MultiHyRL: Robust Hybrid RL for Obstacle Avoidance against Adversarial Attacks on the Observation Space

Jan de Priester
jadeprie@ucsc.edu
University of California Santa Cruz
Department of Electrical and Computer Engineering

Zachary I. Bell
zachary.bell.10@us.af.mil
Air Force Research Lab
Munitions Directorate

Prashant Ganesh
prashantganesh@episci.com
EpiSci

Ricardo G. Sanfelice
ricardo@ucsc.edu
University of California Santa Cruz
Department of Electrical and Computer Engineering

Abstract

Reinforcement learning (RL) holds promise for the next generation of autonomous vehicles, but it lacks formal robustness guarantees against adversarial attacks in the observation space for safety-critical tasks. In particular, for obstacle avoidance tasks, attacks on the observation space can significantly alter vehicle behavior, as demonstrated in this paper. Traditional approaches to enhance the robustness of RL-based control policies, such as training under adversarial conditions or employing worst-case scenario planning, are limited by their policy's parameterization and cannot address the challenges posed by topological obstructions in the presence of noise. We introduce a new hybrid RL algorithm featuring hysteresis-based switching to guarantee robustness against these attacks for vehicles operating in environments with multiple obstacles. This hysteresis-based RL algorithm for coping with multiple obstacles, referred to as MultiHyRL, addresses the 2D bird's-eye view obstacle avoidance problem, featuring a complex observation space that combines local (images) and global (vectors) observations. Numerical results highlight its robustness to adversarial attacks in various challenging obstacle avoidance settings where Proximal Policy Optimization (PPO), a traditional RL method, fails.

1 Introduction

To develop the next generation of autonomous vehicles that can robustly and safely navigate the physical world, reinforcement learning (RL) has shown a lot of promise (Everett et al., 2018; Cimurs et al., 2020; Choi et al., 2021; Feng et al., 2021; Kästner et al., 2021). In Everett et al. (2018); Cimurs et al. (2020); Choi et al. (2021), methodologies for obstacle avoidance are developed that leverage RL and are effective in environments with dynamic obstacles, such as pedestrians. In Feng et al. (2021); Kästner et al. (2021), methodologies are developed for applying RL in challenging environments, such as narrow corridors or highly dynamic environments. One of the main challenges of applying RL safely for safety-critical tasks, such as obstacle avoidance, is the lack of formal robustness guarantees against adversarial attacks.

Adversarial attacks can occur in various forms. Simply stated, attacks can target the state space, such as a change in dynamics like a propeller chipping of a quadrotor, or the observation space, such as a perturbation on the camera image. In this paper, the focus is on robustness against adversarial attacks on the observation space. Specifically, we consider adversarial attacks on vehicle

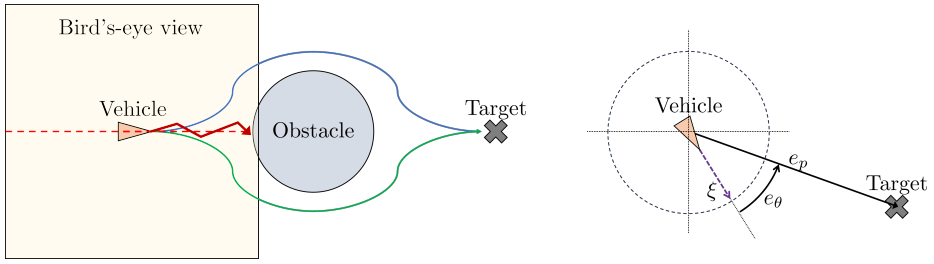


Figure 1: Overview of the bird's-eye view obstacle avoidance setting. The orange arrow depicts the vehicle, the yellow square depicts the bird's-eye view image, the gray circle depicts the obstacle, and the cross depicts the target. The blue and green arrows represent trajectories from the RL policy when the vehicle is above or below the red dashed decision boundary. The red arrow shows the trajectory with small measurement noise. The purple arrow indicates the vehicle's orientation ξ. The black arrows represent the position and orientation errors e_d and e_θ with respect to the target.

problems operating in environments with obstacles. An obstacle introduces a topological obstruction by dividing the navigable space, requiring vehicles to pass the obstacle clockwise or counterclockwise. Employing a discontinuous control strategy allows the vehicle to be directed around the obstacle to its target. However, (arbitrarily) small measurement noise can undermine the global attractiveness of such a discontinuous controller, making the controller lack robustness against (arbitrarily) small measurement noise, see Prieur et al. (2007); Mayhew et al. (2011). To illustrate these robustness issues, consider the problem of steering a vehicle to move past an obstacle so as to reach a target. After successfully training the control algorithm onboard the vehicle, a policy is found that navigates the vehicle to bypass an obstacle and reach a target, rendering the target globally attractive. Figure 1 shows an overview of the found policy: the vehicle steers left when above the decision boundary, illustrated by the red dashed line in Figure 1, and steers right below the decision boundary. However, when the vehicle is near the decision boundary, issues may arise due to noisy observations. For instance, suppose the vehicle is physically above the decision boundary, but the noisy observations may report it to be below it. As a result, the vehicle wrongly turns right. Conversely, if the vehicle is physically below the decision boundary but reported to be above, it would incorrectly turn left. Repetition of this occurrence can cause the vehicle to get stuck in front of the obstacle or drive straight into it, as shown in Figure 1 (see also Section 3.2). This scenario demonstrates that (arbitrarily) small noise can compromise the policy in critical situations. The notion of critical points in the observation space is relevant to other problems where chattering between policies causes undesired behavior. This extends to various RL domains, such as hierarchical RL (Frans et al., 2017; Nachum et al., 2018) and options-based methods (Bacon et al., 2016; Barreto et al., 2021). For example, in a pick-and-place task, a robot might need to place an object into one of two equally distant and suitable boxes. If the robot is positioned at exactly the same distance between the two boxes, even an arbitrarily small amount of noise can cause indecisiveness, leading to chattering between the options and resulting in inefficiencies and delays.

Various approaches have been proposed in the literature to improve the general robustness of RL-based control policies against noisy observations or model parameter alterations. Examples are training in the presence of adversarial attacks (Papernot et al., 2015; Mandlekar et al., 2017; Madry et al., 2018; Tramèr et al., 2018), considering a worst-case scenario (Pinto et al., 2017; Lütjens et al., 2020; Zhang et al., 2020; Everett et al., 2022; Liang et al., 2022), or by using control barrier functions (Emam et al., 2021; Cheng et al., 2023). Notably, a few studies address adversarial attacks targeting the observation space (Lütjens et al., 2020; Zhang et al., 2020; Everett et al., 2022). However, these methods focus on worst-case adversary scenarios for (memoryless) policies with traditional continuous or discrete action spaces. While these methods enhance the general robustness against noisy observations or model parameter alterations, they cannot address the challenges posed by topological obstructions. Specifically, because these methods utilize a continuous or (memoryless) discrete policy parameterization, they fail to prevent the problematic *chattering* behavior as sketched

in the autonomous vehicle example above and shown in more detail in Section 3. Contrary to these works, our approach amalgamates ideas from *hybrid control theory* to overcome the limitations of traditional policies. We introduce a new hybrid RL algorithm, referred to as MultiHyRL, that overcomes these limitations by implementing *hysteresis* switching modeled by a hybrid system. MultiHyRL, described in Section 4, provides a hybrid control policy that is robust against adversarial attacks on the observation space near critical areas for environments with an arbitrary number of randomly located obstacles. MultiHyRL identifies critical points, such as the decision boundary in Figure 1, for a control policy obtained via a standard RL method[1] and uses this policy to separate the state space into overlapping sets, thereby effectively removing the topological obstruction from the state space. Next, new control policies are trained for each overlapping set via a standard RL method, and the policies are combined in a hybrid system that supervises the newly obtained policies and implements hysteresis-based switching by introducing two logic variables. To emphasize the effectiveness of our approach, we consider the bird's-eye view obstacle avoidance problem, described in Section 3, whose observation space is complex with the combination of local (images), shown on the left in Figure 1, and global (vectors) observations, shown on the right in Figure 1. In Section 5, we empirically show the robustness of the MultiHyRL agent compared to the normal agent for various settings.[2]

2 Preliminaries

2.1 Notation

The following notation is used throughout the paper. The n-dimensional Euclidean space is denoted by $\mathbb{R}^n$. The real numbers are denoted by $\mathbb{R}$. The nonnegative real numbers are denoted by $\mathbb{R}_{\geq 0}$, i.e., $\mathbb{R}_{\geq 0} := [0, \infty)$. The natural numbers including 0 are denoted by $\mathbb{N}$, i.e., $\mathbb{N} := \{0, 1, 2, ...\}$. The natural numbers excluding 0 are denoted by $\mathbb{N}_{>0}$, i.e., $\mathbb{N}_{>0} := \{1, 2, ...\}$. The closed unit ball, of appropriate dimension and centered at the origin, in the Euclidean norm is denoted by $\mathbb{B}$. The Euclidean norm of the vector x is denoted by $|x|$. The distance from x to the set nonempty S is denoted by $|x|_S$, which is given by $\inf_{y \in S} |x - y|$. The convex hull of the set S is denoted by $\text{Conv}(S)$. The interior of a set S is denoted by $\text{int}\, S$. The boundary of the set S is denoted by ∂S. The domain of a map f is denoted by $\text{dom}\, f$. The unit circle is denoted by $\mathcal{S}^1$ and is defined as $\mathcal{S}^1 := \{x \in \mathbb{R}^2, |x| = 1\}$. The 4-quadrant inverse tangent is denoted by arctan2. The signum function is denoted by sgn and is defined as $\text{sgn}(\chi) := -1$ if $\chi < 0$ and $\text{sgn}(\chi) := 1$ if $\chi \geq 0$. The identity matrix is denoted by I. The rotation matrix $\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$ is denoted by $\text{Rot}(\theta)$.

2.2 Reinforcement learning framework

Markov decision processes (MDPs) are used as a formalism for Reinforcement Learning (RL) (Puterman, 1994). In an MDP, the learner/controller is referred to as the agent and interacts with an environment. An episode is the system's deployment in an environment from an initial condition and subjected to inputs over a finite or infinite time horizon. The agent's state $z \in \mathcal{Z}$, where $\mathcal{Z} \subset \mathbb{R}^n$ is a set of states, evolves according to its dynamics

$$\dot{z} = f(z, u), \tag{1}$$

where $f : \mathcal{Z} \times \mathcal{U} \to \mathcal{Z}$ and $u \in \mathcal{U}$ is the control input, where $\mathcal{U} \subset \mathbb{R}^m$ is a set of actions. During an episode, the environment yields rewards based on the reward function $R : \mathcal{Z} \times \mathcal{U} \to \mathbb{R}$ to connect specific state-action pairs to reward values. The goal of RL is to find a policy $\pi : \mathcal{Z} \to \mathcal{U}$ that

[1] Proximal Policy Optimization (PPO) (Schulman et al., 2017) is used as the 'normal' RL algorithm for the control policies derived in this paper. Nevertheless, the approach can be applied to other RL methods as well. The details on the used PPO implementation are given in Appendix A.1.

