Policy Optimization to Learn Adaptive Motion Primitives in Path Planning with Dynamic Obstacles

Brian Angulo[1], Aleksandr Panov[2] and Konstantin Yakovlev[3]

***Abstract*—This paper addresses the kinodynamic motion planning for non-holonomic robots in dynamic environments with both static and dynamic obstacles – a challenging problem that lacks a universal solution yet. One of the promising approaches to solve it is decomposing the problem into the smaller sub-problems and combining the local solutions into the global one. The crux of any planning method for non-holonomic robots is the generation of motion primitives that generates solutions to local planning sub-problems. In this work we introduce a novel learnable steering function (policy), which takes into account kinodynamic constraints of the robot and both static and dynamic obstacles. This policy is efficiently trained via the policy optimization. Empirically, we show that our steering function generalizes well to unseen problems. We then plug in the trained policy into the sampling-based and lattice-based planners, and evaluate the resultant POLAMP algorithm (Policy Optimization that Learns Adaptive Motion Primitives) in a range of challenging setups that involve a car-like robot operating in the obstacle-rich parking-lot environments. We show that POLAMP is able to plan collision-free kinodynamic trajectories with success rates higher than 92%, when 50 simultaneously moving obstacles populate the environment showing better performance than the state-of-the-art competitors.**

The code is available at https://github.com/BrianAnguloYauri/POLAMP.

***Index Terms*—Motion and Path Planning, Task and Motion Planning, Reinforcement Learning.**

I. Introduction

AUTONOMOUS robotic systems have become one of the most popular research topics in recent years due to their pronounced potential social benefits. In particular, autonomous driving is developing rapidly and at the same time requires efficient motion planning in complex and highly dynamic environments, meanwhile taking into account the kinodynamic constraints of an non-holonomic autonomous vehicle. Often, the planners that address the first aspect of the problem, i.e. dynamic environment, like the ones presented in [1], [2], [3] do not take into account the kinodynamic constraints. On the other hand, kinodynamic planners often do not explicitly reason about the future changes of the environments, even if these changes are for-seen, e.g. predicted by the control system of the robot. In this work we want to enrich the kinodynamic planning methods with the ability to take the dynamics of the environment as well (at the planning stage).

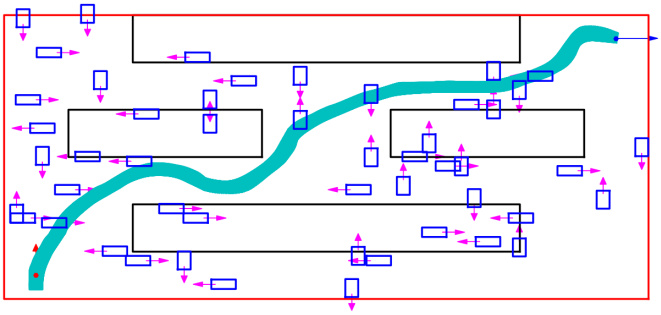

Fig. 1: Illustration of the POLAMP algorithm. **Red** arrow represent the start state, **Blue** arrow – the goal one. **Black** and **Blue** rectangles represent the static and dynamic obstacles respectively. **Cyan** curve is the generated trajectory by A*-POLAMP.

Two common approaches to kinodynamic planning are widespread: lattice-based and sampling-based planning methods. Lattice-based planning methods utilize the so-called motion primitives [4] that form a regular lattice. Each motion primitive represents a small segment of kinodynamically feasible trajectory of the robot, which is pre-computed before planning. At the planning stage the search-based algorithms (e.g. A* [5] of its variants) are used to find the resultant trajectory, represented as a sequence of the motion primitives. Contrary, sampling-based planners, e.g. RRT [6] or RRT* [7], grow a search tree by sampling states in the robot's configuration space and invoke a local planner to connect two states while respecting the kinematic constraints of the robot. Thus, the motion primitives are constructed online (i.e. while planning).

One of the prominent approaches to alleviate the complexity of local planners to respect the kinematic constraints of the robot is to use methods based on reinforcement learning such as methods proposed in RL-RRT [8] and PRM-RL [9]. In this work, we suggest Policy Optimization algorithm to Learn Adaptive Motion Primitives (POLAMP) to take into account the future changes of the environment at the planning stage, while producing plans that satisfy the kinodynamic constraints of the robot. POLAMP utilizes a reinforcement learning approach to find a policy that generates local seg-

Manuscript received: August 28, 2022; Revised December 18, 2022; Accepted December 27, 2022.

This paper was recommended for publication by Editor Bera Aniket upon evaluation of the Associate Editor and Reviewers' comments.

This work was partially supported by a grant for research centers in the field of artificial intelligence, provided by the Analytical Center for the Government of the Russian Federation in accordance with the subsidy agreement (agreement identifier 000000D730321P5Q0002) and the agreement with the Moscow Institute of Physics and Technology dated November 1, 2021 No. 70-2021-00138.

[1]Brian Angulo is with Moscow Institute of Physics and Technology and JSC Integrant brian.angulo@phystech.edu

[2]Aleksandr I. Panov is with Federal Research Center for Computer Science and Control RAS and AIRI panov@airi.net

[3]Konstantin Yakovlev is with Federal Research Center for Computer Science and Control RAS and AIRI yakovlev.ks@gmail.com

Digital Object Identifier (DOI): see top of this page.

ments of trajectory which are embedded in the global planning algorithms, RRT and A*, to generate a global motion plan. Our learnable local planner utilizes local observation to avoid both static and dynamic obstacles and, as well, respect the kinodynamic constraints of the robot. As a result, POLAMP is able to generate feasible solutions with high success rate ($> 92\%$) in the environments with up to 50 moving obstacles thus outperforming competitors.

