# Hybrid Classical/RL Local Planner for Ground Robot Navigation

**Vishnu Dutt Sharma, Jeongran Lee, Matthew Andrews, Ilija Hadžić**

**Keywords:** Mobile robot navigation, Local path planning, Hybrid planning

## Summary

Local planning is an optimization process within a mobile robot navigation stack that searches for the best velocity vector, given the robot and environment state. Depending on how the optimization criteria and constraints are defined, some planners may be better than others in specific situations. We consider two conceptually different planners. The first planner explores the velocity space in real-time and has the robot's dynamic model. It has superior path-tracking and motion smoothness performance. The second planner was trained using reinforcement learning methods to avoid obstacles. It is better at avoiding dynamic obstacles, but at the expense of motion smoothness. We propose a simple, yet effective, meta-reasoning approach that takes advantage of both approaches by switching between planners based on the surroundings. We demonstrate the superiority of our hybrid planner, both qualitatively and quantitatively, over individual planners on a live robot in different scenarios, achieving an improvement of **26%** in the navigation time.

## Contribution(s)

1. This paper present a hybrid local planner for ground robots that uses a classical planner when the immediate environment is simple, and an RL-based planner for more complex local environments.
   **Context:** There have been many recent efforts that apply RL to ground robot local planners. To the best of our knowledge, these use RL all the time, which we believe is excessive for simple environments where classical planners work well.
2. A key contribution is a simpler criterion that decides whether the classical planner or the RL-based planner should be used.
   **Context:** Training one RL planner that operates well in every environment is a difficult and impractical task. Specifically, the motion smoothness can be improved by adding robot dynamics to the reward function at the expense of making the training more difficult. Alternatively different (more specialized) models can be used depending on the situation (classical model that knows robot dynamics vs. learned RL model in our case). We show that a simple decision criterion is sufficient to achieve the benefit from both models and that training another network to implement this meta-policy is unnecessary.
3. We integrate our hybrid local planner into a full ROS stack and implement on physical robots.
   **Context:** Direct deployment of RL-based planners to real-world may not result in efficient operation and may even require fine-tuning in real-world. By implementing and testing our system on physical robots, we show that the proposed solution is applicable in practice.
4. We demonstrate via extensive simulations that our hybrid local planner achieves the "best of both worlds", in that it balances between travel time for simple environments and collision avoidance for more complex dynamic environments.
   **Context:** Previously reported work on hybrid planning focus mainly on collision avoidance for "social navigation".

# Hybrid Classical/RL Local Planner for Ground Robot Navigation

**Vishnu Dutt Sharma**[†], **Jeongran Lee**[†], **Matthew Andrews, Ilija Hadžić**
{vishnu.1.sharma, jeongran.lee, matthew.andrews, ilija.hadzic}@nokia-bell-labs.com

**Nokia Bell Labs, Murray Hill, USA**
[1]**Work done while V. D. Sharma was a summer intern from the University of Maryland, College Park, USA**

[†] Equal contribution

## Abstract

Local planning is an optimization process within a mobile robot navigation stack that searches for the best velocity vector, given the robot and environment state. Depending on how the optimization criteria and constraints are defined, some planners may be better than others in specific situations. We consider two conceptually different planners. The first planner explores the velocity space in real-time and has superior path-tracking and motion smoothness performance. The second planner was trained using reinforcement learning methods to produce the best velocity based on its training "experience". It is better at avoiding dynamic obstacles, but at the expense of motion smoothness. We propose a simple, yet effective, meta-reasoning approach that takes advantage of both approaches by switching between planners based on the surroundings. We demonstrate the superiority of our hybrid planner, both qualitatively and quantitatively, over individual planners on a live robot in different scenarios, achieving an improvement of **26%** in the navigation time.

## 1 Introduction

A mobile robot navigation stack is broadly responsible for safely (and desirably optimally) getting the robot from its present position to the goal while respecting externally or internally imposed constraints. Components of a path and motion planning and control subsystem can be broadly categorized into global planners, local planners/controllers, and motion controllers, which are typically deployed in concert. Global planner finds the path toward the goal location, often expressed as a set of waypoints that the robot must visit. The local planners are responsible for generating the velocity vectors to lead the robot towards the next waypoint.

In a known map, global planners are optimal as they utilize the global costmap, but are brittle in the presence of unknown (and discovered after the fact) dynamic obstacles, such as humans, clutter, unmapped fixtures, and other vehicles. Local planners, on the other hand, can react well in such situations. Additionally, local planners take less time to compute and thus process the data at a higher frequency.

Local planning in velocity space can be characterized as an optimization process (which may in practice produce suboptimal, but acceptable solutions), whose optimization criteria include distance to the next waypoint (or the goal), clearance around the obstacle, smoothness of motion, energy efficiency, and the like. For this discussion, we broadly classify the implementations into classical and learning-based approaches. Classical planners explore the velocity space and evaluate each

proposed velocity against the constraints and the optimization criteria in real-time. Constraints, such as acceleration limits, can be derived from konow vehicle dynamics. To find an optimal solution a classical planner must often search the entire space of admissible velocities, which, depending on the size of the planning window, the number of degrees of freedom, and the complexity of constraints, can be a challenging process.