[2] All the simulation files are available at `www.github.com/HybridSystemsLab/MultiHyRL`.

maximizes the return functional

$$G(\tau) := \int_{t=0}^{T} \gamma^t R(z(t), u(t))\, \mathrm{d}t, \tag{2}$$

where $\tau = (z, u) \in \mathcal{Z} \times \mathcal{U}$ is a solution pair to (1), defined for each $t \in [0, T]$, where $T \in \mathbb{R}_{>0}$ is the horizon length, under the policy $\pi \in \Pi$, where Π is the set of possible policies, and $\gamma \in [0, 1)$ is the discount factor that weighs the importance of future rewards.

2.3 Hybrid Systems

A hybrid system $\mathcal{H} = (C, F, D, G)$ is defined as

$$\mathcal{H} : \begin{cases} \dot{x} &= F(x) \quad x \in C \\ x^+ &= G(x) \quad x \in D \end{cases} \tag{3}$$

where $x \in \mathbb{R}^n$ denotes the state variable, x^+ the state variable after a jump, $F : C \to \mathbb{R}^n$ is a function referred to as the flow map, $C \subset \mathbb{R}^n$ is the set of points referred to as the flow set, $G : D \to \mathbb{R}^n$ the jump map, and $D \subset \mathbb{R}^n$ is the jump set. When the state is in the flow set, the state is allowed to evolve continuously and is described by the differential equation defined by the flow map. When the state is in the jump set, the state is allowed to be updated using the difference equation defined by the jump map. In this way, with some abuse of notation, the solution to (3) is given by a function $(t, j) \mapsto x(t, j)$ defined on a hybrid time domain, which properly collects values of the ordinary time variable $t \in \mathbb{R}_{\geq 0}$ and of the discrete jump variable $j \in \mathbb{N}$. The hybrid system $\mathcal{H}$ allows for the combination of continuous-time behavior (flow) with discrete-time behavior (jumps). For more details on hybrid dynamical systems, see Goebel et al. (2012); Sanfelice (2021).

3 Motivation

3.1 Vehicle Dynamics and Bird's-Eye View Problem Formulation

We consider a vehicle evolving on the plane with the state $z = (p, \xi) \in \mathcal{Z} \subset \mathbb{R}^2 \times \mathcal{S}^1$ and dynamics given by

$$\dot{z} = f(z, u) := \begin{bmatrix} u_v \xi \\ u_r \mathrm{Rot}\left(-\frac{1}{2}\pi\right) \xi \end{bmatrix}, \tag{4}$$

where $p = (p_x, p_y) \in \mathcal{P}$ denotes the planar position, with $p_x \in \mathcal{P}_x$ and $p_y \in \mathcal{P}_y$ being the coordinates along the x- and y-axes, respectively. The orientation of the vehicle is $\xi \in \mathcal{S}^1$, as shown in Figure 1. The control input $u = (u_v, u_r)$ consists of $u_v \in [-1, 1]$ controlling the forward velocity and $u_r \in [-1, 1]$ controlling the orientation. In addition, the environment features $N \in \mathbb{N}_{>0}$ circular obstacles with a radius of $r_{\mathrm{ob}} \in \mathbb{R}_{>0}$ centered at $\mathcal{P}_{\mathrm{ob}} = \{p_{\mathrm{ob},1}, p_{\mathrm{ob},2}, \ldots, p_{\mathrm{ob},N}\} \subset \mathcal{P}$, where $p_{\mathrm{ob},i} = (p_{\mathrm{ob},i,x}, p_{\mathrm{ob},i,y})$ is the planar position of obstacle $i \in \{1, 2, \ldots, N\}$. To detect collisions of the vehicle with an obstacle, we define the collision set $\mathcal{C}$ as the set of planar positions p for which the distance to the boundary of an obstacle is zero, that is, $\mathcal{C} := \{p \in \mathcal{P} : |p|_{\mathcal{P}_{\mathrm{ob}}} \leq r_{\mathrm{ob}}\}$.

The problem to solve consists of designing an RL-based controller for the vehicle in (4) to avoid the obstacles and safely reach the set-point position $p^* = (p_x^*, p_y^*) \in \mathcal{P}$ asymptotically. The full plant model of the system, shown in Figure 2, consists of the vehicle's dynamics (4) and a function that provides measurements, or equivalently, observations based on the perturbed state $z' := z + m$, where $z' = (p', \xi') \in \mathbb{Z}$ and $m \in \mathbb{R}^4$ is measurement noise. The observations consist of the planar position error $e_p(p') = |p'|_{p^*}$, the orientation error $e_\theta(z') = \arctan2(p^* - p') - \arctan2(\xi')$ mapped within $-\pi$ to π, and a binary square image $\mathcal{I} \in \{0, 1\}^{n_{\mathrm{res}} \times n_{\mathrm{res}}}$, where 0 corresponds to no obstacle, 1 corresponds to an obstacle, and $n_{\mathrm{res}} \in \mathbb{N}_{>0}$ is the resolution of the image. The image is taken from above, providing a limited *bird's-eye view* of the environment, as shown in Figures 1 and 2. The image is centered on the vehicle's perceived center of mass, which is located at p'.[3] To ensure

[3]The image is based on the perturbed state z'. Robustness against image-level noise, such as blurriness and rotations, is beyond the scope of this paper as the focus is on perturbations that cause chattering behavior.

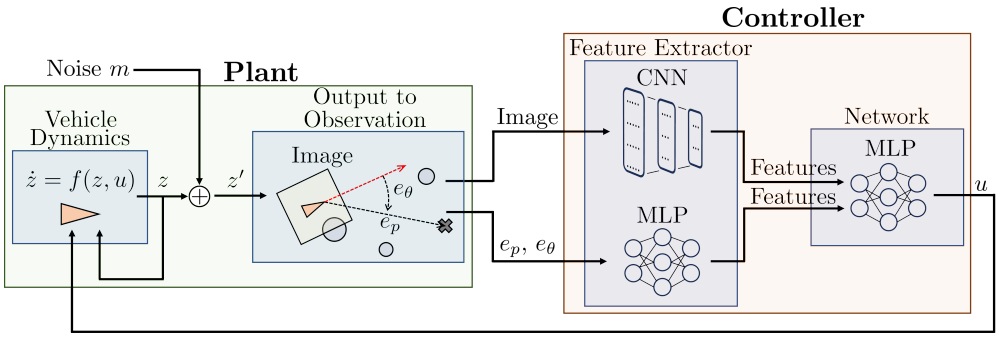


Figure 2: Overview of the closed-loop control architecture for the bird's-eye view obstacle avoidance problem.

consistency of the image with the perceived vehicle's body frame as the vehicle turns, the image is rotated with the angle of ξ' as shown in the '*Output to Observation*' block in Figure 2. Furthermore, the image has an equal width and length of $w_{\text{bev}} \times w_{\text{bev}}$, and a resolution of $n_{\text{res}} \times n_{\text{res}}$ array elements.

The obstacles have the following three properties:
(**O1**) the obstacles are static and have the same radius r_{ob};[4]
(**O2**) the obstacles are disconnected from each other with a spacing between each obstacle greater than $\sqrt{2}w_{\text{bev}}$, such that only one obstacle is visible to the agent for any state $z \in \mathcal{Z}$;
(**O3**) the distance between an obstacle and the set-point position p^* is greater than $\frac{1}{2}\sqrt{2}w_{\text{bev}}$, such that no obstacles are visible to the agent when the agent is at the set-point position $p = p^*$.

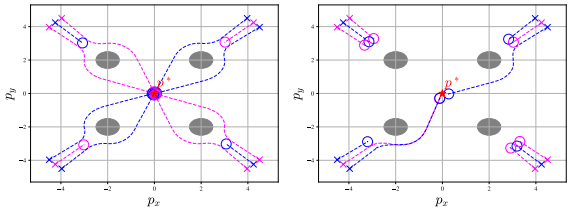
The control architecture, shown in Figure 2, consists of a *feature extractor* and a standard *Multi-Layer Perceptron* (MLP) to map the extracted features to a control policy. An MLP is trained to extract the relevant features from the planar position error e_p and the orientation error e_θ. Concurrently, a *Convolutional Neural Network* (CNN) is trained to extract the relevant features about nearby obstacles from the bird's-eye view image $\mathcal{I}$.[5] The details on the architectures of the CNN and MLPs and their hyperparameters are given in Appendix A.2.

To motivate the agent to navigate around obstacles and reach the set point, a smooth reward function is crafted with the following components three components:
(**R1**) a penalty for deviating from the set-point position p^*, encouraging the agent to reach the set-point position via the shortest path: $-c_1 e_p \in \mathbb{R}_{\leq 0}$, where $c_1 \in \mathbb{R}_{>0}$ is a weight;
(**R2**) a penalty for deviating from the orientation corresponding to the shortest path to the position set point p^* : $-c_2 \min(e_p, c_3)|e_\theta| \in \mathbb{R}_{\leq 0}$, where $c_2, c_3 \in \mathbb{R}_{>0}$ are weights. This component is scaled by $\min(e_p, c_3)$ such that $c_2 \min(e_p, c_3)|e_\theta| \to 0$ as $e_p \to 0$ and $c_2 \min(e_p, c_3)|e_\theta| \to c_2 c_3 |e_\theta|$ as $e_p \to \infty$;[6]
(**R3**) a penalty for when the agent is in observable proximity to an obstacle for any orientation $\xi \in \mathcal{S}^1$ to encourage a safe distance between the agent and the obstacle: $-c_4 B(|p|_{\mathcal{C}})$, where $c_4 \in \mathbb{R}_{>0}$ is a weight. The function B, similar to the one defined in Sanfelice et al. (2006), serves as a barrier function and is defined as $B(\chi) := (\chi - \frac{1}{2}w_{\text{bev}})^2 \ln \frac{w_{\text{bev}}}{2\chi}$ if $\chi \in [0, \frac{1}{2}w_{\text{bev}}]$ and $B(\chi) := 0$ if $\chi > \frac{1}{2}w_{\text{bev}}$. The agent is in observable proximity to an obstacle for any orientation $\xi \in \mathcal{S}^1$ when the distance to an obstacle is less than half the width of the image: $|p|_{\mathcal{C}} \in [0, \frac{1}{2}w_{\text{bev}}]$. Combining these components yields the following reward function, which, in this case, depends on z only:

$$R(z) = -c_1 e_p - c_2 |e_\theta| \min(e_p, c_3) - c_4 B(|p|_{\mathcal{C}}) \quad \forall z \in \mathcal{Z}. \tag{5}$$

[4]The method can be applied to moving obstacles and those with various radii, as demonstrated in Appendix C.
[5]The relevant features for the network in Figure 2 are unknown before training. During training, both the network and feature extractor are trained concurrently, so the meaning of relevant features may change. After training, the extracted features can be analyzed to interpret their meanings.
[6]This scaling is necessary as e_θ is undefined for $p = p^*$. Furthermore, the min saturates the penalty for large values of e_p, such that the orientation penalty does not grow unproportionally large with respect to the other penalties.

