

SACPlanner: Real-World Collision Avoidance with a Soft Actor Critic Local Planner and Polar State Representations

Khaled Nakhleh Minahil Raza Mack Tang Matthew Andrews Rinu Boney Ilija Hadžić
Jeongran Lee Atefeh Mohajeri Karina Palyutina
Nokia Bell Labs - Murray Hill NJ, Espoo Finland & Cambridge UK

Abstract—We study the training performance of ROS local planners based on Reinforcement Learning (RL), and the trajectories they produce on real-world robots. We show that recent enhancements to the Soft Actor Critic (SAC) algorithm such as RAD and DrQ achieve almost perfect training after only 10000 episodes. We also observe that on real-world robots the resulting SACPlanner is more reactive to obstacles than traditional ROS local planners such as DWA.

I. INTRODUCTION

We study the efficacy of Reinforcement Learning (RL) algorithms for obstacle avoidance and local planning in ROS-based robotics systems. RL algorithms are able to learn optimal actions based on a current state and a reward function. The purpose of the ROS local planner is to adhere to a global path to the current robot goal while avoiding local obstacles (which may be dynamic). The RL paradigm is attractive for such a problem since the behavior of an RL agent does not have to be explicitly programmed for every possible scenario. In the RL framework, we specify the reward function, state space, and permissible actions the robot can take. The goal is to obtain a near-optimal planning policy given sufficient training samples. RL agents can potentially exhibit more complex (and hence more responsive) behavior than traditional local planners such as the Dynamic Window Approach (DWA) to Collision Avoidance [1].

RL has recently seen many advances due to the emergence of Deep RL, where the actions are chosen from a policy parametrized by a Deep Neural Network (DNN). One notable success of Deep RL is in learning policies for game environments (e.g. Atari games) modeled as Markov Decision Processes (MDPs) and standardized as *OpenAI Gym* environments [2]. As a result of this success, multiple authors have examined how Deep RL can be applied to robot control [3], [4], [5], [6].

However, these works raise a number of questions that we address in our study. First, they typically measure performance via an *episodic success* criterion, e.g. does the robot reach the goal, does it suffer any collisions etc? We are also interested in the *quality* of the trajectory. Is it smooth? How does it back off from an obstacle? Second, many of these papers address challenging environments where success rates are significantly below 90%. We believe such performance

is unacceptable for practical deployments. Therefore, we are interested in how to achieve near 100% success rates even in complex scenarios. Third, there are alternative obstacle-avoidance algorithms that do not use RL and we would like to quantify the benefits and drawbacks of using an RL-based approach. Lastly, we would like to know which specific RL techniques produce the best performance.

We follow [7], [3] and use ROS together with a *waypoint generator* that specifies a *next* waypoint based on the current robot location and a global plan to the goal. The task of the RL local planner is to reach this next waypoint without hitting any static or dynamic obstacles. Our RL state is an image representation of the obstacles and the next waypoint in polar coordinates. It mimics the image states used in the OpenAI gym environments for Atari games. We train our agents in a simulator with sample maps, and then upload the trained agents onto the robot for testing in the real world. With this setup, we list our contributions as follows:

- We show that modern variants of the Soft Actor-Critic (SAC) RL algorithm such as Reinforcement Learning with Augmented Data (RAD) [8] and Data-regularized Q (DrQ) [9] give significantly improved performance compared to earlier RL algorithms and implementations, and achieve success rates close to 100% after only 10,000 episodes. We refer to the resulting local planner as *SACPlanner*.
- We demonstrate that polar image state representations outperform natural alternatives.
- We analyze the trajectories produced by SACPlanner on real-world robots. (Prior work mostly limited trajectory analysis to simulations with perfect localization etc.) We compare with trajectories produced by DWA and a shortest-path based local planner. In all cases with an unexpected or dynamic obstacle, SACPlanner is much more reactive and hence performs better. The trade-off is a less smooth trajectory when the local planner simply has to follow the global plan.