II. Related Work

The problem of kinodynamic planning is well researched and various approaches such as graph-based, sampling based, optimization, reinforcement learning or the combination of them are used, see [10] for review. Nevertheless, the kinodynamic planning in presence of dynamic obstacles is still a challenging problem.

A widespread approach to kinodynamic planning in robotics is sampling-based planners. The most popular way to account for the robot's dynamics is to sample in the robot's state space and attempt to connect states via different local planners [11], [12], [13], [1], including for car-like robot [14]. The method described in this work also relies on the local planner, but it is learnable and takes the moving obstacles into account. Unlike the methods presented in [1], [15], it assumes that the information on how the obstacles are intended to move in future is available (e.g. predicted from the sensors' observations) and takes this information into account while planning.

Recently a lattice-based planner for car-like robots in highly dynamic environments was proposed [16]. Other variants of lattice-based planners for car-like robots are described in [4], [17], [18]. Contrary to these algorithms the suggested method does not construct a lattice in the high-dimensional space to search for a feasible plan, but uses a local learnable planner to connect states.

There also exist methods that, first, generate a rough path, often the one that does not take the kinodynamic constraints into account, and then generate controls to follow the path respecting the system's dynamics and avoiding obstacles. The variants of these methods are described in [19], [20], [21], [22]. Unlike them the method proposed in this work builds a feasible trajectory in one planning step. Avoiding the moving obstacles is performed by utilizing the knowledge of their future trajectories.

Finally, the most similar methods to the one presented in this article are RL-RRT [8] and PRM-RL [9]. Our method also uses a learning local planner inside a sampling-based planner. However, unlike these methods our local planner considers the presence of dynamic obstacles and is trained using a specifically-designed curriculum learning.

III. Problem Statement

We are interested in planning a feasible kinodynamic trajectory for a non-holomonic robot, that avoids both static and moving obstacles. In particular we are interested in car-like robots whose dynamics is described as [23]:

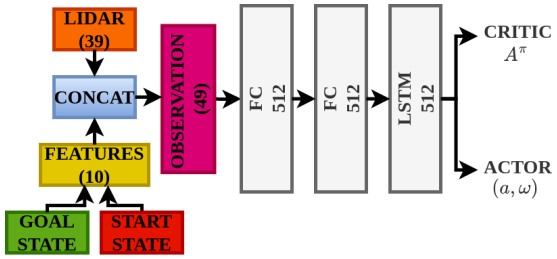

Fig. 2: Actor-Critic architecture that is implemented in POLAMP

$$\begin{aligned} \dot{x} &= v cos(\theta) \\ \dot{y} &= v sin(\theta) \\ \dot{\theta} &= \frac{v}{L} \tan(\gamma), \end{aligned} \tag{1}$$

where x,y are the coordinates of the robot's reference point (middle of the rear axle), θ is the orientation, L is the wheelbase, v is the linear velocity, γ is the steering angle. The former three variables comprise the state vector: $\boldsymbol{x}(t) = (x, y, \theta)$. The latter two variables form the control vector: $\boldsymbol{u}(t) = (v, \gamma)$, which can also be re-written using the acceleration a and the rotation rate ω as follows: $v = v_0 + a \cdot t, \gamma = \gamma_0 + \omega \cdot t$.

The robot is operating in the 2D workspace populated with static and dynamic obstacles. Their shapes are rectangular (as the one of the robot). Let $Obs = \{Obs_1(t), ..., Obs_n(t)\}$ denote the set of obstacles, where $Obs_i(t)$ maps the time moments to the positions of the obstacle's reference point in the workspace. For the static obstacles it obviously holds that $\forall t : Obs_i(t) = Obs_i(0)$. In our work, we consider the functions $Obs_i(t)$ to be known.

Denote by $\mathcal{X}_{free}(t)$ the configurations of the robot which are not in collision with any of the obstacles at time moment t (w.r.t. the robot's and the obstacles' shapes). The problem now is to find the controls (as functions of time) that move the robot from its start configuration s_{start} to the goal one s_{goal} s.t. that the kinodynamic constraints (1) are met and the resultant trajectory is in $\mathcal{X}_{free}(t)$.

IV. Method

We rely on the combination of the global and the local planners to solve the described problem. Global planner is aimed to systematically decompose the problem into the set of sub-problems which are easier to solve, i.e. moving from one configuration to another. The local planner is tailored to solve the latter problem. Any such a sub-problem is in essence the two-boundary value problem with additional constraints (prohibiting the robot to collide with the obstacles) which is hard to solve directly. To this end, we cast this problem as the partially-observable Markov decision process (POMDP) and obtain the policy for solving the POMDP via the reinforcement learning, more specifically via the custom-tailored Proximal Policy Optimization algorithm. Once the policy is obtained (learned) we plug it into the global planner. We use the adaptations of the renowned algorithms, RRT and A*, to get the final solvers. We name this type of solvers as POLAMP – Policy Optimization to Learn Adaptive Motion Primitives.

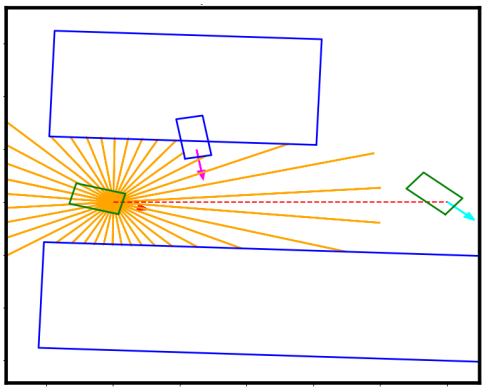

Fig. 3: The learning environment. **Green** rectangle with the **Red** arrow is the current state of the robot. **Green** rectangle with the **cyan** orientation is the goal desired state. **Blue** rectangles are the static obstacles and the **Blue** rectangle with **Pink** arrow is the moving obstacle. **Orange** lines are the laser beams.