3.2 Lack of Robustness



(a) Without noise. (b) With noise $\varepsilon_p = 0.1$.

Figure 3: Visualization of state trajectories (or solutions) for various initial conditions in a four obstacle scenario, with and without measurement noise. The trajectories alternate colors for readability. The gray circles denote obstacles, the $\times$'s denote the initial conditions, the $\circ$'s denote the terminal positions, and the red $\star$ denotes the set-point position p^*.

To demonstrate the lack of robustness in RL-based control policies, a policy for the system described in Section 3.1 is found using the PPO algorithm. Specifically, the control policy is trained for a *canonical setting* with a single obstacle at $p_{\text{ob}} = (-2, 0)$ and a set point at $p^* = (0, 0)$. The observations provided to the agent enable the policy to generalize across various obstacle settings, as long as the obstacles adhere to the properties outlined in Section 3.1. Figure 3 shows the state trajectories for a four-obstacle setting; details on the simulation implementation are provided in Appendix B.4. Figure 3a shows that the policy steers the vehicle clockwise or counterclockwise for a small change in the initial condition (denoted by $\times$). However, the policy fails to pass the obstacle when it approaches the obstacle in the center. This is a limitation of the *continuity* of the policy parameterization, which causes it not to steer away from the obstacle for some states. When an arbitrarily small amount of noise is applied to the perceived position of the vehicle, the policy can mistakenly steer the vehicle clockwise when it should steer counterclockwise and vice versa, inducing "*chattering*" behavior. Figure 3b demonstrates this chattering behavior and the vehicle getting stuck in front of the obstacle due to measurement noise on the measured vehicle position. Details on the simulation are given in Section 5.

In this paper, motivated by such fragility of RL-based algorithms, we develop an algorithm that results in a closed-loop hybrid system whose behavior is robust against those disturbances in the presence of multiple obstacles.

4 Multi Hysteresis-Based RL

A hysteresis-based RL algorithm to guarantee safety and robustness in environments with multiple obstacles and measurement noise is proposed. The algorithm, which we call MultiHyRL, deals with any number of obstacles by implementing a hysteresis switching mechanism that prevents chattering behavior from occurring in the presence of measurement noise. In this section, the MultiHyRL algorithm is discussed.

4.1 MultiHyRL Overview

In simple words, the MultiHyRL algorithm operates as follows:
(**Step 1**) Utilize the RL method of choice, such as PPO, to find an initial control policy;
(**Step 2**) Detect the critical points for the initial control policy; see ($\star$) in Section 4.2. The critical points are the areas in front of each obstacle for which solutions evolve in opposite directions for a small change in the state. For example, the dashed red line in Figure 1 is a collection of critical points;
(**Step 3**) Utilize these critical points to partition the state space into two overlapping sets; see Section 4.3. The union between the overlapping sets covers the whole state space, and the intersection covers the critical points and an area around the set point. By this construction, for each overlapping set, only one path exists that connects the critical points to the set point for each obstacle;
(**Step 4**) Train a new policy for both of these overlapping sets using the chosen RL method. The vehicle's passage around each obstacle is constrained to a single side due to the design of the overlapping sets. Thereby effectively eliminating the topological obstructions. For example, for the scenario depicted in Figure 1, the vehicle is only permitted to navigate around the obstacle via either the

blue trajectory or the green trajectory for each respective overlapping set;
(**Step 5**) Introduce a logic variable to dictate the active policy and a logic variable to indicate the focused obstacle to be avoided. The logic variables change their values only when the system exits the overlapping set corresponding to their current logic value index. This mechanism enables hysteresis switching between the newly acquired policies for each obstacle. With this approach, small measurement noise does not lead to chattering behavior. Instead, the same policy continues to be applied as the logic variables remain unchanged under small measurement noise.

The MultiHyRL algorithm can be applied to any number of obstacles that satisfy the properties outlined in Section 3.1. However, Steps 1-4 are executed for the canonical obstacle setting as discussed in Section 3.2, namely, a single obstacle positioned at $p_{\text{ob}} = (-2, 0)$ and a positional set point $p^* = (0, 0)$. By means of a transformation, the results for the canonical obstacle setting are generalized to apply for any number of obstacles that satisfy the properties outlined in Section 3.1, without the need for fine-tuning the policies found in Step 4 or tweaking the partitions found in Step 3. In the following subsections, the MultiHyRL algorithm is applied to the canonical obstacle setting, and in Section 4.3 the transformation is discussed that generalizes the results to any obstacle setting that satisfies the properties outlined in Section 3.1.

4.2 Finding Critical Points

By leveraging a set of critical points, the state space can be partitioned into two sets for each obstacle. This division is achieved by ensuring that trajectories originating near these critical points evolve in divergent directions within each respective partition and do not leave the partition they start in. A set of critical points $\mathcal{M}^* \subset \mathcal{Z}$ exists for a closed-loop system $\dot{z} = f(z, \pi(z))$ when the following property holds:

($\star$) there exists $\delta > 0$ such that for each state $z \in \mathcal{M}^*$ there exist initial states $z_0, z_1 \in \{z\} + \delta\mathbb{B}$ such that solutions ϕ_0, ϕ_1 to $\dot{z} = f(z, \pi(z))$ starting from z_0, z_1, respectively, satisfy

$$\phi_0(t) \in \text{int } \mathcal{M}_0 \text{ for all } t \in \text{dom } \phi_0 \setminus \{0\} \text{ and } \phi_1(t) \in \text{int } \mathcal{M}_1 \text{ for all } t \in \text{dom } \phi_1 \setminus \{0\}, \quad (6)$$

where $\mathcal{M}_0$ and $\mathcal{M}_1$ are partitions of the environment $\mathcal{Z}$ with properties $\mathcal{M}_0 \cup \mathcal{M}_1 = \mathcal{Z}$ and $\mathcal{M}_0 \cap \mathcal{M}_1 = \mathcal{M}^*$.

In the context of obstacle avoidance (topological obstructions), each obstacle has an associated set of critical points. As discussed in Section 3.2, solutions evolve in divergent directions for a small change in the vehicle's state when facing an obstacle. For each obstacle, the state space can be partitioned into two parts: solutions that steer past the obstacle clockwise (e.g., $\mathcal{M}_0$) and solutions that steer past the obstacle counterclockwise (e.g., $\mathcal{M}_1$). Using ($\star$), we can state that for each obstacle $\ell \in \{1, 2, \ldots, N\}$, we can find a set of critical points $\mathcal{M}_\ell^*$ and partition the state space $\mathcal{Z}$ into $\mathcal{M}_{0,\ell}$ and $\mathcal{M}_{1,\ell}$ with properties $\mathcal{M}_{0,\ell} \cup \mathcal{M}_{1,\ell} = \mathcal{Z}$ and $\mathcal{M}_{0,\ell} \cap \mathcal{M}_{1,\ell} = \mathcal{M}^*$. An algorithm is designed that searches the state space for initial conditions for which ($\star$) holds and thereby finds the sets of critical points $\mathcal{M}_\ell^*$ for each obstacle $\ell \in \{1, 2, \ldots, N\}$. A detailed description of the algorithm is given in Appendix B.1. The algorithm is applied to the canonical setting described in Section 3.2 to find its set of critical points $\mathcal{M}^*$.

4.3 Partitioning the State Space

In this section, the state space for the canonical setting described in Section 3.2 is partitioned into two overlapping sets $\mathcal{M}_0^{\text{ext}}$ and $\mathcal{M}_1^{\text{ext}}$. This partition ensures the vehicle can only pass an obstacle on one side (clockwise or counterclockwise), thus removing the topological obstruction. To create these overlapping sets, a Support Vector Machine (SVM) model is trained to classify if a position p is in $\mathcal{M}_0^{\text{ext}} \setminus \mathcal{M}_1^{\text{ext}}$, $\mathcal{M}_1^{\text{ext}} \setminus \mathcal{M}_0^{\text{ext}}$, or in the overlap $\mathcal{M}_0^{\text{ext}} \cap \mathcal{M}_1^{\text{ext}}$. This classification identifies when a switch needs to occur in the hysteresis switching mechanism, as discussed in Step 5 in Section 4.1. This section discusses the creation of the data set for training the SVM model and the transformation

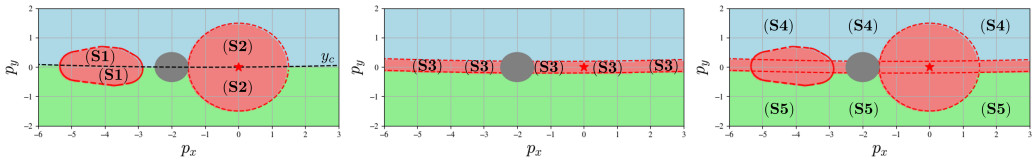


Figure 4: Overview of five regions used for the creation of the SVM data set. The red areas denote regions related to the conditions (**S1**), (**S2**), and (**S3**). The blue and green areas denote the regions (**S4**) and (**S5**), respectively. The black line denotes the center line y_c, the gray circle denotes the obstacle, and the red $\star$ denotes the set-point position p^*.

that generalizes the results for the canonical setting to apply for any number of obstacles that satisfy the properties outlined in Section 3.1.