II. TRAINING AND VALIDATION FRAMEWORK

We use a standard ROS stack in which the robot knows its position up to the accuracy of the localization system. The robot has a 2D map for fixed, known obstacles and it detects dynamic and unknown static obstacles using a LiDAR sensor. From the raw obstacle information the robot constructs a costmap in the form of an Occupancy Grid using

Work performed while K. Nakhleh, M. Raza and M. Tang were summer interns from Texas A&M, Åbo Akademi University and U. Maryland respectively.

the approach of Lu et al. [10]. The costmap window size for the local planner is $8m \times 8m$.

We integrate RL into the robot navigation stack using the framework pioneered by Gldenring et al. [7], [3]. When a new goal is specified the global planner creates a path from the current position to the goal (Fig.1). In this work we use without change the standard ROS *NavFn* planner based on the Dijkstra search algorithm. The path is found based on the obstacles in the map together with any obstacles seen by the LiDAR at the time of path creation.

Whenever a path is created by the global planner, a *waypoint generator* breaks it up into a sequence of waypoints. At all times the local planner maintains a list of 8 waypoints, starting with the one after the waypoint that is closest to the robot. (The method of [7] sometimes starts the list with the closest waypoint to the robot, but we found that could create excessive ‘‘pingponging’’ in the eventual choice of waypoint). From this list of 8, the local planner chooses the first on the list that is not too close to an obstacle. The aim of the RL agent is to move towards the selected waypoint while not hitting any obstacles, *including obstacles that appeared after the global plan was computed*.

A. RL Environment

The RL environment is defined by a state space \mathcal{S} , an action space \mathcal{A} , and a reward function $R(\cdot, \cdot)$. When the RL agent takes action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$, it gains reward $R(s, a)$ and moves to a new state s' according to some state-transition distribution $s' \sim p(\cdot | s, a)$. The actions are linear/angular velocity pairs (v, ω) . The state space is defined by the positions of the next waypoint and the local obstacles relative to the current robot position. We represent the state with an image since this allows us to utilize the convolutional deep RL architectures that have worked well for visually-rich environments such as Atari video games and some robot control tasks. In addition, using such game-like image states is a convenient way to merge the information from the waypoint position, the static objects from the map, and the dynamic obstacles sensed by the LiDAR.

Specifically, our RL state is an image that we refer to as the *polar costmap*. (See Fig. 1.) It is generated by converting the Occupancy Grid representation of the ROS costmap and the next waypoint to polar coordinates. The horizontal axis represents distance from the robot and the vertical axis represents angle. Obstacles are presented in red and the next waypoint is a white square. The motivation for using a polar representation is that it matches the linear/angular velocities that form the action. The state transition naturally follows from the robot movement after an action is taken.

It remains to define the reward function $R(s, a)$ for taking action a in state s . We employ a mix of both dense and sparse rewards. For a given state s , let $(d_{\text{old}}, \theta_{\text{old}})$ be the distance and bearing to the next waypoint in state s , let s' be the new state after taking action a , and let $(d_{\text{new}}, \theta_{\text{new}})$ be the distance and bearing in state s' . Here the bearing is defined to be the difference between the angle to the waypoint and

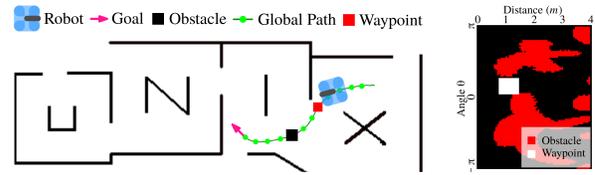


Fig. 1. ROS framework with global map, and polar costmap. The black square represents an obstacle that appeared after the global plan was computed.

the current yaw. We define:

$$\begin{aligned}
 R(s, a) = & (d_{\text{old}} - d_{\text{new}}) \cdot (1 \text{ if } d_{\text{old}} - d_{\text{new}} \geq 0, \text{ else } 2) \\
 & + (|\theta_{\text{old}}| - |\theta_{\text{new}}|) \cdot (1 \text{ if } |\theta_{\text{old}}| - |\theta_{\text{new}}| \geq 0, \text{ else } 2) \\
 & - R_{\text{max}} \cdot (1 \text{ if collision, else } 0) \\
 & + R_{\text{max}} \cdot (1 \text{ if } d_{\text{new}} = 0, \text{ else } 0) \\
 & - G(s'),
 \end{aligned}$$

where R_{max} is a fixed reward/penalty for reaching the waypoint and colliding with an obstacle, respectively, and $G(s')$ is the product of a truncated Gaussian kernel centered at the robot location and the Occupancy Grid in state s' .