A. Learnable Local Planner

a) Background: Formally, POMDP can be represented as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \Omega)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the state-transition model, $\mathcal{R}$ is the reward function, Ω is the observation space. During learning at each time step the agent receives an observation $o_t \in \Omega$, takes an action $a_t \in \mathcal{A}$ and receives a reward $r_t \in \mathcal{R}$. The goal is to learn a policy, i.e. the mapping from the observations to the distributions of actions, $\pi : \Omega \rightarrow P(\mathcal{A})$. The policy should maximize the following expected return from the start state s_t:

$$J(\pi) = \mathbb{E}_{r_i, s_i \sim \mathcal{P}, a_i \sim \pi}[\sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i) | s_t, a_t], i > t,$$

where γ is the discounting factor.

The Q-function is used to concise definition of the most essential information for the agent in order to make an optimal decision:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_i, s_i \sim \mathcal{P}, a \sim \pi}[R_t | s_t, a_t], i > t,$$

In this paper, we consider algorithms of the actor-critic family, which are more stable, have less variance, and are less prone to convergence to a local minimum. The actor updates the policy approximator $\hat{\pi}_w$ using the following equation:

$$\nabla_w J(\hat{\pi}_w) = \mathbb{E}_{\hat{\pi}_w}\left[\nabla_w \log \hat{\pi}_w(s, a) Q^{\hat{\pi}_w}(s, a)\right],$$

where $\hat{\pi}_w$ is an arbitrary differentiable policy. Critic evaluates the approximation of the $Q^{\hat{\pi}_w}(s, a)$ value for the current policy $\hat{\pi}_w$. Actor-critic algorithms have two sets of parameters: a critic updates parameters ϕ of the Q-function, and an actor updates parameters w of the policy according to the critic assumptions.

In this work, we use Proximal Policy Optimization method [24] (PPO) because it has shown the best performance among other methods in our preliminarly evaluation. Actor part of the PPO optimizes the clipped loss function

$$L(s, a, w_k, w) = \min(\frac{\pi_w(a|s)}{\pi_{w_k}(a|s)} A^{\pi_{w_{old}}}(s, a), \\ clip(\frac{\pi_w(a|s)}{\pi_{w_{old}}(a|s)}, 1-\epsilon, 1+\epsilon) A^{\pi_{w_{old}}}(s, a)),$$

where A^{π_w} is an estimation of the advantage function $A(s, a) = Q(s, a) - V(s)$ given by the critic part. Clipping is a regularizer removing incentives for the policy to change dramatically. The hyperparameter ϵ corresponds to how far away the new policy can go from the old while still profiting from the objective. When integrating the PPO algorithm into our method, we considered the state s_t as a function from observation $s_t \approx f(o_t)$, where f is lower layers of neural network approximator of the actor and critic shown in Fig. 2.

b) Observations, actions and rewards: In this paper, we consider actions $a_t = (a, \omega) \in R^2$ that are composed of setting the linear acceleration $a \in (-5, 5)\ m/s^2$ and rotation rate $\omega \in (-\pi/12, \pi/12)\ rad/s$. The latter ones can be converted to robot's controls using the transformations for Eq. 1, where we set the range of the linear velocity in $v \in (0, 4)\ m/s$ and steering angle in $\gamma \in (-\pi/6, \pi/6)\ rad$. Although we consider linear acceleration and rotation rate as actions of the policy, the latter, in principle, is able to work with different control inputs.

The observation o_t is a vector that consists of the $N_{beams} = 39$ measurements of the lidar that cover the 360° surrounding of the robot up to the length of $beam_{max} = 20\ m$ – see Fig. 3 concatenated with the features $(\Delta x, \Delta y, \Delta\theta, \Delta v, \Delta\gamma, \theta, v, \gamma, a, \omega)$, where $\Delta(s_i)$ stands for the difference between the respective parameter s_i of the goal state and the current one, (θ, v, γ) are last three parameters of the current state and (a, ω) are the current controls. We consider an ideal environment, so both simulation and actuation model do not have errors.

The reward function is described by:

$$\mathcal{R} = w_r^T [r_{\text{goal}}, r_{\text{col}}, r_{\text{field}}, r_t, r_{\text{backward}}, r_{v_{\max}}, r_{\gamma_{\max}}],$$

where w_r is a vector of weights, r_{goal} is 1 if the agent has reached the goal state with the $(\epsilon_\rho, \epsilon_\theta)$ tolerance and 0 otherwise, r_{col} is -1 if the agent collides with the obstacles and 0 otherwise, $r_{\text{field}} = \rho_{curr} - \rho_{last}$, where $\rho_{last} = \|s_{t-1} - s_{goal}\|$ and $\rho_{curr} = \|s_t - s_{goal}\|$ we penalize the agent for moving away from the goal, $r_t = -1$ is the constant penalty for each time step, r_{backward} is -1 the the agent is using rear gear (moving backwards) and 0 otherwise, $r_{v_{\max}}$ is -1 for exceeding the maximum speed limit, $r_{\gamma_{\max}}$ is -1 for exceeding the maximum of steering angle threshold. We set the weights to be $w_r = [20, 8, 1, 0.1, 0.3, 0.5, 0.5]$ (empirically those values result in a more efficient learning).

c) Curriculum policy learning: To accelerate training end we propose a three-stage curriculum learning (see Fig. 5). During the first stage, we train the agent in the empty environment. This stage is tailored to learn the kinodynamic constraints of the vehicle. Once the agent achieves an acceptable success rate (80% of the solved tasks), we stop training