To create a training data set for the SVM model, a rough estimate of the partitions $\mathcal{M}_0^{\text{ext}} \setminus \mathcal{M}_1^{\text{ext}}$, $\mathcal{M}_1^{\text{ext}} \setminus \mathcal{M}_0^{\text{ext}}$, and $\mathcal{M}_0^{\text{ext}} \cap \mathcal{M}_1^{\text{ext}}$ is made. The estimate of overlapping $\mathcal{M}_0^{\text{ext}} \cap \mathcal{M}_1^{\text{ext}}$ consists of three regions, namely, (**S1**), (**S2**), and (**S3**). The estimates of $\mathcal{M}_0^{\text{ext}} \setminus \mathcal{M}_1^{\text{ext}}$ and $\mathcal{M}_1^{\text{ext}} \setminus \mathcal{M}_0^{\text{ext}}$ consist of the regions (**S4**) and (**S5**), respectively. An overview of the regions is shown in Figure 4. Region (**S1**) collects positions that are inside the *smoothened set of critical points* $\mathcal{M}^\sigma$. The set of critical points $\mathcal{M}^*$ is smoothened by a factor of $\sigma \in \mathbb{R}_{>0}$ and defined as $\mathcal{M}^\sigma := \text{Conv}\left(\mathcal{M}^* + \sigma\mathbb{B}\right)$. Region (**S2**) collects positions that are in the vicinity of the set-point position, namely, $p \in p^* + c_5\mathbb{B}$, where $c_5 \in \mathbb{R}_{>0}$. Region (**S3**) collects positions that are in the vicinity of the *center line* $y_c : \mathcal{P}_x \to \mathcal{P}_y$ and are not inside the obstacle. A position p is not inside an obstacle if the distance to the obstacle is greater than zero, namely, $|p|_{\mathcal{C}} > 0$. The center line is obtained by fitting a second-order polynomial through the center p_c of $\mathcal{M}^\sigma$ defined as $p_c = (p_{x,c}, p_{y,c}) := \arg\min_{p_{x,c} \in \mathcal{P}_x} \max_{p_{y,c} \in \mathcal{M}^\sigma} |p_{x,c} - p_{y,c}|$, the center of the obstacle p_{ob}, and the set-point position p^*. A position $p = (p_x, p_y)$ is in the vicinity of the center line y_c if $y_c(p_x) - c_6 \leq p_y \leq y_c(p_x) + c_6$, where $c_6 < r_{\text{ob}}$. Region (**S4**) collects positions $p = (p_x, p_y)$ that are not in the regions (**S1-S3**) and are above the center line, that is, $p_y > y_c(p_x)$. Region (**S5**) collects positions $p = (p_x, p_y)$ that are not in the regions (**S1-S3**) and are below the center line, that is, $p_y < y_c(p_x)$. A training dataset is generated by labeling a grid of positions according to the regions (**S1-S5**). An SVM model is trained using this dataset with a Gaussian radial basis function to ensure that the boundaries between the sets $\mathcal{M}_0^{\text{ext}} \setminus \mathcal{M}_1^{\text{ext}}$, $\mathcal{M}_1^{\text{ext}} \setminus \mathcal{M}_0^{\text{ext}}$, and $\mathcal{M}_0^{\text{ext}} \cap \mathcal{M}_1^{\text{ext}}$ are smooth. The sets $\mathcal{M}_0^{\text{ext}}$ and $\mathcal{M}_1^{\text{ext}}$ are then obtained by inferencing the trained SVM model. Namely, a position is in $\mathcal{M}_i^{\text{ext}}$ if the position is in $\mathcal{M}_i^{\text{ext}} \setminus \mathcal{M}_h^{\text{ext}}$ or $\mathcal{M}_0^{\text{ext}} \cap \mathcal{M}_1^{\text{ext}}$ for $i, h \in \{0, 1\}$ and $i \neq h$.

To generalize the SVM model to find the sets $\mathcal{M}_0^{\text{ext}}$ and $\mathcal{M}_1^{\text{ext}}$ for any obstacle settings, a transformation is applied to the inputs of the SVM model. The details on the transformation are discussed in Appendix B.2. Additionally, if more than one obstacle is present in the environment, the sets $\mathcal{M}_0^{\text{ext}}$ and $\mathcal{M}_1^{\text{ext}}$ are partitioned with a *relaxed* Voronoi partition into N Voronoi cells, one for each obstacle. The relaxed partition allows for overlap between the cells to prevent *Zeno behavior* (infinite jumps in finite time) for the resulting hybrid system.[7] With this relaxed Voronoi partition, the pair of sets $\mathcal{M}_{0,\ell}^{\text{ext}}$ and $\mathcal{M}_{1,\ell}^{\text{ext}}$ are obtained for each obstacle $\ell \in \{1, 2, \ldots, N\}$. Figure 5 shows an example of the sets $\mathcal{M}_{0,\ell}^{\text{ext}}$ and $\mathcal{M}_{1,\ell}^{\text{ext}}$ for a four obstacle setting $N = 4$.

4.4 Training two new control policies

After finding the extended overlapping sets $\mathcal{M}_0^{\text{ext}}$ and $\mathcal{M}_1^{\text{ext}}$ for the canonical obstacle-setting described in Section 3.2, two new policies are trained for the same obstacle setting. A policy, denoted π_0, is trained on $\mathcal{M}_0^{\text{ext}}$ and another policy, denoted π_1, is trained on $\mathcal{M}_1^{\text{ext}}$.[8] To train these policies, a slight modification is made to the reward function. Specifically, for each $i \in \{0, 1\}$, the

[7] See Sanfelice (2021) for more information on Zeno behavior. The details and additional examples on the relaxed Voronoi partition are given in Appendix B.3.

[8] The policy parameters from Step 1 in Section 4.1 can initialize policies π_0 and π_1 to speed up training.

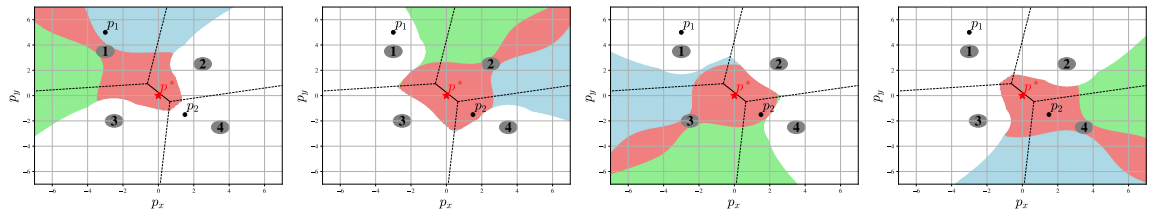


Figure 5: Visualization of the state space partition for four obstacles. For each obstacle $\ell \in \{1, 2, 3, 4\}$, the blue & red and the green & red areas denote the sets $\mathcal{M}^{\text{ext}}_{0,\ell}$ and $\mathcal{M}^{\text{ext}}_{1,\ell}$, respectively. The red area denotes the intersection $\mathcal{M}^{\text{ext}}_{0,\ell} \cap \mathcal{M}^{\text{ext}}_{1,\ell}$. The black lines denote the Voronoi vertices, the gray circles denote the obstacles, and the red circle denotes the set-point position p^*. The points p_1 and p_2 are referenced in Section 4.5 to derive the hybrid control algorithm.

set $\mathcal{Z} \setminus \mathcal{M}^{\text{ext}}_i$ is added to the collision set $\mathcal{C}$. This penalizes the agent for leaving its respective extended overlapping set, motivating it to stay inside. For the single obstacle-setting described in Section 3.2, the policy π_0 steers the vehicle clockwise past the obstacle for each $z \in \mathcal{M}^{\text{ext}}_0$, and the policy π_1 steers the vehicle counterclockwise past the obstacle for each $z \in \mathcal{M}^{\text{ext}}_1$. As discussed in Section 3.2, the policies π_0 and π_1 generalize across various obstacle settings, as long as the obstacles adhere to the properties outlined in Section 3.1.

4.5 Hybrid Control Algorithm for Supervising Policies

After obtaining the sets $\mathcal{M}^{\text{ext}}_{0,\ell}$ and $\mathcal{M}^{\text{ext}}_{1,\ell}$ for each obstacle $\ell \in \{1, 2, \ldots, N\}$ and the control policies π_0 and π_1, a hybrid control algorithm is designed to supervise the policies. To do so, two logic variables are introduced: $q \in \{0, 1\}$ indicating the active policy and $\lambda \in \{1, 2, \ldots, N\}$ indicating the focused obstacle to be avoided. To construct the hybrid closed-loop system $\mathcal{H} = (C, F, D, G)$, we define its state as the collection of the vehicle's state z, and the newly introduced logic variables q and λ, namely, $x := (z, q, \lambda) \in \mathcal{Z} \times \{0, 1\} \times \{1, 2, \ldots, N\}$. Next, we define the *flow map* F that describes the continuous evolution of the state z as

$$\begin{bmatrix} \dot{z} \\ \dot{q} \\ \dot{\lambda} \end{bmatrix} = F(x) := \begin{bmatrix} f(z, \pi_{\mathcal{H}}(z, q)) \\ 0 \\ 0 \end{bmatrix} \quad x \in C, \tag{7}$$

where $\pi_{\mathcal{H}}$ is the hybrid control policy that allows us to switch between π_0 and π_1 depending on the value of the logic variable q. The hybrid control policy is given by

$$\pi_{\mathcal{H}}(z, q) := \begin{cases} \pi_0(z) & \text{if} \quad q = 0 \\ \pi_1(z) & \text{if} \quad q = 1 \end{cases}. \tag{8}$$

In (7), it can be seen that the vehicle's state flows according to its dynamics (4) under the hybrid control policy (8), the logic variables q and λ do not change during flow, and that the system flows whenever the state is inside the *flow set* C, which is defined below. As the logic variables do not change during flow, the hybrid policy remains equal to π_0 or π_1 during flow, depending on the value of q. To define the flow set, consider the four obstacle setting and the sets $\mathcal{M}^{\text{ext}}_{0,\ell}$ and $\mathcal{M}^{\text{ext}}_{1,\ell}$ for each obstacle $\ell \in \{1, 2, 3, 4\}$ shown in Figure 5. Suppose the vehicle's position is $p_1 = (p_x, p_y) = (-3, 5)$, that is, the vehicle's state z is in the overlapping extended set $\mathcal{M}^{\text{ext}}_{0,1}$; see Figure 5. From this vehicle position, the vehicle has to pass obstacle 1 clockwise. So for the system to flow, the focused obstacle has to be obstacle 1, that is, $\lambda = 1$, and the policy that steers the vehicle clockwise has to be active, namely, policy π_0 and thus $q = 0$. Moreover, for x to flow, the logic variables q and λ must correspond to the indexes of the overlapping extended set $\mathcal{M}^{\text{ext}}_{q,\lambda}$ the vehicle's state is in. Therefore, the flow set is given by

$$C := \bigcup_{q \in \{0,1\},\, \lambda \in \{1,2,\ldots,N\}} \left(\mathcal{M}^{\text{ext}}_{q,\lambda} \times \{q\} \times \{\lambda\} \right). \tag{9}$$