The first two terms of $R(s, a)$ incentivize getting closer to the waypoint both in terms of distance and bearing. Note that the penalty for moving away from the waypoint (both in distance and bearing) is double the reward for moving towards it. Hence there is a net penalty for moving away from the waypoint and then back towards it. We have found that this ‘‘doubling the penalty for negative progress’’ has a significant effect on encouraging the agent to move directly to the waypoint if there are no obstacles in the way. The final Gaussian term penalizes movement towards an obstacle.



Fig. 2. Dummy training environment (left) with polar costmap (right).

We find that it is more efficient to train our RL agents on a ‘‘dummy’’ training environment that does not require the full complexity of ROS or a detailed physics simulation. For this dummy training environment we place a robot start position and a single waypoint in an environment with obstacles as shown in Fig.2. The robot is the blue square, the waypoint is the red square, and the larger green square around the robot is the support of the truncated Gaussian kernel. For each episode in the RL training, we pick an obstacle configuration and then use the above reward to encourage the RL agent to move towards the waypoint without hitting obstacles. Once the agent is trained we can run it directly in our ROS environment (either a Gazebo simulation or on real robots) since the state definition is the same in all cases. We remark, however, that the specific obstacle configurations on which

we do the training are *not* the same as the configurations on which we do our eventual experiments, since we want trained agents that generalize to any unseen obstacle configuration.

III. PREVIOUS WORK AND COMPARISON ALGORITHMS

The canonical local planner algorithm for ROS is the Dynamic Window Approach [1]. At each instant, DWA calculates a set of achievable (v, ω) pairs based on the current velocities and achievable acceleration characteristics of the robot. For each velocity pair DWA calculates a score based on how closely the arc follows the global plan, and on how far the arc is from any obstacle. It then chooses the best velocity pair based on this score.

Multiple recent papers have investigated how well local planner behavior can be learned via RL. Gldenring et al. [7] developed a framework that has been followed by many subsequent papers in which the global plan is partitioned into waypoints and the task of the RL agent is to get to the next waypoint.

Patel et al. [4] combines the DWA and RL approaches. The resulting DWA-RL algorithm calculates a cost for each potential velocity pair, but then uses RL to select the best pair based on the full spectrum of costs, rather than just picking the lowest cost pair. The work of Kstner et al. [5] distinguishes between humans, robots and static objects and uses an RL state that is a combination of the raw LiDAR input, the distance/angle to the goal, the position of nearby humans and the position of nearby robots. A follow-up paper [11] looks at different methods for choosing the next waypoint, and compares the fixed partition of Gldenring et al. [7], [3], with alternative methods that choose the waypoint more dynamically. The work of Liu et al. [6] uses a similar RL state. The main difference is that they represent pedestrian and robot movement using the CrowdNav algorithm of [12], and they represent the LiDAR information via both the raw LiDAR values and an Occupancy Grid.

In many of these papers the success rate of the trained agent is significantly under 100%. For example, the agent of [7] converges to a rate less than 70%. Moreover, this prior work typically provides trajectory plots from a Gazebo simulation. Our goal is to train an agent with close to 100% success, and then analyze trajectories from a real-world deployment (with the associated imperfections in sensing and localization). We also observe that if the goal is to get to the next waypoint, then an alternative is to repeatedly calculate a shortest path in the Occupancy Grid. We have found that modern python implementations of Dijkstra’s algorithm can do this sufficiently fast, and so we also compare against a local planner that uses the next segment of the shortest path to define the robot velocities. Note however that the shortest path will change over time as the robot and obstacles move.

IV. SOFT ACTOR CRITIC ALGORITHM

The objective in RL is to maximize the expected sum of rewards that the agent will receive in the future: $G = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$, where the expectation is taken over the agent policy $a_t \sim \pi(\cdot|s_t)$ and the state transition function

$s_{t+1} \sim p(\cdot|s_t, a_t)$. The parameter $\gamma \in (0, 1]$ is a discount factor used to reduce the weight given to future rewards.