Learning-based planners are exposed to various situations offline and trained to map the robot state to the deemed best velocity, typically using a neural network. The complexity of searching and evaluating the solution is moved to an offline training process. The real-time computation becomes the model inference and does not involve explicit search. The performance of these planners strongly depends on how the training environment was set up, the variety of situations the robot has been exposed to, and how well the dynamics of the robot were captured during the training. Typically, reinforcement learning (RL) techniques are used here. As with all learning-based algorithms, false results are possible and it is impossible to guarantee that the RL-planner will always produce optimal or even correct solutions. Nevertheless, RL-planners have been shown to produce useful results that generalize well (Güldenring et al. (2020); Van Dinh et al. (2017); Liu et al.; Kästner et al. (2021); Patel et al. (2021); Nakhleh et al. (2023)). In Nakhleh et al. (2023), we designed an RL-planner with superior obstacle-avoidance performance compared to a widely used Dynamic Window Approach (DWA) planner (Fox et al. (1997)), but the price of this improvement is an uneven and jerky motion, even when no dynamic obstacles were present in the robot path.

This lack of smoothness limits the attractiveness of RL-based local planners as a general solution. If the robot is moving through a large open space, or if it is moving in a maze-like structure with known, fixed walls, it can stay close to the global plan. Classical local planners typically excel at generating smooth motion toward the goal. In this case, instantaneous decisions of an RL-based planner are overkill and can lead to rapid changes in velocity that do not provide any benefit. Although reworking the training process to penalize uneven motion may lead to improved behavior, it is unclear how the two opposing criteria would reflect on general performance. In addition, conceiving a new training process and designing an improved reward function is an arduous effort that is often subject to trial and error.

Alternatively, one can simply recognize that an RL-based planner performs better when confronted with an unexpected or dynamic obstacle, whereas a classical planner performs better when the robot simply needs to track the global plan. In this context, a pragmatic solution is to conceive a decision tree that recognizes the current situation and switches to using the planner known to produce a better solution. The existing works (Dey et al. (2023); Raj et al. (2024)) have proposed learning the switching criteria with a neural network, which requires further training and may suffer from the typical shortcomings of the learning-based approaches, such as generalizability.

In this paper, we propose a simple hybrid planner that detects if the global plan is obstructed by an unexpected obstacle and picks the solution provided by a (more responsive) RL-planner. Otherwise, it takes the solution provided by a classical planner. Our work demonstrates an effective deployment of an RL-based approach in the real-world, through targeted application. We demonstrate via experiments that this hybrid approach responds well in the obstructed case while maintaining smooth performance in the non-obstructed case.

## 2 Related Work

Local planners play an important role in obstacle avoidance and have been a topic of interest for a long time (Sanchez-Ibanez et al. (2021)). Classical planning approaches, which do not employ learning, are widely used across robotics applications. Reactive replanning (Fox et al. (1997); Rösmann et al. (2017)), artificial potential field (Bin-Qiang et al. (2011)), and fuzzy logic-based approaches (Yan & Li (2016)) are examples. One such widely used method, proposed by Fox et al. (1997) and called Dynamic Window Approach (DWA) planner, uses reactive replanning and has been frequently used as the baseline planner by the Robot Operating System (ROS) navigation stack (navigationros). Because of its widespread use and availability in open-source community, ROS implementation of

DWA has often been used as the baseline, despite the algorithm being relatively old. For this reason, we baseline our results to DWA.

An alternate way to design a local planner is to learn the system model using data and fine-tune the learning model in a new environment. Such learning-based approaches have been introduced in the past few years and have been growing rapidly in number. A deep reinforcement learning (DRL) framework is often used for training in such approaches as it allows the robot to interact with the environment without needing data collection and annotation (Van Dinh et al. (2017); Güldenring (2019); Güldenring et al. (2020); Liu et al.; Kästner et al. (2021); Patel et al. (2021); Nakhleh et al. (2023)). The framework proposed by Güldenring (2019), which uses 2D local map and waypoints from the global plan for state representation, was used as the base for the development of many subsequent works. In Nakhleh et al. (2023), we studied and compared classical planners and different RL network architectures and proposed a method that used a polar representation of the costmap in state representations. This network, named SACPlanner, was trained in a simulation environment and tested on live robots. SACPlanner outperformed other approaches, including DWA, in safely avoiding collisions with static and dynamic obstacles. The practical result was a more responsive planner, but slower and jerky motion caused by the robot trying to move cautiously even when the path ahead was clear. Arguably, this behavior can be improved with training in a higher-fidelity simulation environment, but at the risk of breaking other desirable properties achieved during the original training.

APPL framework by Xiao et al. (2022) addresses the shortcomings in a classical planner by tuning its parameters from demonstration, interventions, and evaluative feedback from human users. However, reliance on classical planner also means that its limitations, such as limited planning horizon, will limit the eventual solution as well. Hence, in some scenarios, a learning-based planner may be more beneficial than a well-tuned classical planner.

One way to get the benefits of different types of planners is to use an ensemble of methods with user-defined control. The use of such *hybrid* planning strategies to harness both classical and learning-based approaches is a fairly recent development (von Rueden et al. (2020)). Existing work in the literature has explored both hybrid robotic planners consisting of classical approaches (Orozco-Rosas et al. (2019)) and planners using learning-based approaches (Lu et al. (2020)). Existing hybrid planners combining classical and learning-based approaches lie in the middle of this spectrum and aim to combine the model-based classical approaches and data-based learning approaches by switching between them.

Almadhoun et al. (2021) use heuristics-based criteria to switch between a classical and a learning-based approach to generate viewpoints for 3D reconstruction. Linh et al. (2022) and Dey et al. (2023) study ground robot navigation but they rely on neural networks for learning and focus on high-level planning. Raj et al. (2024) also proposed a neural network-based switch, but they focused on social navigation only. In contrast, our work contributes towards the development of a local planner that uses a hybrid approach that combines classical and learning-based methods. We design a heuristics-based logic for switching between a DWA planner and SACPlanner, enjoying the benefits of both. This hybrid planner exhibits a superior performance with a simple design which forgoes the need to train another neural network for switching.