Next, suppose again that the vehicle's position is p_1, that is, $z \in \mathcal{M}_{0,1}^{\text{ext}}$, the logic variable $\lambda = 1$, and the logic variable $q = 1$. This time, the state x is not in the flow set (9) as $q \neq 0$. Therefore, the value of the logic variable q has to *jump* from $q = 1$ to $q = 0$ such that the state x after the jump is in the flow set (9). To capture this jump for the logic value q, we define the *hysteresis* jump set as

$$D_{h,q,\lambda}^{\text{hyst}} := \overline{\left(\mathcal{M}_{h,\lambda}^{\text{ext}} \setminus \mathcal{M}_{q,\lambda}^{\text{ext}}\right)} \times \{q\} \times \{\lambda\}, \tag{10}$$

for $h \in \{0, 1\}$ where $h \neq q$ and h is the value to reset q to. In (10), it can be seen that this jump set captures the condition when the focused obstacle λ corresponds to the overlapping set $\mathcal{M}_{q,\lambda}^{\text{ext}}$, but the value of q needs to be switched. The second type of jump can occur when the obstacle that needs to be focused does not correspond to the value of the logic variable λ. For example, suppose again that the vehicle's position is p_1, that is, $z \in \mathcal{M}_{0,1}^{\text{ext}}$, the logic variable $q = 0$, and the logic variable $\lambda = 3$. From this state, the hybrid system cannot flow as the state is not in the flow set (9). Hence, we want to reset the value of λ from $\lambda = 3$ to $\lambda = 1$. Additionally, the new value for the logic variable λ can be arbitrarily chosen as the sets $\mathcal{M}_{0,\ell}^{\text{ext}}$ and $\mathcal{M}_{1,\ell}^{\text{ext}}$ for $\ell \in \{1, 2, \ldots, N\}$ overlap by their construction. For example, suppose that the vehicle's position is $p_2 = (p_x, p_y) = (1.5, -1.5)$, that is, $z \in \bigcap_{q\in\{0,1\},\, \lambda\in\{2,3,4\}} \mathcal{M}_{q,\lambda}^{\text{ext}}$, the logic variable $q = 0$, and the logic variable $\lambda = 1$. The logic variable λ needs to be reset as the state is not in the flow set, but the logic variable can be updated to 2, 3, or 4. When multiple options are available, the new value of λ is picked uniformly at random from the available options, that is, $\lambda^+ \in \{2, 3, 4\}$. An instantiation of λ^+ would be 3. The *jump map* G that captures these situations is given by

$$\begin{bmatrix} z^+ \\ q^+ \\ \lambda^+ \end{bmatrix} \in G(x) := \begin{bmatrix} z \\ \begin{cases} q & \text{if} \quad x \in \mathcal{M}_{q,\lambda'(x)}^{\text{ext}} \times \{q\} \times \{\lambda'(x)\} \\ \left\{h \in \{0,1\} \setminus \{q\} : x \in D_{h,q,\lambda'(x)}^{\text{hyst}}\right\} & \text{else} \end{cases} \\ \lambda'(x) \end{bmatrix} \quad x \in D, \tag{11}$$

where

$$\lambda'(x) = \begin{cases} \Gamma(x) & \text{if} \quad x \in \overline{D \setminus D_\lambda^{\text{fcs}}} \\ \lambda & \text{else} \end{cases}, \quad D_l^{\text{fcs}} := \left(\mathcal{M}_{0,l}^{\text{ext}} \cup \mathcal{M}_{1,l}^{\text{ext}}\right) \times \{0, 1\} \times \{l\}, \tag{12}$$

where D_l^{fcs} is the set of points that correspond to the *focused* obstacle $l \in \{1, 2, \ldots, N\}$ and Γ is the focused obstacle jump map for switching the logic parameter λ (in)deterministically and is given by

$$\Gamma(x) = \left\{\Lambda \subset \{1, 2, \ldots, N\} : \left(\bigcap_{l\in\Lambda} D_l^{\text{fcs}}\right) \setminus \left(\bigcup_{i\notin\Lambda,\, i\in\{1,2,\ldots,N\}} D_i^{\text{fcs}}\right)\right\}. \tag{13}$$

Lastly, the *jump set* D of for the hybrid system is given by

$$D := \bigcup_{q\in\{0,1\},\, \lambda\in\{1,2,\ldots,N\}} \left(D_{q,\lambda}^{\text{hyst}} \cup D_\lambda^{\text{fcs}}\right) = \overline{(\mathcal{Z} \times \{q\} \times \{\lambda\}) \setminus C}. \tag{14}$$

It is important to note that the sets $\mathcal{M}_{0,\lambda}^{\text{ext}}$ and $\mathcal{M}_{1,\lambda}^{\text{ext}}$ overlap each other in front of each obstacle λ with respect to the positional set point. Furthermore, the sets $\mathcal{M}_{0,\lambda}^{\text{ext}} \cup \mathcal{M}_{1,\lambda}^{\text{ext}}$ and $\mathcal{M}_{0,h}^{\text{ext}} \cup \mathcal{M}_{1,h}^{\text{ext}}$ also overlap each other for each obstacle λ and h where $\lambda \neq h \in \{1, 2, \ldots, N\}$. Therefore, the hybrid system cannot continuously switch between its values of q and λ in the presence of measurement noise. Hence, the hybrid system cannot switch continuously between its policies π_0 and π_1 to pass an obstacle in the presence of small measurement noise. The vehicle will pass the obstacle clockwise if $q = 0$ and counterclockwise if $q = 1$, thereby implementing a *hysteresis* switching effect between the two policies and granting robustness against the small measurement noise. The level of robustness is related to the amount of overlap the sets $\mathcal{M}_{0,\lambda}^{\text{ext}}$ and $\mathcal{M}_{1,\lambda}^{\text{ext}}$ and the sets $\mathcal{M}_{0,\lambda}^{\text{ext}} \cup \mathcal{M}_{1,\lambda}^{\text{ext}}$ and $\mathcal{M}_{0,h}^{\text{ext}} \cup \mathcal{M}_{1,h}^{\text{ext}}$ for $\lambda \neq h \in \{1, 2, \ldots, N\}$ have. Specifically, if the overlap width is greater than $\varepsilon \in \mathbb{R}_{>0}$, the hybrid system is robust against chattering behavior caused by measurement noise of ε on the vehicle's perceived position.

5 Numerical Validation

In this Section, we apply the MultiHyRL algorithm as described in Section 4 to overcome the lack of robustness of the normal agent. To demonstrate the robustness of the MultiHyRL agent, both the normal agent and the MultiHyRL agent are simulated in various settings for many initial conditions in the presence of the *exact same* measurement noise signal. The controllers use a sample-and-hold approach for both the normal and MultiHyRL agents, modeled as a hybrid system; details are given in Appendix B.4.

The normal and MultiHyRL agents are deployed in two environments with eight and ten static obstacles that satisfy the properties outlined in Section 3.1. Four measurement noise settings are considered for both agents: (a) no measurement noise, $\varepsilon_p = \varepsilon_\xi = 0$, (b) only positional noise, $\varepsilon_p = 0.2$, $\varepsilon_\xi = 0$, (c) only orientational noise, $\varepsilon_p = 0$, $\varepsilon_\xi = 0.3$, and (d) both positional and orientational noise, $\varepsilon_p = 0.05$, $\varepsilon_\xi = 0.1$. The applied measurement noise signal $m : \mathcal{S}^1 \times \mathbb{R}_{\geq 0} \to \mathbb{R}^4$ is given by

$$m(\xi, t) = \begin{bmatrix} m_{\text{sgn}}(t)\varepsilon_p \\ -m_{\text{sgn}}(t)\varepsilon_p \\ (\text{Rot}(m_{\text{sgn}}(t)\varepsilon_\xi) - I)\xi \end{bmatrix}, \quad m_{\text{sgn}}(t) = \text{sgn}\left(\cos\left(\frac{\pi t}{\Delta t}\right)\right), \tag{15}$$

where the function m_{sgn} changes its sign at every sampling time interval Δt. For the MultiHyRL agent, the logic variables q and λ are randomly initialized. Figure 6 shows the simulation results for each setting. In the first and third rows of column (a) in Figure 6, it can be seen that the normal agent gets stuck in front of an obstacle for a select few initial conditions. This can be attributed to the continuous policy that is used for the normal agent. Specifically, by the property ($\star$) in Section 4.2, solutions get stuck at critical points located in front of the obstacles. Furthermore, the first and third rows of columns (b), (c), and (d) show that the normal agent's ability to reach the positional set point worsens significantly due to measurement noise, causing chattering behavior and resulting in many initial conditions getting stuck in front of obstacles.

Contrary to the normal agent, the MultiHyRL agent reaches the positional set point for all noise settings and initial conditions, as shown in the second and fourth rows of Figure 6.[9] This is due to the hysteresis and obstacle focus mechanisms that prevent chattering in front of the obstacles. The MultiHyRL agent's solutions sometimes cross each other for similar initial positions due to the random initialization of logic parameters q and λ. For example, if the initial position is in the overlapping region between $\mathcal{M}^{\text{ext}}_{0,\lambda}$ and $\mathcal{M}^{\text{ext}}_{1,\lambda}$ for obstacle λ, the initial value of q can be 0 or 1, steering the vehicle clockwise or counterclockwise, respectively. In column (a), the MultiHyRL agent typically takes a slightly longer path to the positional set point than the normal agent due to these mechanisms. However, the robustness benefits of the MultiHyRL agent outweigh this slight performance loss in the noiseless setting.