Continuous control problems, such as the local navigation task considered in this paper, are often approached using actor-critic algorithms that learn two functions called the actor and the critic. The actor is a policy function $a \sim \pi_{\theta}(\cdot|s)$ with parameters θ . The critic $Q_{\phi}(s, a)$ with parameters ϕ estimates the action-value function $Q^{\pi}(s, a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a]$ of policy π , which is the expected cumulative reward after taking action a in state s and following policy π after that.

In this work, we use a state-of-the-art off-policy actor-critic algorithm called Soft Actor-Critic (SAC) [13], [14]. It is based on the maximum entropy RL framework which augments the standard RL objective with an entropy maximization objective: $G = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)))]$, where α is a learnable temperature parameter that balances the importance of both objectives. The entropy maximization motivates the agent to succeed at the task while acting as randomly as possible, aiding exploration.

In SAC, the actor and critic functions are parameterized as deep neural networks. The actor is a Gaussian policy with the mean and diagonal covariance parameters produced by the neural network. The actor and critic networks are updated by sampling minibatches of $(s_t, a_t, r_t, s_{t+1}, d_t)$ transitions from a replay buffer \mathcal{D} , where d_t is a terminal signal denoting the end of the episode. The parameters for the critic network Q_{ϕ} are trained to minimize the soft Bellman residual:

$$J_Q(\phi) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}, d_t) \sim \mathcal{D}} [Q_{\phi}(s_t, a_t) - y_t]^2,$$

where the learning target y_t is

$$y_t = r_t + \gamma(1 - d_t)V(s_{t+1}),$$

and the soft value function

$$V(s_t) = E_{a_t \sim \pi(\cdot|s_t)} [Q_{\bar{\phi}}(s_t, a_t) - \alpha \log \pi(a_t|s_t)] \quad (1)$$

is approximated using a Monte Carlo estimate of the policy π_{θ} and a target Q network $Q_{\bar{\phi}}(s_t, a_t)$ whose parameters $\bar{\phi}$ is maintained as the exponentially moving average of the Q network parameters ϕ . SAC also makes use of clipped double Q-learning [15], where the Q estimates are computed as the minimum value of an ensemble of two critic networks with different initializations trained on the same data. This helps prevent overestimation bias in Q-learning with non-linear function approximators.

The parameters of the actor/policy network π_{θ} are updated to maximize the maximum entropy RL objective:

$$J_{\pi}(\theta) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\mathbb{E}_{a_t \sim \pi_{\theta}(\cdot|s_t)} [\alpha \log \pi_{\theta}(a_t|s_t) - Q_{\phi}(s_t, a_t)] \right].$$

The learnable temperature parameter α can be automatically updated such that the policy network satisfies a minimum expected entropy constraint. See [14] for more details.

While SAC often performs well on continuous control tasks with low-dimensional observations, learning a mapping from high-dimensional states (images) to continuous actions (linear and angular velocities) typically requires massive

amounts of robot-environment interactions. This is because the agent must learn to extract the right information from the images to successfully perform the task at hand. SAC with a convolutional encoder can be used to learn low-dimensional representations of image observations, which are then provided to the actor and critic networks. However, this often fails. Sample-efficient learning of SAC agents from image observations requires additional supervision such as input reconstruction [16], contrastive representation learning [17], or image augmentations [8], [9].

In this work, we consider the recently proposed RAD [8] and DrQ [9] methods that apply image augmentations for sample-efficient learning of continuous control policies from image observations. In RAD and DrQ, the image observations are transformed with a random shift before each forward pass on the convolutional encoder. DrQ further proposes to average the Q-learning targets in Eq. 1 over K image transformations. This reduces the variance in the learning targets of the critic, improving the stability and efficiency of learning.

We apply random shift image augmentation (by ± 4 pixels) to the costmap observations. The augmented images are passed to a convolutional encoder consisting of four 3×3 convolutional layers with 32 filters and a stride of 1 followed by ReLU activation. The output of the final convolutional layer is flattened and passed to a fully connected layer, followed by layer normalization and tanh activation to yield a 50-dimensional state representation. The actor and critic networks use the same MLP architecture with 4 fully connected layers of 1024 hidden units. The actor predicts the mean and diagonal covariance of the Gaussian policy based on the encoded state vector. The critic networks predict the scalar state-action values based on the encoded state vector and an action vector. Following previous works [16], [17], [8], [9], we train the convolutional encoder network using only the critic loss and then detach the network parameters from the actor loss for improved training stability.