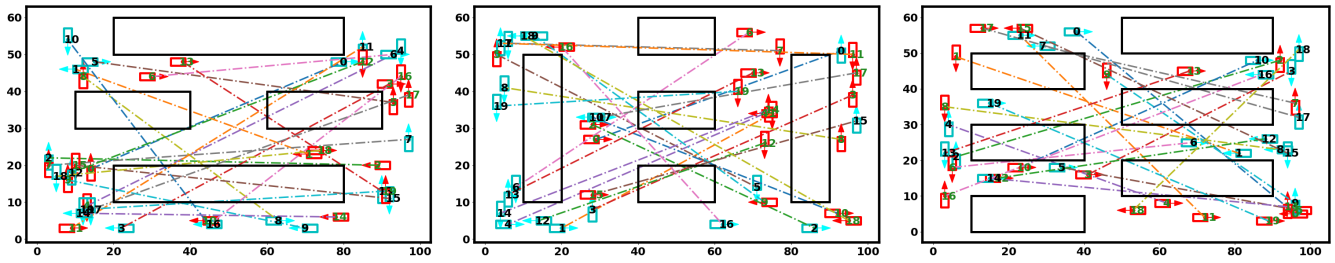

Fig. 4: Maps (1-3) used in our tests. **Red** rectangles and arrows show the start coordinates and orientations (with the start velocity $v = 0$ and the steering angle $\gamma = 0$), and **Cyan** rectangles and arrows show the goal coordinates and orientations.

Algorithm 1 POLAMP with RRT planner

Require: s_{start}, s_{goal}, $Obs(t)$, N_{max}, RL-PI, D, N_{nbs}, R_{ex}
Ensure: $\mathcal{P}$: Motion Plan

$s_{start}.t \leftarrow 0$
$\mathcal{T} \leftarrow$ INITIALIZETREE(s_{start})
while $N_{\max}$ was not reached **do**
  $s_{rand} \leftarrow$ RANDOMSAMPLE
  $neighbors \leftarrow$ NEAREST($\mathcal{T}, s_{rand}, N_{nbs}$)
  for $s_i \in neighbors$ **do**
    $s_j \leftarrow$ EXTEND(s_i, s_{rand}, R_{ex})
    $s_j \leftarrow$ RL-STEER($s_i, s_j, s_{goal}, Obs(t)$, RL-PI, D)
    if $s_j.tr$ is not empty **then**
      $\mathcal{T} \leftarrow$ APPEND(s_j)
      if $s_{goal}.tr$ is not empty **then**
        $\mathcal{T} \leftarrow$ APPEND(s_{goal})
        return $\mathcal{P}$ = MOTIONPLAN($\mathcal{T}$)
      else
        break
return $\mathcal{P} = \emptyset$

and proceed to the next stage. In the second stage, we re-train the policy in a new environment which is populated with static obstacles so the agent learns to avoid the collisions with them. In the last stage, we add an adversarial dynamic obstacle to the static environment so the agent learns to circumnavigate it or wait in place if needed to let the obstacle go away. The latter is the essential skill for planning with dynamic obstacles.

B. Global planners

Although our learnable local planner can generate a trajectory between two nearby states it is not well-suited for constructing a long-term plans. Thus we suggest using a global planner as well that can consistently explore different regions of the workspace relying on the global observation and find the ways to reach the remotely located goals. In this work, we utilize the classical algorithms RRT and A* as the global planners. For the detailed explanation of these algorithms we refer the reader to the original papers, and now proceed with an overview.

The pseudocodes of both algorithms are given in Alg. 1 and Alg. 2 respectively. The main difference between the sampling-based (i.e. RRT) and the lattice-based (i.e. A*) algorithms is how to choose the state to extend and how to extend the given state. On the one hand RRT uses *RandomSample* in the state space to grow the search tree randomly from the *Nearest* state in the tree using *Extend* to limit the maximum distance of the states that should be connected. On the other hand, A* does not choose a random sample, but rather uses a deterministic priority queue of states, OPEN, to choose which state to expand (extend). The OPEN queue is sorted in order of increasing f-values, where $f(s) = g(s) + \epsilon \cdot h(s)$ consists of two terms $g(s)$ and $h(s)$. $g(s)$ is the cost of the shortest path from the start state to the current one, and $h(s)$ is the heuristic estimate of the cost from s to goal. Upon choosing a most promising state A* the next states (*Successors*) using a fixed set of motion primitives through which the robot reaches the next states.

The major difference between these classical algorithms and POLAMP is that POLAMP explicitly reasons about time moments to take the dynamic obstacles into account while planning. Local planning is implemented with the *RL-STEER* function. This function solves a local planning problem, defined by the two states s_i and s_j. If the distance between s_i and s_{goal} is less than D then the goal is attempted to be reached from s_i. To reach the target state the policy **RL-PI** is used which has an access to the information on how the dynamic obstacles move, i.e. $Obs(t)$. If **RL-PI** managed to connect the states, it returns the generated trajectory $s_j.tr$ and the time by which the target state is reached, i.e. $s_j.t$. Thus all the states in the search tree bear the information on their reaching time which is used while planning.

In this work, we use a modified version of RRT, when at each iteration *Nearest* gets several N_{nbs} with the maximum radius of extend R_{ext} and tries to generate trajectories to them until one of them is build. Unlike the original algorithm A*, where the search ends when the goal state is expanded, in this work, the search ends as soon as the trajectory to the final state is found. To generate the successors we use the technique of online motion primitives from [16], i.e. we apply discrete controls $\xi = (a, \gamma)$ for a period of H to determine the robot's desired configurations. Then we use our learned policy to construct collision-free trajectories to these configurations.