## 3 Preliminaries

The local planner/controller is responsible for generating the velocity vector that makes progress toward the goal or the next waypoint. Some implementations explore the velocity space and score candidate velocities based on forward simulation in the configuration space (which, strictly speaking, makes them planners), whereas others solve a constrained optimization problem that maps the state to an action (which, strictly speaking, makes them controllers). These planners/controllers can either generate motion in the velocity space and leave it to a lower-level motion controller to generate the actuation, or directly solve for actuation. A motion controller (if present separately from the local planner/controller) generates the actuation that delivers the desired velocity vector.

In this paper, we focus on local planning/control in velocity space, and for simplicity, we use the term "local planner" to mean any subsystem that generates the desired velocity vector based on the present robot configuration (specifically, the robot pose) and the state of the surrounding environment (specifically, the next waypoint pose, goal pose, and perception of obstacles). In the following subsections, we describe the classical and learning-based local planners used in our work: DWA and SACPlanner, respectively.

### 3.1 Dynamic Window Approach (DWA)

The DWA planner generates a set of admissible velocities, which are the velocities that can be reached given the present velocity and the robot's dynamic constraints (e.g., acceleration limits). For each admissible velocity, DWA performs a forward simulation to calculate the resulting trajectory should the robot use this velocity. Finally, each simulated trajectory is scored, and the one with the lowest cost is selected. The objective function reflects progress towards the goal, clearance from obstacles, adherence to the global plan (distance to the waypoint), and twirling.

DWA considers the robot's dynamics when constructing the set of admissible velocities to search. It is these dynamic constraints, such as acceleration limits, that lead to smooth motion. The overall motion is a series of arcs determined by the angular and linear velocity, where each planning step produces one such arc. If there are no obstacles on the path, the planner will pick the arc that best advances the robot toward the next waypoint, as the distance from the global plan is part of the cost function. In an obstacle-free environment, the selection of the best velocity will be a balance between sticking to the global plan (advancing to the nearest waypoint) and advancing toward the goal (cutting corners in the global plan to reach the goal sooner). Parameters allow the user to tune the planner to balance between these behaviors. While this single-arc planning works well in general situations, scenarios requiring complex velocity profiles, as described below, may make DWA ineffective.

If there is an obstacle in the path, the obstacle-distance component of the cost function will start to dominate, and the arcs that point away from the global path will have a lower cost, consequently making the robot deviate from the global plan or the goal. As the robot steers away, the plan-distance and goal-distance components of the cost function will equalize, and the robot will gravitate back to the plan. Three possible scenarios may follow: 1) The robot may have made sufficient forward progress that the next waypoint is behind the obstacle, in which case the local planner will return the robot to the path determined by the global plan; 2) The robot may turn back toward the obstacle and need to steer away again, this time in a more difficult situation due to obstacle proximity; 3) The global planner may trigger and generate a new set of waypoints that will guide the robot around the obstacle.

Ideal local planners should always result in the first case, which would enable them to deal with obstacles independently. The second case can often lead to a live-lock, which manifests as a robot approaching an obstacle and indecisively oscillating without making progress. In some cases, collisions may occur due to sensor limitations. For example, in our experiments, we observed collisions because the LiDAR sensor we used has a minimum-range distance. Once the robot gets too close to an obstacle, the reflections are not registered, and the robot charges into the obstacle. Augmenting the robot with a second, short-distance sensor to prevent these collisions resulted in the previously described live-locks.

We argue that these shortcomings are direct consequences of the single-arc motion planning that DWA uses. Successful obstacle avoidance requires three consecutive arcs, as shown in green in Fig. 1. The first arc pushes the robot away from the obstacle, the second sends it back on track once the obstacle has been successfully navigated, and the third realigns the direction with the plan. The DWA planner simply does not explore the space beyond one velocity vector, and longer simulation time merely extends the arc into space that is not relevant for evaluating the motion. We confirmed this through a series of experiments, tuning one parameter at a time while tracing the DWA code to

find the root cause. All tests pointed to the lack of visibility into the subsequent arcs that may follow the one being scored.

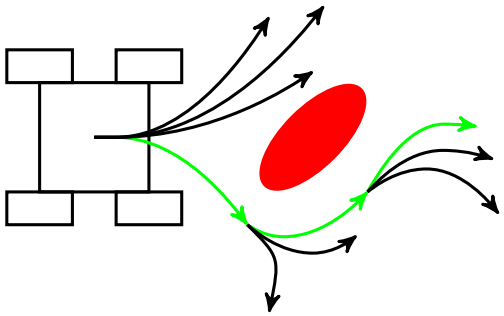


Figure 1: Confronting an obstacle in a series of arc-motions.

Extending the planner to explore a series of velocity vectors scales exponentially with the number of composite arcs to be explored. The sequence of arcs shown in green in Fig. 1 successfully navigates around the obstacle; however, to select it, all three arcs in the sequence must be scored. At each step, there are multiple candidate arcs (shown in black) that must also be scored to find the optimal path around the obstacle.

The third case is commonly used in practice to counter the previously described problem. Replanning at the global level is achieved either by running the global planner periodically at a low rate (e.g., once every few seconds) or by having "patience" timers built into the navigation stack that trigger the global planner when deemed necessary. Careful tuning of cost-function weights, timer values, and other constraints can result in satisfactory and safe performance of the navigation stack; however, this process is arduous, and practitioners often resort to trial and error.