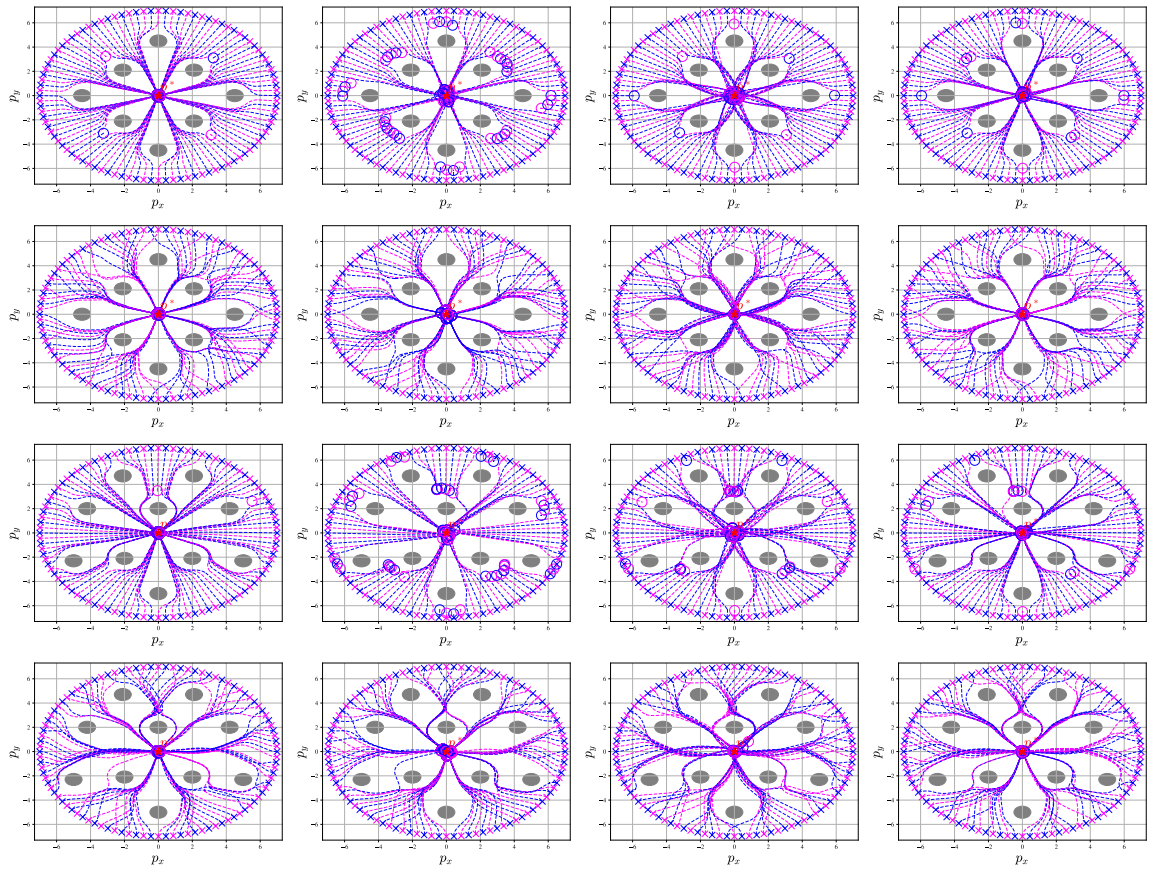
Without any modifications, the MultiHyRL algorithm can be applied to obstacles of different sizes and even dynamic obstacles.[10] To further validate the MultiHyRL agent's performance over the normal agent, they can be deployed in a simplified game of Capture the Flag against each other. Additional simulations are provided in Appendix C.

6 Conclusion

This paper presents a new hybrid RL algorithm, referred to as MultiHyRL, for vehicles operating in environments with an arbitrary number of randomly located obstacles to overcome the challenges posed by topological obstructions. MultiHyRL provides a hybrid control policy with hysteresis switching that is robust against adversarial attacks on the observation space near critical areas for such environments. MultiHyRL addresses the 2D bird's-eye view obstacle avoidance problem featuring a complex observation space that combines local (images) and global (vectors) observations.

[9] There exists another topological obstruction on the vehicle's orientation ξ, which can be dealt with following the procedure done on the unit circle problem in de Priester et al. (2022).

[10] For dynamic obstacles, we assume they do not move faster than the vehicle and satisfy the properties in Section 3.1.



(a) $\varepsilon_p = \varepsilon_\xi = 0$. (b) $\varepsilon_p = 0.2$ and $\varepsilon_\xi = 0$. (c) $\varepsilon_p = 0$ and $\varepsilon_\xi = 0.3$. (d) $\varepsilon_p = 0.05$ and $\varepsilon_\xi = 0.1$.

Figure 6: Visualization of the state trajectories under different measurement noise settings for the normal agent (rows 1 and 3) versus the MultiHyRL agent (rows 2 and 4) in two static obstacle environments. The logic variables q and λ are randomly initialized for the MultiHyRL agent. The trajectories alternate colors for readability. The gray circles denote obstacles, the $\times$'s denote the initial conditions, the $\circ$'s denote the terminal positions, and the red $\star$ denotes the set-point position p^*.

Numerical results highlight its robustness to adversarial attacks in various challenging obstacle avoidance settings where PPO, a traditional RL method, fails. With some modifications, the MultiHyRL algorithm can be extended to 3D tasks, as future work will demonstrate. While our primary focus is on obstacle avoidance, the concept of critical points in the observation space applies to other problems where chattering between policies causes undesired behavior. This relevance extends to various RL domains, such as hierarchical RL and options-based methods. The MultiHyRL algorithm can be adapted to enhance the robustness of these methods as well.

Acknowledgments

Research by Jan de Priester and R. G. Sanfelice partially supported by NSF Grants no. CNS-2039054 and CNS-2111688, by AFOSR Grants nos. FA9550-19-1-0169, FA9550-20-1-0238, FA9550-23-1-0145, and FA9550-23-1-0313, by AFRL Grant nos. FA8651-22-1-0017 and FA8651-23-1-0004, by ARO Grant no. W911NF-20-1-0253, and by DoD Grant no. W911NF-23-1-0158.

References

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. *CoRR*, abs/1609.05140, 2016. URL http://arxiv.org/abs/1609.05140.

André Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Jonathan J. Hunt, Shibl Mourad, David Silver, and Doina Precup. The option keyboard: Combining skills in reinforcement learning. *CoRR*, abs/2106.13105, 2021. URL https://arxiv.org/abs/2106.13105.

Yikun Cheng, Pan Zhao, and Naira Hovakimyan. Safe and efficient reinforcement learning using disturbance-observer-based control barrier functions. In Nikolai Matni, Manfred Morari, and George J. Pappas (eds.), *Proceedings of The 5th Annual Learning for Dynamics and Control Conference*, volume 211 of *Proceedings of Machine Learning Research*, pp. 104–115. PMLR, 15–16 Jun 2023. URL https://proceedings.mlr.press/v211/cheng23a.html.

Jaewan Choi, Geonhee Lee, and Chibum Lee. Reinforcement learning-based dynamic obstacle avoidance and integration of path planning. *Intelligent Service Robotics*, 14:663–677, 2021.

Reinis Cimurs, Jin Han Lee, and Il Hong Suh. Goal-oriented obstacle avoidance with deep reinforcement learning in continuous action space. *Electronics*, 9(3), 2020. ISSN 2079-9292. doi: 10.3390/electronics9030411. URL https://www.mdpi.com/2079-9292/9/3/411.

Jan de Priester, Ricardo G. Sanfelice, and Nathan van de Wouw. Hysteresis-based rl: Robustifying reinforcement learning-based control policies via hybrid control. In *2022 American Control Conference (ACC)*, pp. 2663–2668, 2022. doi: 10.23919/ACC53348.2022.9867627.

Yousef Emam, Paul Glotfelter, Zsolt Kira, and Magnus Egerstedt. Safe model-based reinforcement learning using robust control barrier functions. *CoRR*, abs/2110.05415, 2021. URL https://arxiv.org/abs/2110.05415.

Michael Everett, Yu Fan Chen, and Jonathan P. How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. *CoRR*, abs/1805.01956, 2018. URL http://arxiv.org/abs/1805.01956.

Michael Everett, Björn Lütjens, and Jonathan P. How. Certifiable robustness to adversarial state uncertainty in deep reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 33(9):4184–4198, 2022. doi: 10.1109/TNNLS.2021.3056046.

Shumin Feng, Bijo Sebastian, and Pinhas Ben-Tzvi. A collision avoidance method based on deep reinforcement learning. *Robotics*, 10(2):73, May 2021. ISSN 2218-6581. doi: 10.3390/robotics10020073. URL http://dx.doi.org/10.3390/robotics10020073.

Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. *CoRR*, abs/1710.09767, 2017. URL http://arxiv.org/abs/1710.09767.

Rafal Goebel, Ricardo G. Sanfelice, and Andrew R. Teel. *Hybrid Dynamical Systems: Modeling, Stability, and Robustness*. Princeton University Press, 2012. ISBN 9780691153896. URL http://www.jstor.org/stable/j.ctt7s02z.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL https://api.semanticscholar.org/CorpusID:6628106.

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 *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1213–1220, 2021. doi: 10.1109/IROS51168.2021.9636039.

Yongyuan Liang, Yanchao Sun, Ruijie Zheng, and Furong Huang. Efficient adversarial training without attacking: Worst-case-aware robust reinforcement learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 22547–22561. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/8d6b1d775014eff18256abeb207202ad-Paper-Conference.pdf.

Björn Lütjens, Michael Everett, and Jonathan P. How. Certified adversarial robustness for deep reinforcement learning. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura (eds.), *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 1328–1337. PMLR, 30 Oct–01 Nov 2020. URL https://proceedings.mlr.press/v100/lutjens20a.html.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

Ajay Mandlekar, Yuke Zhu, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Adversarially robust policy learning: Active construction of physically-plausible perturbations. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3932–3939. IEEE, 2017.

C. G. Mayhew, R. G. Sanfelice, and A. R. Teel. Quaternion-based hybrid controller for robust global attitude tracking. *IEEE Transactions on Automatic Control*, 56(11):2555–2566, November 2011. doi: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5701762. URL https://hybrid.soe.ucsc.edu/files/preprints/50.pdf.

Ofir Nachum, Shixiang (Shane) Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/e6384711491713d29bc63fc5eeb5ba4f-Paper.pdf.

Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *CoRR*, abs/1511.04508, 2015. URL http://arxiv.org/abs/1511.04508.

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2817–2826. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/pinto17a.html.

Christophe Prieur, Rafal Goebel, and Andrew R. Teel. Hybrid feedback control and robust stabilization of nonlinear systems. *IEEE Transactions on Automatic Control*, 52(11):2103–2117, 2007. doi: 10.1109/TAC.2007.908320.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., USA, 1st edition, 1994. ISBN 0471619779.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

R. G. Sanfelice. *Hybrid Feedback Control*. Princeton University Press, New Jersey, 2021.

R. G. Sanfelice, M. J. Messina, S. E. Tuna, and A. R. Teel. Robust hybrid controllers for continuous-time systems with applications to obstacle avoidance and regulation to disconnected set of points. In *Proc. 25th American Control Conference*, pp. 3352–3357, 2006. doi: http://ieeexplore.

ieee.org/iel5/11005/34689/01657236.pdf?tp=\&isnumber=\&arnumber=1657236. URL https://hybrid.soe.ucsc.edu/files/preprints/7.pdf.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rkZvSe-RZ.

Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Mingyan Liu, Duane Boning, and Cho-Jui Hsieh. Robust deep reinforcement learning against adversarial perturbations on state observations. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 21024–21037. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/f0eb6568ea114ba6e293f903c34d7488-Paper.pdf.

A Reinforcement Learning Implementation Details

A.1 Proximal Policy Optimization

This paper uses OpenAI's Stable Baselines 3 implementation of the PPO algorithm (Raffin et al., 2021) to learn an approximate value function $\hat{v}$, parameterized by ϕ, and stochastic policy π^s, parameterized by θ. For a training episode, the simulation provides information at time points $k \in \{0, 1, \ldots, K\}$ where $K \in \mathbb{N}$ is the time point corresponding to the maximum simulation horizon time. The approximated value function $\hat{v}$ is trained by minimizing the mean squared error between the approximated value function $\hat{v}$ and the returns G_k observed during training episodes, given by

$$G_k(\zeta) = \sum_{\ell=k}^{K-1} \gamma^{\ell-k} R(z_{\ell+1}, u_\ell), \tag{16}$$

where $\zeta = \{(u_0, z_1), (u_1, z_2), \ldots, (u_{K-2}, z_{K-1}), (u_{K-1}, z_K)\}$ is the discrete-time solution pair to (1). The update of the approximate value function's parameters ϕ is given by

$$\phi_{\psi+1} = \min_{\phi} \frac{1}{|\mathcal{D}_\psi| K} \sum_{\zeta \in \mathcal{D}_\psi} \sum_{k=0}^{K-1} \left(\hat{v}(z_k; \phi) - G_k\right)^2, \tag{17}$$

where $\psi \in \mathbb{N}$ is the update step and $\mathcal{D}_\psi$ is a set containing discrete-time solution pairs ζ obtained during training episodes under the stochastic policy π^s_ψ.[11] The value function update in (17) is done via gradient ascent with Adam (Kingma & Ba, 2014).

During training, a stochastic policy is used such that the agent can explore the environment, that is, to observe the rewards obtained for a large variety of state-action pairs. A multivariate Gaussian distribution is used to parameterize a stochastic policy and is given by

$$\pi^s(u|z) = \mathcal{N}\left(\mu(z), \Sigma^2(z)\right), \tag{18}$$

[11] As the value function estimates the return under a specific policy π, the trajectories used for updates on the approximate value function must be under the same policy. Therefore, the trajectories ζ obtained under different policies $\pi^s \neq \pi^s_\psi$ are not used to update ϕ_ψ.

where $\mu : \mathcal{Z} \to \mathcal{U}$ denotes the mean vector of the multivariate Gaussian distribution $\mathcal{N}$ and $\Sigma^2 : \mathcal{Z} \to \mathcal{U} \times \mathcal{U}$ the diagonal covariance matrix of the multivariate Gaussian distribution $\mathcal{N}$. A neural network parameterizes this distribution; we use a neural network to map the states z to a mean vector μ and a state covariance matrix Σ^2. A deterministic policy $\pi : \mathcal{Z} \to \mathcal{U}$ is then obtained by evaluating the mean vector μ of the stochastic policy π^s, that is, $\pi = \mu$. The policy is parameterized by a neural network with parameters θ. The update of the policy's parameters θ is given by

$$\theta_{\psi+1} = \max_{\theta} \frac{1}{|\mathcal{D}_\psi| K} \sum_{\zeta \in \mathcal{D}_\psi} \sum_{k=0}^{K-1} L(\zeta, \theta_\psi, \theta), \tag{19}$$

where L is the objective function given by

$$L(\zeta, \theta_\psi, \theta) = \min \left(\frac{\pi^s(u_k|z_k;\theta)}{\pi^s_\psi(u_k|z_k;\theta_\psi)} \hat{A}_{\psi,k}(\zeta), g(\epsilon, \hat{A}_{\psi,k}(\zeta)) \right), \tag{20}$$

where the estimated advantage $\hat{A}_{\psi,k}$ at update step ψ and time step k is obtained via the Generalized Advantage Estimator (GAE) (Schulman et al., 2016) method and is given by

$$\hat{A}_{\psi,k}(\zeta) = \sum_{l=k}^{K-1} (\gamma\lambda)^{l-k} \delta_\psi(z_l, u_l, z_{l+1}), \tag{21}$$

where $\delta_{\psi,k}$ is the TD error at update step ψ and time step k given by

$$\delta_\psi(z_l, u_l, z_{l+1}) = R(z_{l+1}, u_l) + \hat{v}_\psi(z_{l+1}) - \hat{v}_\psi(z_l), \tag{22}$$

and $\lambda \in (0, 1]$ is a factor for the trade-off between bias and variance for the GAE. If $\lambda \to 0$, the bias for the GAE is minimal, but the variance is high for the GAE. If $\lambda = 1$, the variance is minimal for the GAE, but the GAE is most biased. Furthermore,

$$g(\epsilon, \hat{A}_{\psi,k}) = \begin{cases} (1+\epsilon)\hat{A}_{\psi,k} & \text{if } \hat{A}_{\psi,k} \geq 0 \\ (1-\epsilon)\hat{A}_{\psi,k} & \text{if } \hat{A}_{\psi,k} < 0 \end{cases}, \tag{23}$$

where $\epsilon \in \mathbb{R}_{>0}$ is a hyperparameter that controls how far a new policy may deviate from the old policy. The policy update in (19) is done via gradient ascent with Adam (Kingma & Ba, 2014).

A.2 Network Architectures and Hyperparameters

This section describes the network architectures used and the hyperparameters used for the PPO algorithm. The architecture of the CNN is given by

$$\text{CNN}(\mathcal{I}) = \text{Flatten}\Big(\text{MaxPool2D}\big[\text{ReLU}(\text{Conv2D}(\mathcal{I}, \Psi_{\text{Conv2D}})), \Psi_{\text{MaxPool2D}}\big]\Big), \tag{24}$$

where $\mathcal{I}$ is the input image with equal width and length of $w_{\text{bev}} \times w_{\text{bev}} = 2 \times 2$ and a resolution of $n_{\text{res}} \times n_{\text{res}} = 64 \times 64$ array elements, Flatten is a function that converts the 2D input into a 1D vector, MaxPool2D is a 2D max pooling layer to reduce the spatial dimensions of the input feature map with hyperparameters $\Psi_{\text{MaxPool2D}} = \{\text{kernel size} = 2\}$, ReLU is the rectified linear unit activation function given by $f(\chi) = \max(\chi, 0)$ for an input χ, and Conv2D is the 2D convolution operation for with hyperparameters $\Psi_{\text{Conv2D}} = \{\text{out channels} = 2, \text{kernel size} = 6, \text{stride} = 2, \text{padding} = 0\}$. The architecture of the feature extractor's MLP is given by

$$\text{MLP}_{\text{feature}}(e) = \text{ReLU}(w_{\text{feature}} e) + b_{\text{feature}}, \tag{25}$$

where $e = (e_p, e_\theta)$ and w_{feature} and b_{feature} are the weights and biases of the feature extractor's MLP with 32 neurons. The architecture of the network's MLP, that is, the actor and critic networks, is given by

$$\text{MLP}_{\text{network}}(\text{features}) = w_{n,2}\text{ReLU}(w_{n,1}\text{ReLU}(w_{n,0}\text{features}) + b_{n,0}) + b_{n,1}) + b_{n,2}, \tag{26}$$

Parameter	Value	Parameter	Value
Time steps	200	Sampling time in seconds	0.05
Total training steps	5000000	Entropy coefficient	0.001
Number of parallel environments	8	Learning rate	0.0001
Discount factor γ	0.99	GAE factor λ	0.95
Batch size	64	Clip range	0.2
Value function coefficient	0.5	Maximum value for gradient clipping	0.5

Table 1: Hyperparameters used for the PPO algorithm

where features= $(\text{CNN}(\mathcal{I}), \text{MLP}_{\text{feature}}(e))$, and $w_{n,i}$ and $b_{n,i}$ are the weights and biases of the network's MLP with 32 neurons for each layer $i \in \{0, 1, 2\}$. The actor network (26) outputs two variables, namely, the mean μ and the diagonal covariance matrix Σ that defines the stochastic policy (18) for the input features. The critic network outputs one value for the input features: the approximated value function.

The hyperparameters used for the PPO algorithm are shown in Table 1.

B MultiHyRL Implementation Details

B.1 Algorithms

This Section provides the pseudocode for the algorithm used in Section 4.2 to find the critical points of a control policy π for the MultiHyRL algorithm. The goal of Algorithm 1 is to search the state space for initial conditions for which $(\star)$ holds under a control policy π. For the first iteration of the algorithm, the set of initial conditions (points) $\mathcal{Z}_0$ is a grid with a spacing between points of $\delta \in \mathbb{R}_{>0}$. Next, for each point $z_c \in \mathcal{Z}_0$, we test $n_{\text{rp}} \in \mathbb{N}_{>0}$ times if two points near z_c evolve in divergent directions. Specifically, the two nearby points z_1 and z_2 are computed using Algorithm 2. Algorithm 2 computes a *symmetric point pair* $(z_1, z_2) \in z_c + \delta\mathbb{B}$ by sampling a random vector of the same dimensionality as z_c from a uniform distribution, that is, $\nu \sim \mathcal{U}(-\delta, \delta)$, and the points are defined as $z_1 = z_c + \nu$ and $z_2 = z_c - \nu$.[12] Next, the system is simulated under the control policy π for a horizon time of $T \in \mathbb{R}_{>0}$ to obtain the solutions $t \mapsto \phi_i(t)$ for $i \in \{1, 2\}$. To determine if the solutions ϕ_1 and ϕ_2 evolve in divergent directions, the sum of the absolute differences between the two trajectories d is computed at times t when the simulation provides information. Specifically, at each sampling time interval t, we evaluate ϕ_1 and ϕ_2. Moreover, the difference at each time point t is adjusted by subtracting the initial differences $\phi_1(0)$ and $\phi_2(0)$ to focus on the divergence from the initial conditions and is normalized by the sampling time Δt. Next, the value of d is compared to the distance threshold value $\eta \in \mathbb{R}_{>0}$, which acts as a benchmark to identify significant divergences between trajectories. If $d > \eta$, the point z_c is *potentially* a critical point and is stored in a set $\mathcal{Z}_0^c$ that is reset for each iteration of the main loop of Algorithm 1.[13] After evaluating every point in $\mathcal{Z}_0$ and populating the set $\mathcal{Z}_0^c$, k-means clustering is applied on $\mathcal{Z}_0^c$ to "summarize" the points by $n_{\text{cp}} \in \mathbb{N}_{>0}$ clusters in $\mathcal{Z}_0^c$. This ensures that the total number of points evaluated in each iteration of the main loop remains computationally manageable. The set $\mathcal{Z}_0$ is reset to an empty set and repopulated with the newly found n_{cp} clusters.[14] Lastly, the value of δ is reduced by a factor $\beta \in (0, 1)$ to facilitate convergence to a set of critical points. If δ becomes smaller than the sampling time Δt, the sampling time is also scaled by β to ensure the system samples fast enough to dodge obstacles if it starts close to them.[15] After reducing the value of δ and repopulating the set $\mathcal{Z}_0$, the main loop of the algorithm is run again until $n_{\text{it}} \in \mathbb{N}_{>0}$ of the main loop have passed. By iterative refining δ, the

[12] The *symmetry* of the point pair (z_1, z_2) refers to the addition/subtraction of the random vector ν.