V. EXPERIMENTAL EVALUATION

We evaluated POLAMP (and compared it with the competitors) in two types of environments: with static obstacles and with both static and dynamic obstacles.

Algorithm 2 POLAMP with A* planner

Require: s_{start}, s_{goal}, $Obs(t)$, H, ξ, D, RL-PI, N_{max}
Ensure: $\mathcal{P}$: Motion Plan

CLOSED $\leftarrow \emptyset$, OPEN $\leftarrow \emptyset$
$s_{start}.t \leftarrow 0$, $g(s_{start}) \leftarrow 0$, $f(s_{start}) \leftarrow h(s_{start})$
OPEN $\leftarrow$ INSERT(s_{start})
while OPEN is not empty or N_{max} was not reached **do**
  $s_i \leftarrow$ OPEN.POP(), CLOSED $\leftarrow$ INSERT(s_i)
  SUCCESSORS $\leftarrow$ GETNEXTSTATES(ξ, H)
  for $s_j \in$ SUCCESSORS **do**
    $s_j \leftarrow$ RL-STEER($s_i, s_j, s_{goal}, Obs(t)$, RL-PI, D)
    if $s_j.tr$ is empty **then**
      continue
    if $s_{goal}.tr$ is not empty **then**
      CLOSED $\leftarrow s_{t_n}$, CLOSED $\leftarrow s_{goal}$
      return $\mathcal{P}$ = MOTIONPLAN(CLOSED)
    $c(s_i, s_j) \leftarrow$COST($s_{t_n}.tr$)
    if $g(s_j)$ is better than any previous one **then**
      OPEN $\leftarrow$ INSERT(s_j)
return $\mathcal{P} = \emptyset$

A. Policy learning

To train the policy we created a dataset of different tasks (start and goal states) in three types of environments: empty, static, dynamic. Every of these environments had a size $40m \times 40m$. Each task was generated randomly in a way that the distance between the start and goal locations was in the interval of $[15, 30]$m, moreover the difference in orientations did not exceeded $\frac{\pi}{4}$. The task was considered solved if the agent reached the goal state with the Euclidean error $\epsilon_\rho \leq 0.3$ m and the orientation error $\epsilon_\theta \leq \pi/18$ rad with no collisions.

To generate tasks in static environments we sampled 12 fragments of size $40m \times 40m$ from the map depicted on Fig. 4 on the left (Map1), which has the size of $100m \times 60m$. For training in dynamic environments we populated the static environments with one adversarial dynamic obstacle. I.e. the start state of the dynamic obstacles and its trajectory were generated semi-randomly in such way that with a very high chance it will intersect the path of the agent and will force the latter to detour/wait. An illustration is given in Fig. 3.

Similarly to the train dataset we created a separate set of validation tasks. We used them to measure the progress of training, i.e. once in a while we evaluated the performance of the currently trained policy on the validation tasks. If the success rate (the fraction of the solved tasks) was lower than 80% we continue learning, in the opposite case – we stopped learning.

The effect of curriculum learning. To qualitatively assess the effect of the proposed curriculum learning we trained two policies: the first (baseline) was trained immediately in the dynamic environment, π^{stand}, while the second one, π^{curr}, was trained with the proposed three-stage curriculum. The corresponding learning curves are shown in Fig 5. Evidently the curriculum policy π^{curr} starts to converge from approx. 300M time step with almost 30 of reward and in this time the standard policy π^{stand} only achieves the reward of 13 (and starts converging later). Thus, we confirm that the suggested curriculum leads to a faster convergence, which is especially useful when the resources, e.g. training time, are limited.

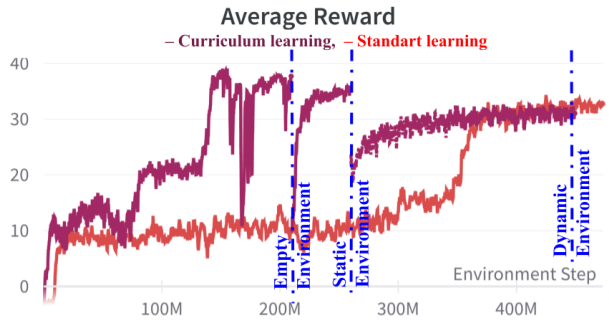

Fig. 5: A comparison of learning curves between curriculum and standart learning for our policy. The dash lines represent the intermediate trained policy in the respecting environment.

Agent	Dynamic	Orientation	SR %
$\pi^{st}_{w/o-\theta}$	no	no	99
$\pi^{st}_{w-\theta}$	no	yes	32
$\pi^{dyn}_{w/o-\theta}$	yes	no	28
$\pi^{dyn}_{w-\theta}$	yes	yes	22

TABLE I: The results of the trained DDPG agent in different setups.

Training the learnable baseline. The learnable baseline which we primarily aimed to compare with was RL-RRT [8]. Similarly to POLAMP it is a combination of the global planner, RRT, with the learnable local planner, based on the DDPG policy. To provide a fair comparison we trained this policy on our dataset from scratch. However, even after a prolonged training its success rate on the validation tasks was not exceeding 22%.

To understand the reasons of such performance we conducted additional training for the three variants of this policy in more simple setups. The characteristics of those setups and the resultant success rates are shown in Table I. Notably, the policy that ignored the orientation constraints and dynamic obstacles (the same setting from RL-RRT), $\pi^{stat}_{w/o-\theta}$, showed a very good performance – almost 100% success rate. This goes in line with the original paper on RL-RRT as the authors considered this setting. However, when the setup becomes more complex, the performance of the policy drops significantly. For example, the policy which ignores the dynamic obstacle, $\pi^{stat}_{w/-\theta}$, showed only 32% SR, and the one that ignores the goal orientation, $\pi^{dyn}_{w/o-\theta}$, – 28%. Thus, we conduct that this type of policy has an acceptable performance only in basic setups.