The more cases the local planner can handle independently, the more robust the navigation stack will be when assistance from the global planner is enabled. In our evaluation, we disallow global replanning because we are interested in the performance of the local planner alone, rather than the entire navigation stack. This results in collision-avoidance performance that some practitioners may find surprisingly poor; however, this is due to confusing the performance of the complete navigation stack with the performance of the local planner in isolation.

### 3.2 SACPlanner

SACPlanner, Nakhleh et al. (2023), is an RL-based planner that outperforms DWA in challenging situations and successfully solves the problem described in Section 3.1. An intuitive explanation is that the arc motion it selects is statistically the most likely to be the correct first step in the chain of velocity vectors that will avoid the obstacle and put the robot back on the planned path. There is no velocity-space exploration, and although a single compute step is more complex, it eliminates the problem of exponential scaling.

SACPlanner uses a polar representation of the local costmap as the input to the neural network (see Fig. 3) and outputs an angular and linear velocity pair as the action for the robot. It uses the Soft Actor-Critic (Haarnoja et al. (2018a)) method for training with a mixture of dense and sparse rewards that quantify the robot's progress towards the goal and collision-avoidance, similar to DWA's objective function. Even though it is trained in a simulation environment, it generalizes well and using polar representation of the local costmap as the state helps in sim-to-real transfer without fine-tuning. It demonstrated that a real robot can successfully execute PointGoal navigation in complex mazes and with unexpected obstacles, whereas DWA typically ends up in a state from which it does not make meaningful progress toward the goal or in some cases collides. We have experimentally determined that when the collision occurs, it is typically due to the sensor limitation. Namely, the LiDAR has the minimum range below which it becomes "blind". Whenever the collision occurred, it

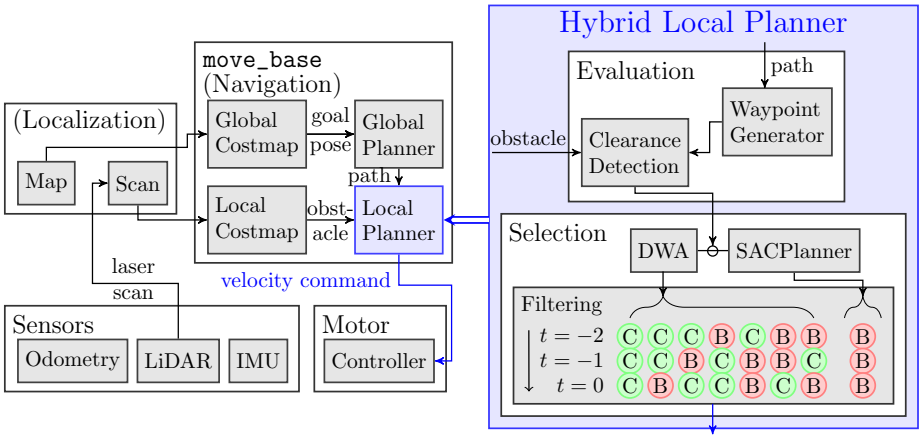



Figure 2: ROS framework and the architecture of our hybrid local planner.

would be because the DWA planner pushed the robot too close to the obstacle to provoke the sensing problem. We believe that if the sensing were augmented to resolve this problem the problem would simply morph into stalling the robot in front of the obstacle. SACPlanner, on the other hand, never brought the robot into such a situation and successfully avoided the obstacles despite the sensing limitation.

A learning-based planner effectively retains the mapping between the input and the output as network weights. This results in SACPlanner potentially learning how to behave in a conservative fashion to safely avoid obstacles. Whereas, DWA is limited to executing motion on circular arcs, SACPlanner can traverse complex trajectories. However, as SACPlanner looks at the costmap in an instant only, the robot's motion is jerky and it moves at a slower speed, making it inefficient even when there is no obstacle ahead.

These two planners represent two seemingly contrasting planning approaches. Choosing one planner from these is essentially a tradeoff between *smoothness* and *responsiveness*. While DWA is more suitable for moving on a static map, SACPlanner is better equipped for successful navigation in complex and dynamic environment. We use this idea to propose a hybrid approach that uses both planners for safer and more efficient planning. This also enables a judicious application of a RL-based planner, SACPlanner in this context, to the real-world scenarios, without the need to fine-tune the network for Sim2Real transfer.

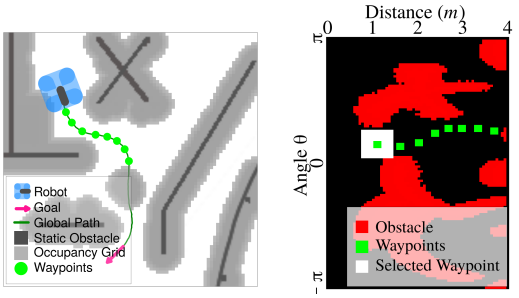
## 4 Hybrid Local Planner

We propose a hybrid local planning approach that combines the benefits of a classical planner and a learning-based planner. Specifically, we run DWA and SACPlanner in parallel and switch between them based on the clearance ahead of the robots. Fig. 2 shows the architecture of our implementation. The box labeled `move_base` comes from standard ROS navigation stack and we modified the local planner plugin to include the DWA code verbatim from the ROS navigation stack, the SACPlanner implementation, along with the code that implements the switching policy. This is illustrated by the box on the right labeled Hybrid Local Planner.