V. PERFORMANCE OF RL TRAINING

We now evaluate the training performance of RAD, DrQ and other baseline RL methods in our dummy training environment. We train the RL agents on our polar costmap environment from Section II-A, and compare against Cartesian costmap environments (similar to [7]) where we do not convert to polar coordinates before generating the image. We train for 10,000 episodes with the hyper-parameter values listed in Table I. The trained agents are evaluated over 1000 episodes in the training environment. We define the success rate as the percentage of episodes in which the agent reached the goal. The collision rate is defined as the percentage of episodes where the agent collides with the obstacles. An episode can be neither a success nor a collision if the robot stops and the episode times out.

In Table II, we compare the polar and Cartesian costmaps using the RAD version of SAC. While the information regarding the robot’s orientation is implicit in the polar costmap, this information is missing in the Cartesian

TABLE I
HYPER-PARAMETER VALUES FOR SAC AGENT TRAINING.

Hyper-parameter	Value
Training episodes	10000
Random exploration episodes	10
Mini-batch size	128
Replay buffer capacity	10^6
Discount factor γ	0.99
Optimizer	Adam
Learning rate	0.001
Critic target update frequency	2
Critic target update rate τ	0.01
Actor update frequency	2

TABLE II
COMPARISON OF CARTESIAN AND POLAR COSTMAPS USING RAD AGENT IN THE DUMMY ENVIRONMENT.

Costmap	Orientation Information	Success Rate	Collision Rate
Polar	Implicit	98.7%	0.08%
Cartesian	Rotation	42.0%	37.7%
	Arrow	45.5%	36.0%
	Channel	65.3%	25.9%

costmap. We explored three ways to represent this: (i) rotating the Cartesian costmap by the robot orientation angle, (ii) drawing an arrow at the center of the costmap to denote the robot orientation, or (iii) appending an extra channel to the costmap with the robot orientation angle. The agent with polar costmap observations significantly outperforms those with Cartesian costmap observations. We hypothesize that this is because the polar costmaps better match the action space of the robot and also implicitly represent the robot orientation information, which allows for better generalization. We use the better performing polar costmap environment in the rest of our experiments.

We next compare the performance of RAD, DrQ (with $K=2$), and the following RL baselines in Table III:

- **DQN.** To evaluate if discrete control is easier to learn, we discretize the action space of the robot with six possible linear/angular velocity pair combinations and train a standard DQN agent from the stable baselines library [18].
- **PPO.** To evaluate if the SAC agents perform better than other actor-critic algorithms, we also compare against the popular PPO agent from the stable baselines library [18].
- **SAC from raw LiDAR observations.** To evaluate the importance of image-based game-like states, we compare against a SAC agent trained on raw LiDAR observations (similar to [5], [6]). For this agent, the actor and critic networks receive state vectors consisting of the raw LiDAR readings and the coordinates of the next waypoint.
- **DWA-RL with SAC.** To evaluate if it is beneficial to combine the standard DWA planner with RL, we implement the observation space and reward function of the DWA-RL method [4] and train our SAC agent on this hybrid setup.

The DrQ method achieves the highest success rate ($> 99\%$) with the fewest collisions. We also experimented with

TABLE III
COMPARISON OF RL AGENTS IN THE DUMMY ENVIRONMENT.

Method	Success Rate	Collision Rate
DQN	33.9%	51.2%
PPO	83.6%	7.5%
SAC from LiDAR	34.2%	47.5%
DWA-RL with SAC	7.3%	70.3%
RAD	98.7%	0.08%
DrQ	99.4%	0.02%



Fig. 3. Robot experiment test cases.

stacking four consecutive frames as observations to the DrQ method but observed that these agents tend to have trouble navigating around obstacles, reducing the success rate to 94.9%. We note that the success rates we obtain with the baseline algorithms are lower than those observed in the literature [7], [4], [5], [6]. We believe this is partly because we only run for 10,000 episodes (which corresponds to < 500000 steps). However, this is sufficient for training the DrQ agent and demonstrates the sample-efficiency of this variant of SAC. Another potential reason is that our training environment contains challenging scenarios requiring tight turns (see Fig. 2), but this is necessary to obtain agents that will work for the real-world cases described below.