The poor performance of the RL-RRT in the case of more complex environmental conditions and with a large number of dynamic obstacles is primarily due to the instability of the learning process of the DDPG algorithm in a stochastic environment. DDPG belongs to the class of the off-policy methods, saves experience from different episodes in the replay buffer (including those that led to collisions), and generates a deterministic policy relative to the value function.

Map	Planner	SR,%	TTR,%	Samples,%	Time,%
1	POLAMP-RRT	100	100	100	100
	POLAMP-A*	100	93	179	103
	RRT-ES	90	120	3851	104
	RL-RRT	40	96	2424	578
	SST*	85	140	4124	111
	SST*(3x)	90	113	11576	351
2	POLAMP-RRT	100	100	100	100
	POLAMP-A*	100	78	121	85
	RRT-ES	62.5	143	1322	107
	RL-RRT	4.5	153	677	308
	SST*	82.5	123	1225	102
	SST*(3x)	100	100	3405	293
3	POLAMP-RRT	100	100	100	100
	POLAMP-A*	100	84	143	89
	RRT-ES	31	102	3426	98
	RL-RRT	8	126	1532	407
	SST*	58.8	141	3560	101
	SST*(3x)	85	111	9073	287

TABLE II: Results of the experiments on the static maps.

In POLAMP, we use the on-policy PPO method, when only the latest relevant trajectories are considered to improve the policy, which at the later stages of training are unlikely to contain collision situations. In a number of works [25], [26], [27], on-policy algorithms showed a significant advantage over off-policy in a stochastic environment, due to the ability to generate a stochastic policy. The advantage of PPO over DDPG in our task is undeniable when using curriculum learning when a replay buffer prevents the DDPG from adapting to the new conditions of the next stage of training.

B. Evaluation In Static Environments

We used three different maps, resembling the parking lots, for the evaluation – see Fig. 4. Each map had a size of $100m \times 60m$ and was generated based on the dataset from [28]. Please note, that only several fragments of Map1 were observed by the policy during training, while Map2 and Map3 were not used while training at all. For each map, we generated 20 different planning instances, i.e. the start-goal location pairs. We generated them randomly and discarded the instances for which the straight-line distance between start and goal was less than 50m (in order to avoid non-challenging tasks). Start/goal orientations were also chosen randomly as the multiplicative of 90°. Each test was repeated 30 times. A test was counted as failure if the robot was not able to reach the goal with following tolerance: $\epsilon_\rho \leq 0.5$ m and $\epsilon_\theta \leq \pi/18$.

We compared POLAMP to the following algorithms: RRT that utilized a well-known non-learnable steering function based on the exponential stabilization [29] (denoted RRT-ES), a kinodynamic motion planner SST* [30], RL-RRT [8] – a state-of-the-art planning method with a learnable local planner (details on learning this planner were provided above).

For the RRT part of the algorithms, we set the radius of the Extend method $R_{ext} = 10$ m, the maximum distance which we can reach the goal from is $D = 30$ m, the number of nearest neighbors $N_{nbs} = 5$ and the maximum number of iterations of the RRT $N_{max} = 1500$ for the POLAMP-RRT and RL-RRT, and $N_{max} = 3000$ for the RRT-ES and SST*. We additionally ran SST with 9000 iterations to study how the solution of this probabilistic complete and asymptotic optimal planner will improve. We denote this variant as SST*(3x). For POLAMP-A* we used the same parameters as for RRT. Additionally, we used the 7 discrete steering angles ranged uniformly between $[\gamma_{min}, \gamma_{max}]$, the linear velocity $v = 2$ and the time horizon $H = 3$ s to generate the lattice of the motion primitives. All these values were chosen following a preliminary evaluation aimed at identifying the suitable parameters' values.

The metrics we used were: success rate (SR) – how often the planner produces a path that reaches the goal, time to reach the goal (TTR), total number of samples and the runtime of the algorithm.

The results are presented in Table II. Notably, POLAMP has a much higher success rate compared to the other algorithms reaching almost 100% in every map. This shows that our learnable local planner, indeed, generalizes well to the unseen consitions. The observable trend is that POLAMP requires much fewer samples than RRT-ES to generate the motion plan. For example, for Map2 POLAMP requires 14x and 12x less samples compared to RRT-ES and SST* respectively. This is because POLAMP performs collision avoidance for local steering while RRT-ES and SST* do not. While the success rate of SST*(3x) is notably higher and approaches 100% on Map2, it requires way more samples compared to SST* and, consequtively, POLAMP. In comparison with RL-RRT, POLAMP also requires less samples.

Also, is can be noted that RL-RRT has a higher success rate for the Map1 than for the other maps, meaning that, unlike our policy, the policy of RL-RRT did not generalize well to the other two maps. We can suggest that the main reason for RL-RRT not being able to perform well on Map2 and Map3 is that the learnable component of that planner, i.e. DDPG, was not able to learn sufficiently well in our setup, i.e. provided only with the instances that were taken from the Map1. In other words, the DDPG policy was not able to learn well in our dataset and was overffited to Map1. Meanwhile, PPO that used the same amount of data for training, was able to generalize to solving local pathfinding queries on (the unseen during training) Map2 and Map3. Thus, we infer that PPO is a more sample efficient policy that, generally, should be preferred over DDPG in similar setups.

