficient motion planning for mobile robots. In: IEEE International Conference on Robotics and Automation, ICRA '04. vol. 5, pp. 4344–4349 Vol.5. IEEE (2004)
15. Pek, C., Schuppe, G.F., Esposito, F., Tumova, J., Kragic, D.: Spatial: Monitoring and planning of robotic tasks using spatio-temporal logic specifications (2023)

16. Quann, M., Ojeda, L., Smith, W., Rizzo, D., Castanier, M., Barton, K.: Chance constrained reachability in environments with spatially varying energy costs. Robotics and Autonomous Systems **119**, 1–12 (2019)
17. Quann, M., Ojeda, L., Smith, W., Rizzo, D., Castanier, M., Barton, K.: Power Prediction for Heterogeneous Ground Robots Through Spatial Mapping and Sharing of Terrain Data. IEEE Robotics and Automation Letters **5**(2), 1579–1586 (2020)
18. Rabin, S.: A* aesthetic optimizations, in: Game programming gems. Charles River Media (2000)
19. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: International Conference on Computer Aided Verification. pp. 202–215. Springer (2004)
20. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
21. Watkins, C.J.C.H.: Learning from delayed rewards. King's College, Cambridge United Kingdom (1989)
22. Xu, S., Zidek, R., Cao, Z., Lu, P., Wang, X., Li, B., Peng, H.: System and experiments of model-driven motion planning and control for autonomous vehicles. IEEE Transactions on Systems, Man, and Cybernetics: Systems **52**(9), 5975–5988 (2021)
23. Yang, Q., Simão, T.D., Jansen, N., Tindemans, S.H., Spaan, M.T.: Reinforcement learning by guided safe exploration. arXiv preprint arXiv:2307.14316 (2023)