VI. DESIGN OF ROBOT EXPERIMENTS

We now describe our experiments for testing the local planners on a physical robot. We use a ClearPath Robotics Jackal robot [19] equipped with LiDAR, set to a scanning frequency of 5Hz. The experiments cover a range of scenarios that an autonomous robot would encounter in the physical world. A failed traversal translates into a robot’s collision with a static obstacle (e.g. wall), or a dynamic obstacle (e.g. pedestrian). Moreover, if the planner fails to complete the global plan, then the robot fails that scenario. In addition to simply measuring success/failure, we are also interested in the nature of the trajectory produced by each approach. Is it smooth? How does the robot react to an obstacle?

Test cases. The experiments were conducted in a facility that includes an open room and a maze component with tight corners and narrow doorways shown in Fig.3. We refer to the maze shown in the first two images of Fig.3 as the *UNIX maze room* (named after letters that make up obstacles in four separate rooms). We describe four test cases:

- **(C1) Room I to room N through doorway:** shown in Fig.3 (left). Here the robot’s task is to travel through a narrow doorway while making a 180-degree turn. In this case, all the obstacles are fixed and included in the global map, and

so the only job of the local planner is to follow the global plan (which will be a collision-free path from start to goal) as closely as possible. However, in order to make the turn smoothly the planner must maintain a small turn radius (the ratio between linear and angular velocity).

- **(C2) Room I to room X with “unexpected” static obstacle:** shown in Fig.3 (mid). In this experiment, the robot goal is selected before the obstacle is in place. After the goal is selected and the global plan is computed, a static obstacle (a cardboard cutout of a person) is placed in the robot’s global plan. As the robot nears the obstacle, the next eligible waypoint will be beyond the obstacle and the local planner will need to navigate round the obstacle.

- **(C3) Avoiding a walking pedestrian on a straight path:** shown in Fig.3 (right). Here the robot must traverse a straight path while a pedestrian is walking towards the robot. This case tests the local planner’s ability to detect and navigate around a moving object. For this experiment it would always be possible to generate an “unavoidable collision” by having the pedestrian walk quickly at high speed into the robot. To avoid this we ask the pedestrian to stop when they are right in front of the robot. The desired behavior is then for the robot to back up or turn round the pedestrian. The undesired behavior is to keep on moving forward into the pedestrian.

- **(C4) Pedestrian crossing the robot path:** We extend the previous test case (C3) by asking the pedestrian to perpendicularly cross the robot’s global path. The desired behavior is for the robot to wait and then continue after the pedestrian has crossed.

Local planners. We test with the DrQ variant of SAC since it had the best training performance of all the RL algorithms in Section V. We log the trajectories for the resulting *SACPlanner* and compare against the Dynamic Window Approach (DWA), as well as the Shortest Path (SP) planner discussed in Section III that always tries to get to the next waypoint using a shortest path in the Occupancy Grid.

VII. EXPERIMENTAL RESULTS

The robot trajectories for each of (C1)-(C4) are shown in Fig.4. We denote the start and goal along with the collision points. For (C1)&(C2) we swap the direction of travel for half the runs. The color of the trajectory represents linear velocity. We also show the Occupancy Grid values in gray (taken from the map and the LiDAR). For (C3)&(C4) with a dynamic obstacle the gray shading captures all the positions of the obstacle over time. The 3 local planners have qualitatively different behavior which we now describe in detail for each case.

- **(C1):** DWA (which generates circular arcs) has the smoothest trajectory through the door. However, when starting at the top it miscalculated the best turning radius and aborted next to the ‘N’ obstacle each time. The SP planner never collided with an obstacle and (not surprisingly since it was running shortest paths on a grid) it traveled in a series of straight lines (whose endpoints are denoted with green dots). SACPlanner was also successful in all cases. However, it had