C. Evaluation In Dynamic Environments

For this series of the experiments, we used Map2 and Map3, i.e. the maps that were not used for training. These maps were populated with the varying number of dynamic obstacles: from 0 to 70. Every dynamic obstacle is a rectangular shape car-like robot. Its trajectory is generated by sampling the random control input (a, ω) every 10th time step. We generated 5 different trajectories for every dynamic obstacle. Two different start-goal pairs were chosen for each map. Each test was repeated 20 times for the sampling-based planners.

As before we compared POLAMP to RL-RRT. We also compared to A*-CMP [16]. For this algorithm we used the same parameters as for POLAMP-A*. Another baseline was the combination of RRT with the seminal Dynamic Window Approach (DWA) [31] as a local planner (RRT-DWA). The latter is capable of avoiding moving obstacles and is widely

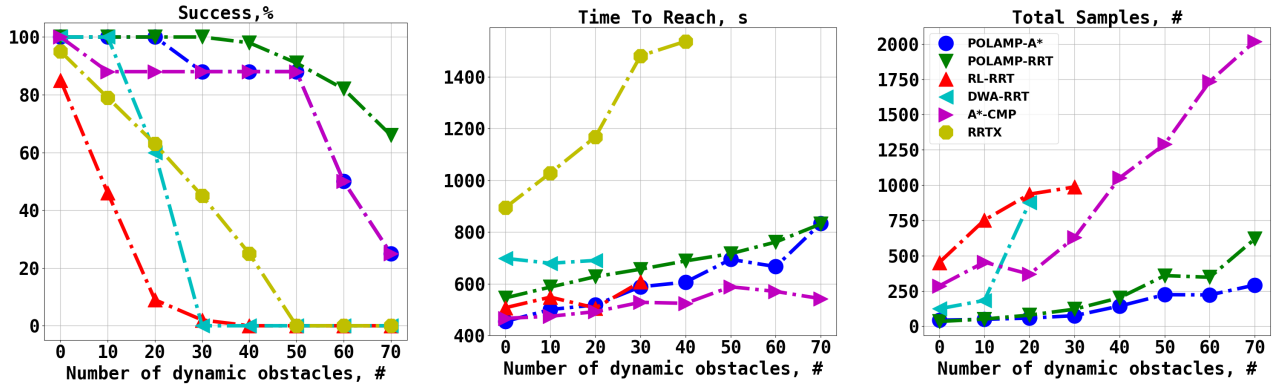

Fig. 6: Planning results for the maps with dynamic obstacles (success rate, time to reach and number of samples). The legend for all algorithms is shown in the figure on the right.

used in robotics. For RRT-DWA we did not account for the final orientation as DWA is not tailored to obey orientation constraints. Also, we compared to RRTX [1] that used Dubins steering function [32]. This algorithm is essentially a plan-execute-re-plan type of algorithm that re-uses the search tree while the robot is moving towards the goal. For better performance of RRTX at each re-planning iteration we did not take into account the moving obstacles located more then 20 m away from robot. In the case of RRTX, the SR means how often the robot can reach the goal without collisions while executing the path. Additionally because RRTX needs much more samples during the re-planning we do not show this metric for RRTX.

The results are presented in Fig 6. The first clear trend is that POLAMP-RRT, POLAMP-A* and A*-CMP in all cases maintain a high success rate ($> 92\%$) until the number of dynamic obstacles goes beyond 50. However, POLAMP-A* and POLAMP-RRT require much fewer samples than A*-CMP to find the trajectory. This is because the A*-CMP requires two groups of primitives. One group of primitives allows accelerate and move at a constant speed while another group tries to decelerate to avoid collision with dynamic obstacles. However our algorithm only requires one group of primitives, because our policy is able to decelerate to avoid collision with dynamic obstacles when it is necessary.

We also note that there are trade-off between POLAMP-RRT, POLAMP-A* and A*-CMP. On the one hand, POLAMP-RRT is slightly better than the baseline A*-CMP and our POLAMP-A* in terms of success rate. Thanks to the randomness of RRT, POLAMP-RRT is able to explore more and can solve complicated tasks, unlike A* which performs a systematic non-explorative search. On the other hand, A*-CMP has the lowest duration in comparison with the rest algorithms. The latter is because in each iteration A*-CMP uses the minimum and maximum acceleration to generate the neighbors, i.e. the algorithm makes abrupt changes in speed. However, our local learnable steering tries to change the speed smoothly due to the presence of obstacles. Our algorithm is better than the other baselines RL-RRT, and RRT-DWA. Due the poor performance of the $\pi^{dyn}_{w-\theta}$ the RL-RRT algorithm did not show good results. RRT-DWA works well only when the number of obstacles is small.

POLAMP-RRT and POLAMP-A* are also better than RRTX. RRTX tries to replan the path online but sometimes when the current path is occluded by dynamic obstacles the robot is forced to stop and stay in its place until it finds another solution. In these situations, the robot can get into a deadlock from where it is impossible to get out without a collision because of moving obstacles. This problem is due to RRTX not taking into account the future trajectories of dynamic obstacles while planning. Besides, TTR of RRTX is almost double that of the other algorithms. This is because RRTX has abrupt path changes when the path is affected by the appearance of dynamic obstacles.

Overall, the conducted experiments show that our policy π^{curr} generalizes well to both new environments and increasing number of dynamic obstacles (recall that it was trained only with one moving obstacle). A combination of that policy with a search-based or sampling-based global planner works well in challenging environments with dozens of simultaneously moving obstacles. Some experimental videos are provided in the Multimedia Materials.

VI. Conclusion

In this paper, we considered a problem of kinodynamic planning for non-holonomic robot in the environments with dynamic obstacles. We enhanced the two classical planning methods, A* and RRT, with a learnable steering function that takes into account kinodynamic constraints and both static and moving obstacles. We designed a reward function and created a specific curriculum for learning the steering behaviors. The resultant algorithm, POLAMP, was evaluated empirically in both static and dynamic environments and was shown to outperform the state-of-the-art baselines (both learnable and non-learnable).