### 4.1 Waypoint Generation

First, we use the method proposed by Güldenring (2019) to find waypoints on the local map. We use the waypoints both to decide which local planner to use and also to create the goal in case the SACPlanner is selected. To generate the waypoints, the global plan leading to the goal, generated with Dijkstra's algorithm, is downsampled and a fixed number of waypoints, 8 in our case, are

selected on the local costmap, as shown in Fig. 3(a). This set of waypoints helps the robot align with the global plan and thus also avoids local minima. The first waypoint not on the obstacles is fed to SACPlanner as the goal in the polar image as Fig. 3(b).

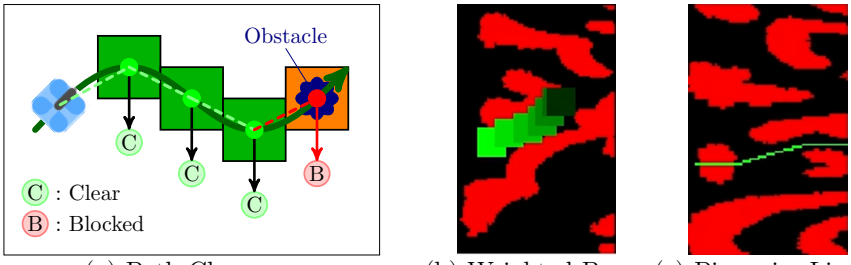



(a) Local Costmap (b) Polar Representation

Figure 3: Waypoint generation.

### 4.2 Clearance Detection

To switch between the planners, the robot needs to determine the clearance ahead. In order to enable an early response, we find if the path without any dynamic obstacle can be traversed without collision. We use the waypoints generated on the local costmap for clearance detection. We check if this path is obstructed anywhere on the local map. If the whole path is unobstructed, we consider the path to be *clear*. Otherwise, this path is considered as *blocked*. Fig. 4(a) demonstrates this approach. The clearance detector can be defined as weighted boxes around the waypoints as in Fig. 4(b). But the size of the box should be tuned since a smaller box can miss obstacles residing in between gaps, whereas a bigger box cannot get through a narrow pathway smoothly. To avoid this, we create a piecewise linear trajectory to approximate the path the robot would have followed if there were no dynamic obstacles in the environment. The path shown in the example Fig. 4(c) is detected as not clear since the initial part of the trajectory is blocked by an obstacle (shown as a red blob).




(a) Path Clearance (b) Weighted Boxes (c) Piecewise Linear

Figure 4: Clearance detection.

### 4.3 Filtering

Noise in the sensor data could result in the clearance detector rapidly flip-flopping between the two planners if only the latest clearance is used for planner selection. For stabilization, the switch should take place only when we are confident about the presence of an obstacle on the path. The typical way to tackle noise in such a situation is to check the likelihood $\mathcal{L}(b|O_{t-n:t})$ of the path being blocked ($b$) based on the past $n$ observations ($\mathcal{O}$) till the current time $t$. If the likelihood of obstacles is higher than a user-defined threshold $\tau$, we consider the path to be blocked.

We implement this strategy as a filter that keeps track of the last $n = 3$ path clearance statuses from the detector. If all the statuses indicate a blocked path, we use the SACPlanner, effectively using $\tau = 1$. Otherwise, DWA is used. This scheme is visualized in the *Filtering* step (right bottom box) in Fig. 2. This design helps in using the SACPlanner when the sensors strongly indicate the presence of an obstacle on the path and results in efficient navigation as the comparatively smoother and faster approach, DWA, is used most of the time and the switching occurs only if necessary.

We use Robot Operating System (ROS) to implement this pipeline in C++ and Python. Our approach runs DWA and SACPlanner in parallel and switches between them by using the velocity prescribed by the selected planner.

## 5 Implementation Details

### 5.1 SACPlanner

SACPlanner, Nakhleh et al. (2023), is a Reinforcement Learning (RL) based planner with a state space $\mathcal{S}$, an action space $\mathcal{A}$, and a reward function $R(\cdot, \cdot)$. The actions are simply the linear/angular velocity pairs $(v, \omega)$. For the state space, it uses an image representation that allows the RL machinery that has been developed for video games. Specifically, the RL state is an image that combines a goal point and all the obstacles that are either derived from the static map or sensed by LiDAR.

The goal point is one of the waypoints already discussed in Section 4. In particular, it selects the first waypoint that does not coincide with an obstacle. It combines this waypoint with the Occupancy Grid representation of the ROS costmap (that represents the nearby obstacles). It then creates a polar representation of the waypoint and obstacles, where the horizontal axis represents the distance from the robot and the vertical axis represents the angle. (See Fig. 3(b) for an example.)

Figure 5: Dummy training environment (left) and the associated polar costmap (right).

To train SACPlanner it is convenient and faster to train it offline than in real time, and so we recreated and utilized a simulated "dummy environment". For each training episode, we pick a synthetic obstacle map and place a robot starting point and a waypoint as in Fig. 5 (left). The episode is successful if the robot reaches the waypoint. The RL state during the training is the associated polar costmap, as described above and shown in Fig. 5 (right).

SACPlanner is trained using a Soft Actor-Critic (SAC) approach (Haarnoja et al. (2018b;c)), where the actor is a policy function and the critic evaluates the actor-value function. SAC augments the standard RL objective with an additional entropy maximization term. Additionally, RAD (Laskin et al. (2020)) and DrQ (Kostrikov et al. (2020)) methods are used to apply a variety of image augmentations when training the actor/critic functions.

The reward function $R(s, a)$ for taking action $a$ in state $s$ is defined as follows. 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 $s'$.