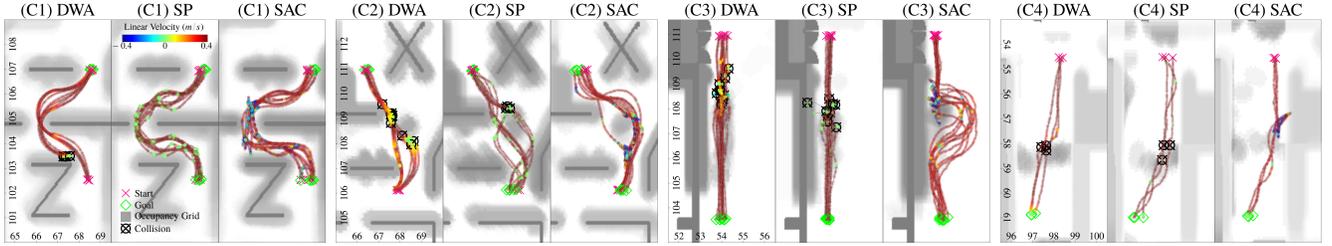


Fig. 4. Trajectory comparison between DWA, Shortest Path (SP) vs. SAC agent for each test case.

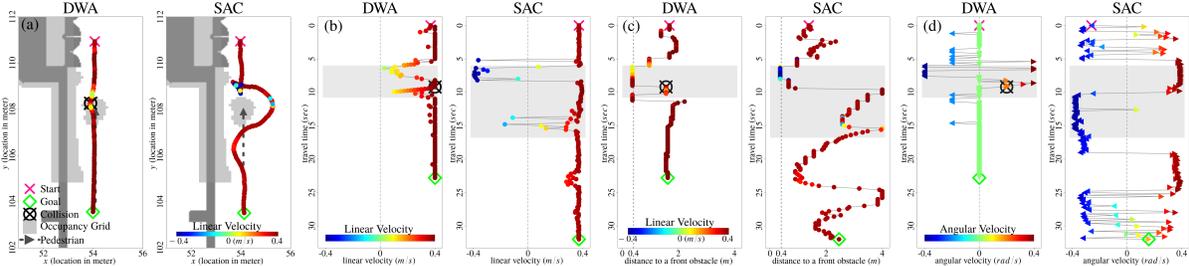


Fig. 5. Trajectory comparison between DWA and SACPlanner based on logs from the test case (C3).

to “back off” multiple times (denoted by the blue parts of the trajectory) before aligning correctly with the doorway.

- **(C2):** In this case a static obstacle appears on the global plan. Although DWA tried to deviate from the global plan, it did not do so enough, and therefore collided with the obstacle every time. The SP planner was successful when starting from the bottom. When starting from the top, the shortest path around the obstacle alternated between “going left” and “going right”. This indecision led to some collisions. SACPlanner often backed off multiple times when confronted with the obstacle. However, it eventually made it round the obstacle every time.

- **(C3):** Both DWA and SP were unable to deal with the fact that the pedestrian obstacle was approaching and hence the “correct” trajectory kept changing. Even though the pedestrian stopped right in front of the robot, both DWA and SP kept going and caused a collision. SACPlanner went backwards when the pedestrian got close and then directed the robot to take a wide berth in the available open space.

Quantitative metrics: In Table IV we show the mean travel time (s), mean travel distance (m), mean speed (m/s), and collision rate for the 3 local planners on (C1)-(C3) across all runs. For DWA on (C1) we only consider the non-aborted runs. For (C2)-(C3) we remove the obstacle after each collision and so the robot will still reach the goal. We note that the “backing off” behavior of SACPlanner leads to greater distances/times than DWA and SP, but this how it is able to achieve a much lower collision rate.

TABLE IV

SUMMARY STATISTICS OF TRAJECTORIES FROM TEST CASES.

	(C1)			(C2)			(C3)		
	DWA	SP	SAC	DWA	SP	SAC	DWA	SP	SAC
Time	21.80	30.10	37.20	30.70	20.90	28.50	27.50	22.30	33.10
Distance	7.13	8.93	10.70	5.47	6.26	8.57	8.77	8.01	10.80
Speed	0.33	0.30	0.29	0.18	0.30	0.30	0.32	0.36	0.33
Collision	0.5	0	0	1.0	0.3	0	1.0	0.9	0

- **(C4)** When the pedestrian switches to walking across the robot’s path rather than walking towards it, the results are similar to (C3). Both DWA and SP are not reactive enough and collide every time. However, SACPlanner backs off when the pedestrian is close, and then resumes traveling towards the goal after the pedestrian has passed through.