Possibly, the main limitation of our work is that we assume the accurate knowledge of the future trajectories of the moving obstacles. Considering the uncertain trajectories is an appealing direction of future work. Another direction is to fuse more machine learning techniques to the planning

pipeline, e.g. to utilize the learnable methods for choosing the informative regions for driving expansions in global planners. We also plan to use POLAMP in applications related to self-driving vehicles, such as automatic parking with dynamic obstacles, etc. Some attempts in this direction have already been made [33], [34], but a complete experimental study is required.

REFERENCES

[1] M. Otte and E. Frazzoli, "Rrtx: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *The International Journal of Robotics Research*, vol. 35, no. 7, pp. 797–822, 2016.

[2] M. Phillips and M. Likhachev, "Sipp: Safe interval path planning for dynamic environments," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 5628–5635.

[3] K. Yakovlev and A. Andreychuk, "Towards time-optimal any-angle path planning with dynamic obstacles," in *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 2021, pp. 405–414.

[4] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009.

[5] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[6] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[7] S. Karaman, M. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT*," in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1478–1483.

[8] H. T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies," *IEEE Robotics and Automation Letters*, vol. 4, 2019.

[9] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson, "Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 5113–5120.

[10] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1135–1145, 2015.

[11] J. hwan Jeon, S. Karaman, and E. Frazzoli, "Anytime computation of time-optimal off-road vehicle maneuvers using the rrt," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*, 2011, pp. 3276–3282.

[12] D. J. Webb and J. Van Den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA 2013)*, 2013, pp. 5054–5061.

[13] C. Xie, B. J. Van Den, S. Patil, and P. Abbeel, "Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver." In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2015.

[14] G. Vailland, V. Gouranton, and M. Babel, "Cubic Bezier Local Path Planner for Non-holonomic Feasible and Comfortable Path Generation," *IEEE*, pp. 7894–7900, May 2021.

[15] Z. H. Y. Chen and S. Li, "Horizon-based lazy optimal rrt for fast, efficient replanning in dynamic environment," *Auton Robot*, vol. 43, p. 2271–2292, 2019.

[16] J. Lin, T. Zhou, D. Zhu, J. Liu, and M. Q.-H. Meng, "Search-based online trajectory planning for car-like robots in highly dynamic environments," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 8151–8157.

[17] M. Rufli and R. Siegwart, "On the design of deformable input-/state-lattice graphs," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 3071–3077.

[18] J. Ziegler and C. Stiller, "Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 1879–1884.

[19] C. Pérez-D'Arpino, C. Liu, P. Goebel, R. Martín-Martín, and S. Savarese, "Robot navigation in constrained pedestrian environments using reinforcement learning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 1140–1146.

[20] A. I. Panov, "Simultaneous Learning and Planning in a Hierarchical Control System for a Cognitive Agent," vol. 83, no. 6, pp. 869–883, 2022.

[21] G. P. Kontoudis and K. G. Vamvoudakis, "Kinodynamic motion planning with continuous-time q-learning: An online, model-free, and safe navigation framework," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 12, pp. 3803–3817, 2019.

[22] M. Jamal and A. Panov, "Adaptive Maneuver Planning for Autonomous Vehicles Using Behavior Tree on Apollo Platform," in *Artificial Intelligence XXXVIII. SGAI 2021. Lecture Notes in Computer Science*, M. Bramer and R. Ellis, Eds., 2021, vol. 13101, pp. 327–340.

[23] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[25] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," in *Proceedings of the 35th International Conference on Machine Learning, PMLR*, vol. 80, 2018, pp. 1861–1870.

[26] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller, "Maximum a posteriori policy optimisation," in *6th International Conference on Learning Representations*, 2018.

[27] M. Hessel, I. Danihelka, F. Viola, A. Guez, S. Schmitt, L. Sifre, T. Weber, D. Silver, and H. van Hasselt, "Muesli: Combining Improvements in Policy Optimization," in *Proceedings of the 38th International Conference on Machine Learning, PMLR*, vol. 139, 2021, pp. 4214–4226. [Online]. Available: http://arxiv.org/abs/2104.06159http://proceedings.mlr.press/v139/hessel21a.html

[28] M.-R. Hsieh, Y.-L. Lin, and W. H. Hsu, "Drone-based object counting by spatially regularized regional proposal network," pp. 4165–4173, 2017.

[29] A. Astolfi, "Exponential stabilization of a wheeled mobile robot via discontinuous control," *Journal of Dynamic Systems, Measurement, and Control*, vol. 121, no. 1, pp. 121–126, 1999.

[30] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 528–564, 2016.

[31] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[32] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[33] D. Ivanov and A. I. Panov, "Application of Reinforcement Learning in Open Space Planner for Apollo Auto," in *Proceedings of the Fifth International Scientific Conference "Intelligent Information Technologies for Industry" (IITI'21). IITI 2021. Lecture Notes in Networks and Systems*, S. Kovalev, V. Tarassov, V. Snasel, and A. Sukhanov, Eds. Springer, 2022, vol. 330, pp. 35–43.

[34] G. Gorbov, M. Jamal, and A. I. Panov, "Learning Adaptive Parking Maneuvers for Self-driving Cars," in *Proceedings of the Sixth International Scientific Conference "Intelligent Information Technologies for Industry" (IITI'22). IITI 2022. Lecture Notes in Networks and Systems*, S. Kovalev, A. Sukhanov, I. Akperov, and S. Ozdemir, Eds., 2023, vol. 566, pp. 283–292.