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

where $R_{\max}$ is the reward/penalty for reaching the waypoint or hitting an obstacle, and $G(s')$ is the product of a truncated Gaussian kernel centered at the robot location and the occupancy grid in state $s'$. (The kernel is represented by the green square in Fig. 5.) It incentivizes direct navigation by doubling the penalty for moving away from the waypoint vs. moving towards it. After 10000 training episodes, SACPlanner achieves a 98% episode success rate.

### 5.2 Running the planners in parallel

To implement the hybrid planner we used the `move_base` ROS package (ros). We instantiate three planners using its base planner class: (1) DWA, (2) SACPlanner, and (3) Hybrid Planner. While the first two compute the appropriate velocity profile, only the latter can send the velocity commands to the motion controller. The hybrid planner calls both DWA and SACPlanner functions for its planning functions, effectively running them in parallel. In the output function, responsible for generating the velocity vector, the planner runs the decision logic described in Section 3.2 and publishes velocity computed by the selected planner only.

## 6 Experiment Setup

(a) Unix maze testbed (b) Doorway (c) Cardboard (d) Approaching (e) Crossing

Figure 6: Real experimental environment and 4 test case scenarios (C1-4) from left to right.

Our experimental setup is similar to Nakhleh et al. (2023) for fair comparison. We run experiments using a ClearPath Robotics Jackal robot (cle) in an indoor facility with an open room and a maze with narrow pathways and tight corners, as shown in Fig. 6a. We refer to the maze as the *UNIX maze room* after the letters that constitute the walls inside the maze. In the discussion ahead, we also refer to these letters to indicate the location of the experiment. The robot uses a Velodyne LiDAR running at 10Hz for perception and the planner runs at 5Hz (half the sampling rate of the LiDAR). We study different challenging scenarios in a known map as follows:

**(C1) Obstacle-Free Intricate Trajectory:** This task evaluates if the robot is able to traverse on a serpentine trajectory passing through a narrow doorway. Moving on this trajectory requires that the robot make a $180°$ turn. For this setup, we move the robot from Room I to Room N through a narrow doorway as shown in Fig. 6b. Successful traversal requires that the robot closely follows the global plan on the known map. The challenge for the local planners lies in adjusting their speed timely while accounting for the inertia to avoid collision with the walls.

**(C2) Unexpected Static Obstacle on Path:** In this case, we test if the robot is able to react well to an unexpected object on the path that appears after the global planning is done and stays at a fixed location for the rest of the experiment. This experiment is realized by moving the robot between Room I and Room X, as shown in Fig. 6c. Here we use a life-sized cardboard cutout of a person as

the static obstacle and place it on the robot's global path after the robot starts moving. This setup is similar to *Doorway* setting in Raj et al. (2024). Successful execution requires that the robot moves past the obstacle from the side.

**(C3) Dynamic Obstacle on Path:** Here we test the robot's ability to dynamic obstacles on the robot's global path. For this, we move the robot in a straight line in an open area and a pedestrian walks quickly toward the robot after the robot starts moving on the global path, in a straight line. An obstacle moving at a high speed makes it difficult for the local planner to react in time as the obstacle only shows up after it has entered the robot's local map and keeps changing the location. This situation is shown in Fig. 6d and is similar to the *Frontal* setting in Raj et al. (2024). To achieve success in this case, the robot must react early and back up or move around the pedestrian, or else it will collide with the pedestrian.

**(C4) Dynamic Obstacle Crossing the Path:** While C3 checks the situation when the dynamical obstacle moves directly towards the robot, here we test if the robot can react well when a pedestrian crosses the robot's straight line path perpendicularly. Fig. 6e shows this test case. This is similar to the *Intersection* case in Raj et al. (2024). In this situation, even if the robot observes the pedestrian on its local map, it may not react in time as the obstacle is not yet on the global path. A successful execution requires the robot to back up to turn away from the pedestrian before moving ahead.

We compare the hybrid planner with DWA and SACPlanner across all these situations for 10 runs for C1, C2, and C3, and for 3 runs for C4. In C1 and C2, we also switch the start and goal location for half of the runs. As we focus on task efficiency, we compare the average distance traversed, velocity, time taken to navigate, and the number of collisions (in percentage) for each planner.

## 7 Results

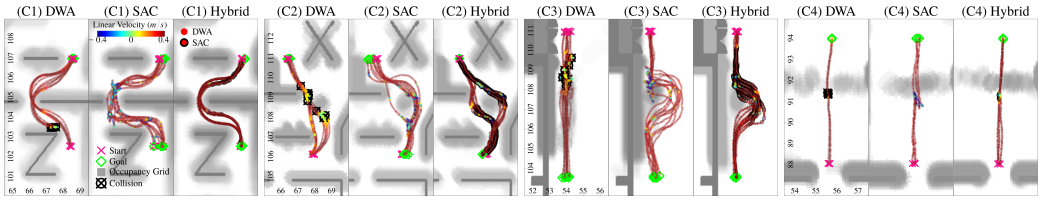



Figure 7: Trajectory comparison between DWA, SACPlanner vs. Hybrid planner agent for each test case.

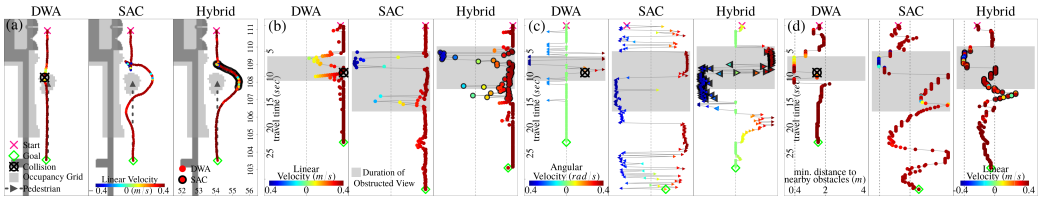



Figure 8: Trajectory comparison between DWA, SAC and Hybrid planners based on logs from the scenario (C3).