A. Trajectory Analysis

In order to understand more deeply the difference in behavior of DWA and SACPlanner, Fig.5 depicts a single run from test case (C3). Fig.5(a) shows the trajectory, Fig.5(b) plots the linear velocity, Fig.5(c) shows the distance to the nearest ‘front obstacle’ (within $\pm \frac{\pi}{4}$ rad range from the current yaw), and Fig.5(d) plots the angular velocity.

The key feature of these plots is that when the pedestrian is close, DWA slows down and turns a little, whereas SACPlanner goes into reverse (note the blue color in Fig.5(b)&(c)) and turns a lot so as to go around the pedestrian. This “reactiveness” to obstacles also manifests in more turning even when the robot can go in a straight line.

VIII. CONCLUSIONS AND FUTURE WORK

In this work, we have examined how training for RL-based local planners can be improved by using polar costmaps and regularization on top of the SAC algorithm to achieve success rates close to 100% after only 10,000 episodes. In addition, we have done a detailed trajectory analysis to show how the resulting SACPlanner is more robust and more responsive to dynamic obstacles than non-RL algorithms. For future work, we would like to improve the smoothness of SACPlanner when there are no unexpected obstacles, and we plan to develop a cooperative version of SACPlanner for when two or more robots are in close proximity.

REFERENCES

- [1] 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.
- [2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: <http://arxiv.org/abs/1312.5602>
- [3] R. Gldenring, M. Grner, N. Hendrich, N. J. Jacobsen, and J. Zhang, "Learning local planners for human-aware navigation in indoor environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021*. IEEE, 2020, pp. 6053–6060. [Online]. Available: <https://doi.org/10.1109/IROS45743.2020.9341783>
- [4] U. Patel, N. K. S. Kumar, A. J. Sathiamoorthy, and D. Manocha, "DWA-RL: Dynamically feasible deep reinforcement learning policy for robot navigation among mobile obstacles," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6057–6063.
- [5] L. Kstner, C. Marx, and J. Lambrecht, "Deep-reinforcement-learning-based semantic navigation of mobile robots in dynamic environments," in *16th IEEE International Conference on Automation Science and Engineering, CASE 2020, Hong Kong, August 20-21, 2020*. IEEE, 2020, pp. 1110–1115. [Online]. Available: <https://doi.org/10.1109/CASE48305.2020.9216798>
- [6] L. Liu, D. Dugas, G. Cesari, R. Siegwart, and R. Dub, "Robot navigation in crowded environments using deep reinforcement learning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 5671–5677.
- [7] R. Gldenring, "Applying deep reinforcement learning in the navigation of mobile robots in static and dynamic environments," Master's thesis, University of Hamburg, 2019.
- [8] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement learning with augmented data," *Advances in neural information processing systems*, vol. 33, pp. 19 884–19 895, 2020.
- [9] I. Kostrikov, D. Yarats, and R. Fergus, "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels," *arXiv preprint arXiv:2004.13649*, 2020.
- [10] D. V. Lu, D. Hershberger, and W. D. Smart, "Layered costmaps for context-sensitive navigation," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 709–715.
- [11] L. Kstner, X. Zhao, T. Buiyan, J. Li, Z. Shen, J. Lambrecht, and C. Marx, "Connecting deep-reinforcement-learning-based obstacle avoidance with conventional global planners using waypoint generators," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - Oct. 1, 2021*. IEEE, 2021, pp. 1213–1220. [Online]. Available: <https://doi.org/10.1109/IROS51168.2021.9636039>
- [12] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6015–6022.
- [13] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.
- [14] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.
- [15] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.
- [16] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus, "Improving sample efficiency in model-free reinforcement learning from images," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, 2021, pp. 10 674–10 681.
- [17] A. Srinivas, M. Laskin, and P. Abbeel, "Curl: Contrastive unsupervised representations for reinforcement learning," *arXiv preprint arXiv:2004.04136*, 2020.
- [18] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: <http://jmlr.org/papers/v22/20-1364.html>
- [19] "Jackal unmanned ground vehicle," ClearPath Robotics, Product Datasheet, available online, <https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/>.