The robot trajectories for each of C1-C4 are shown in Fig. 7. We denote the start and goal along with the collision points. The color of the trajectory represents linear velocity and the circles with a thick black border represent where SACPlanner has been used for the hybrid planner. We also show the Occupancy Grid values in gray (taken from the map and the LiDAR). For C2-C4 the gray shading captures all the positions of the unexpected obstacle over time. The three local planners have qualitatively different behavior. DWA collides with the walls or obstacles in all cases except a few in C1. SACPlanner allows the robot to circumnavigate the obstacles but it results in the robot

moving slowly, even with negative velocity in some cases, and usually results in a long detour. In each case, the hybrid planner helps the robot avoid obstacles successfully, while moving on a smooth trajectory with high speed, making it more suitable than either individual planner.

Table 1 summarizes the quantitative comparison averaged over 10 runs for C1-C3 with standard deviation provided in below parenthesis. (For brevity we refer to SACPlanner as SACs in the table.) The hybrid planner is faster than both DWA and SACPlanner, as shown by the higher average speed. Collisions when DWA is used, result in the robot covering a shorter distance without success. SACPlanner has the same success rate as the hybrid planner, but the hybrid planner results in a relative improvement of **26%** in the navigation time with **18%** shorter path length. Notably, our planner exhibits safe and efficient navigation in situations similar to prior works (Raj et al. (2024)), without the need to learn when to switch with a neural network.

Table 1: Summary statistics of trajectories from test cases.

| | (C1) | | | (C2) | | | (C3) | | |
|---|---|---|---|---|---|---|---|---|---|
| | DWA | SAC | Hybrid | DWA | SAC | Hybrid | DWA | SAC | Hybrid |
| Time (s) | 21.60 (0.61) | 37.20 (12.80) | **21.10** (0.40) | 30.70 (15.70) | 28.50 (6.70) | **23.60** (3.13) | 27.50 (7.04) | 33.10 (2.24) | **27.10** (2.03) |
| Distance (m) | **6.26** (0.93) | 10.70 (1.95) | 7.67 (0.13) | **5.47** (0.78) | 8.57 (1.56) | 7.41 (0.88) | **8.77** (2.35) | 10.80 (0.87) | 9.40 (0.52) |
| Speed (m/s) | 0.31 (0.04) | 0.29 (0.03) | **0.39** (0.01) | 0.22 (0.09) | 0.30 (0.02) | **0.31** (0.01) | 0.32 (0.05) | 0.33 (0.01) | **0.36** (0.02) |
| Collision | 50% | 0% | **0%** | 100% | 0% | **0%** | 100% | 0% | **0%** |

To understand more deeply why the hybrid planner performs better, we show in Fig. 8 the behavior of each planner in a single run from the test case (C3). The beginning and ending behavior of the hybrid planner is closer to a straight line, since DWA is selected using full-speed (dark red) linear velocities as in Fig. 8a, 8b. The shaded area in Fig. 8c-8d represents the duration of time when LiDAR first captures the pedestrian in its view in the polar costmap until he stops walking at the location $x = 54$m, $y = 108$m. From the overall travel time, the hybrid planner gets the robot to the goal faster than SACPlanner without any collisions. The reaction time (in seconds) to begin turning starting from when the robot first enters the shaded area, 4.05s for hybrid planner, 5s for SACPlanner and 5.99s for DWA planner. In addition, the hybrid planner gets around the pedestrian about 3.5 seconds faster than SACPlanner (8.36s $<$ 11.8s). The transition in rotational velocities is much smoother in the hybrid case since it reverts to DWA after passing around the pedestrian as in Fig. 8c. Moreover, when the robot is far from the pedestrian the angular velocity is zero (green). This explains how the hybrid planner almost eliminates the jerky motion caused by SACPlanner. Fig. 8d shows the distance to the nearest 'front obstacle' (within $\pm\frac{\pi}{4}$rad range from the current yaw). The hybrid planner manages both safe and efficient distance during the whole travel time.

The results highlight that the hybrid planner makes appropriate use of both planners for navigation in various scenarios. It moves smoothly and quickly through clear areas and is responsive in face of obstacles discovered along the path. This behavior is also safer, both for the robot and for the humans acting as dynamic obstacles.

## 8 Discussion and Future Work

We present a hybrid local planner that combines DWA, a classical planning method, and SACPlanner, a learning-based planning approach. Experiments on a ClearPath Jackal robot in various situations show that the proposed approach is safer and more efficient than the two constituent planners, showing a significant improvement in navigation time without any collision. The design of our switch forgoes the need to collect data and train another neural network, making it more suitable than learning-based switching from real-world deployment.

Our analysis shows that a purely RL-based planner trained in simulation may suffer from shortcomings owing to imperfections in the simulation. Even if trained in a realistic environment, choice of inputs and the reward design may produce sub-optimal solution, fit for specific scenarios. Our work shows how such planners still can still be helpful through judicious application along with classical approaches.

In this work, we focus on a heuristics-based approach to define the criteria for switching between the planners. Future work will explore more sophisticated approaches. A drawback of the hybrid approach is that the shortcomings of the constituent planners appear when the hybrid planner uses them. An example of this would be some jerky motion of the robot, owing to the SACPlanner, while the robot tried to avoid the obstacle. In the future, we intend to work on improving the constituent planners to further improve the overall performance of the hybrid planner.

## References

Jackal unmanned ground vehicle. ClearPath Robotics, Product Datasheet. Available at: https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/.

ROS.org move_base, URL: http://wiki.ros.org/move_base.

Randa Almadhoun, Tarek Taha, Lakmal Seneviratne, and Yahya Zweiri. Multi-robot hybrid coverage path planning for 3d reconstruction of large structures. *IEEE Access*, 10:2037–2050, 2021.

Ye Bin-Qiang, Zhao Ming-Fu, and Wang Yi. Research of path planning method for mobile robot based on artificial potential field. In *2011 International Conference on Multimedia Technology*, pp. 3192–3195. IEEE, 2011.

Sombit Dey, Assem Sadek, Gianluca Monaci, Boris Chidlovskii, and Christian Wolf. Learning whom to trust in navigation: dynamically switching between classical and neural planning. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5235–5242. IEEE, 2023.

Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

Ronja Güldenring. Applying deep reinforcement learning in the navigation of mobile robots in static and dynamic environments. 2019. URL https://api.semanticscholar.org/CorpusID:198947279.

Ronja Güldenring, Michael Görner, Norman Hendrich, Niels Jul Jacobsen, and Jianwei Zhang. Learning local planners for human-aware navigation in indoor environments. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6053–6060. IEEE, 2020.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018a.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018b.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018c.

Linh Kästner, Xinlin Zhao, Teham Buiyan, Junhui Li, Zhengcheng Shen, Jens Lambrecht, and Cornelius 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*, pp. 1213–1220. IEEE, 2021. DOI: 10.1109/IROS51168.2021.9636039. URL https://doi.org/10.1109/IROS51168.2021.9636039.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *Advances in neural information processing systems*, 33: 19884–19895, 2020.

KU Linh, Johannes Cox, Teham Buiyan, Jens Lambrecht, et al. All-in-one: A drl-based control switch combining state-of-the-art navigation planners. In *2022 International Conference on Robotics and Automation (ICRA)*, pp. 2861–2867. IEEE, 2022.

Lucia Liu, Daniel Dugas, Gianluca Cesari, Roland Siegwart, and Renaud Dubé. Robot navigation in crowded environments using deep reinforcement learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5671–5677. IEEE.

Yan Lu, Weihong Wang, and Lilin Xue. A hybrid cnn-lstm architecture for path planning of mobile robots in unknow environments. In *2020 Chinese Control And Decision Conference (CCDC)*, pp. 4775–4779. IEEE, 2020.

Khaled Nakhleh, Minahil Raza, Mack Tang, Matthew Andrews, Rinu Boney, Ilija Hadžić, Jeongran Lee, Atefeh Mohajeri, and Karina Palyutina. SACPlanner: Real-world collision avoidance with a soft actor critic local planner and polar state representations. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9464–9470. IEEE, 2023.

navigationros. ROS navigation package. Web Page (ROS documentation), 2020. available at http://wiki.ros.org/navigation.

Ulises Orozco-Rosas, Kenia Picos, and Oscar Montiel. Hybrid path planning algorithm based on membrane pseudo-bacterial potential field for autonomous mobile robots. *IEEE Access*, 7: 156787–156803, 2019.

Utsav Patel, Nithish K Sanjeev Kumar, Adarsh Jagan Sathyamoorthy, and Dinesh 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)*, pp. 6057–6063. IEEE, 2021.

Amir Hossain Raj, Zichao Hu, Haresh Karnan, Rohan Chandra, Amirreza Payandeh, Luisa Mao, Peter Stone, Joydeep Biswas, and Xuesu Xiao. Rethinking social robot navigation: Leveraging the best of two worlds. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16330–16337. IEEE, 2024.

Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Kinodynamic trajectory optimization and control for car-like robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5681–5686. IEEE, 2017.

Jose Ricardo Sanchez-Ibanez, Carlos J Perez-del Pulgar, and Alfonso García-Cerezo. Path planning for autonomous mobile robots: A review. *Sensors*, 21(23):7898, 2021.

Nguyen Van Dinh, Nguyen Hong Viet, Lan Anh Nguyen, Hong Toan Dinh, Nguyen Tran Hiep, Pham Trung Dung, Trung-Dung Ngo, and Xuan-Tung Truong. An extended navigation framework for autonomous mobile robot in dynamic environments using reinforcement learning algorithm. In *2017 International Conference on System Science and Engineering (ICSSE)*, pp. 336–339. IEEE, 2017.

Laura von Rueden, Sebastian Mayer, Rafet Sifa, Christian Bauckhage, and Jochen Garcke. Combining machine learning and simulation to a hybrid modelling approach: Current and future directions. In *Advances in Intelligent Data Analysis XVIII: 18th International Symposium on Intelligent Data Analysis, IDA 2020, Konstanz, Germany, April 27–29, 2020, Proceedings 18*, pp. 548–560. Springer, 2020.

Xuesu Xiao, Zizhao Wang, Zifan Xu, Bo Liu, Garrett Warnell, Gauraang Dhamankar, Anirudh Nair, and Peter Stone. Appl: Adaptive planner parameter learning. *Robotics and Autonomous Systems*, 154:104132, 2022.

Yupei Yan and Yangmin Li. Mobile robot autonomous path planning based on fuzzy logic and filter smoothing in dynamic environment. In *2016 12th World congress on intelligent control and automation (WCICA)*, pp. 1479–1484. IEEE, 2016.