[13] Potentially here refers to the fact solutions can satisfy $d > \eta$ for large values of δ while the point z_c is not actually a critical point. Therefore, the point z_c is considered *potentially* a critical point. To verify whether the point z_c is indeed a critical point, the value of δ is decreased every iteration of the main loop of Algorithm 1.

[14] If the number of points in $\mathcal{Z}_0$ is less than n_{cp}, the clustering step is skipped, and $\mathcal{Z}_0$ is not reset.

[15] If the sampling cannot be adjusted in this manner, the algorithm stops when $\delta < \Delta t$.

Algorithm 1: Finding critical points

Data: Policy $\pi \in \Pi$ to be evaluated, number of iterations $n_{\text{it}} \in \mathbb{N}_{>0}$, number of random symmetric point pairs per center point $n_{\text{rp}} \in \mathbb{N}_{>0}$, initial set of points $\mathcal{Z}_0 \subset \mathcal{Z}$, Euclidean distance $\delta \in \mathbb{R}_{>0}$ between initial points $z_c \in \mathcal{Z}_0$, simulation horizon time $T \in \mathbb{R}_{>0}$, sampling time Δt, distance threshold $\eta \in \mathbb{R}_{>0}$, reduction factor distance points $\beta \in (0, 1)$, maximum number of point clusters $n_{\text{cp}} \in \mathbb{N}_{>0}$.

Result: n_{cp} clusters centers of critical points.

for n_{it} *iterations* **do**
  Define the empty set Z_0^c.
  for *each point* $z_c \in \mathcal{Z}_0$ **do**
    for n_{rp} *iterations* **do**
      Apply Algorithm 2 with z_c and δ to find z_1 and z_2.
      for z_1 *and* z_2 **do**
        Simulate the system under policy π for a horizon time of T seconds to obtain to obtain the solutions $t \mapsto \phi_i(t)$ for $i \in \{1, 2\}$.
      Compute the sum of the differences between the solutions ϕ_1 and ϕ_2 at t's where the simulation provides information: $\sum_{t=0}^{T} |\phi_1(t) - \phi_1(0) - \phi_2(t) + \phi_2(0)|\Delta t = \ d$.
      if $d > \eta$ **then**
        if *in the* n_{it}*th iteration* **then**
          Store z_c in Z_0^c.
        else
          Store z_c, z_1, and z_2 in Z_0^c.
  Apply k-means clustering on $\mathcal{Z}_0^c$ to find n_{cp} cluster centers of critical points. Store cluster centers as $\mathcal{Z}_0$.
  Set $\delta \leftarrow \beta\delta$.
  if $\delta < \Delta t$ **then**
    Set $\Delta t \leftarrow \Delta t\beta$

Algorithm 2: Generate a symmetric point pair

Data: Center point of the symmetric point pair z_c, spread of the symmetric point pair δ.

Result: Two points $z_1, z_2 \in z_c + \delta\mathbb{B}$ that are point symmetric w.r.t. z_c.

Sample a random vector of the same dimensionality as z_c from a uniform distribution: $\nu \sim \mathcal{U}(-\delta, \delta)$.

Define the point symmetric points w.r.t. z_c as $z_1 = z_c + \nu$ and $z_2 = z_c - \nu$.

algorithm converges to a continuous set of critical points as the sampling time Δt goes to zero and the simulation horizon time T and the number of cluster centers of critical points goes to infinity.

Algorithm 1 assumes that the system can be initialized from any initial condition, which may be challenging in some settings. For example, this can be problematic if the training is performed on a real system. Algorithm 1 can be adapted to work for such systems by storing and utilizing the state trajectories (solutions) obtained during training. Algorithm 1 still starts with a grid of points $\mathcal{Z}_0$. However, instead of computing a symmetric point pair (z_1, z_2) from which we simulate the system, we retrieve two solutions (or tails of solutions) that start from a δ neighborhood of z_c and evaluate the remainder of the algorithm in the same manner as before. For this adapted version of Algorithm 1, the underlying assumptions are that the state trajectories obtained during training sufficiently explore the areas of critical points and that the state is measurable to determine if a solution (or tail of a solution) starts from a δ neighborhood of z_c.

B.2 Transforming the SVM's Input

This Section describes the transformation used in Section 4.3. The transformation is performed on the input of the canonical setting's SVM model to obtain the extended overlapping sets for different obstacles or set-point positions. The function $h^{\text{trans}} : \mathcal{P} \to \mathcal{P}$ maps a position from the desired obstacle and set point setting into the canonical setting. Let $p_{\text{ob}}^{\text{can}} = (p_{x,\text{ob}}^{\text{can}}, p_{y,\text{ob}}^{\text{can}}) \in \mathcal{P}$ and $p^{*,\text{can}} = (0, 0)$ denote the obstacle position and the positional set point for the canonical setting, respectively. Furthermore, let $p_{\text{ob}}^{\text{des}} = (p_{x,\text{ob}}^{\text{des}}, p_{y,\text{ob}}^{\text{des}}) \in \mathcal{P}$ and $p^{*,\text{des}} = (p_x^{*,\text{des}}, p_y^{*,\text{des}}) \in \mathcal{P}$ denote the obstacle position and the positional set point for the desired setting, respectively. The function h^{trans} is given by

$$h^{\text{trans}}(p) = \begin{bmatrix} r^{\text{trans}}(p) \cos \phi^{\text{trans}}(p) \\ r^{\text{trans}}(p) \sin \phi^{\text{trans}}(p) \end{bmatrix}, \tag{27}$$

where r^{trans} is the ray and ϕ^{trans} is the angle of the transformed point in polar coordinates given by

$$r^{\text{trans}}(p) = r^{\text{adj}}(p) \frac{r_{\text{ob}}^{\text{can}}}{r_{\text{ob}}^{\text{adj}}}, \quad \phi^{\text{trans}}(p) = \phi^{\text{adj}}(p) - \phi_{\text{ob}}^{\text{adj}} + \phi_{\text{ob}}^{\text{can}}, \tag{28}$$

where $r_{\text{ob}}^{\text{can}}$ is the ray and $\phi_{\text{ob}}^{\text{can}}$ is the angle of p_{ob} in polar coordinates, and r^{adj} and ϕ^{adj} are given by

$$r^{\text{adj}}(p) = \sqrt{\left(p_x - p_x^{*,\text{des}}\right)^2 - \left(p_y - p_y^{*,\text{des}}\right)^2}, \quad \phi^{\text{adj}}(p) = \arctan2\left(p_y - p_y^{*,\text{des}}, p_x - p_x^{*,\text{des}}\right), \tag{29}$$

and $r_{\text{ob}}^{\text{adj}}$ and $\phi_{\text{ob}}^{\text{adj}}$ are given by

$$r_{\text{ob}}^{\text{adj}}(p) = \sqrt{\left(p_x - p_{x,\text{ob}}^{\text{adj}}\right)^2 - \left(p_y - p_{y,\text{ob}}^{\text{adj}}\right)^2}, \quad \phi_{\text{ob}}^{\text{adj}}(p) = \arctan2\left(p_y - p_{y,\text{ob}}^{\text{adj}}, p_x - p_{x,\text{ob}}^{\text{adj}}\right). \tag{30}$$

B.3 Relaxed Voronoi Partition

This section discusses the creation of the relaxed Voronoi partition used in Section 4.3 to obtain the extended overlapping sets. The Voronoi partitions are created with respect to the N obstacle locations. In a conventional Voronoi partition, the cell $i \in \{1, 2, \ldots, N\}$ is the set of points for which obstacle i is the closest in terms of Euclidean distance. As the cells are created based on which obstacle is closest, only the boundary of cells can overlap. Without overlap of these cells, the closed-loop hybrid system (7-14) in Section 4.5 can exhibit chattering or Zeno behavior around the boundaries of each cell for an infinitely small amount of noise. For more details on this chattering phenomenon and Zeno behavior, see Sanfelice (2021); de Priester et al. (2022). The conventional Voronoi partitions are relaxed by expanding the cell boundaries based on the distance to the cell's obstacle and enforcing a minimum distance to other obstacles. The latter constraint ensures that obstacles in other cells are not visible when the agent is trying to pass the current focused obstacle. Figure B.3 shows the relaxed Voronoi partition in an additional setting to the setting shown in Figure 5 in Section 4.3.

B.4 Sample-and-Hold Hybrid Systems

This section describes the hybrid systems used in Section 5 to model the normal and MultiHyRL agents as sample-and-hold control systems. The state for the normal agent is given by $x = (z, u, \tau, m) \in X_{\text{nrm}}$, where $\tau \in [0, \Delta t]$ is a timer and $m \in \mathbb{R}^4$ is the applied measurement noise. For the normal agent, the hybrid system $\mathcal{H}_{\text{nrm}} = (C_{\text{nrm}}, F_{\text{nrm}}, D_{\text{nrm}}, G_{\text{nrm}})$ is given by

$$\begin{bmatrix} \dot{z} \\ \dot{u} \\ \dot{\tau} \\ \dot{m} \end{bmatrix} = F_{\text{nrm}}(x) := \begin{bmatrix} f(z, u) \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad x \in C_{\text{nrm}}, \tag{31}$$

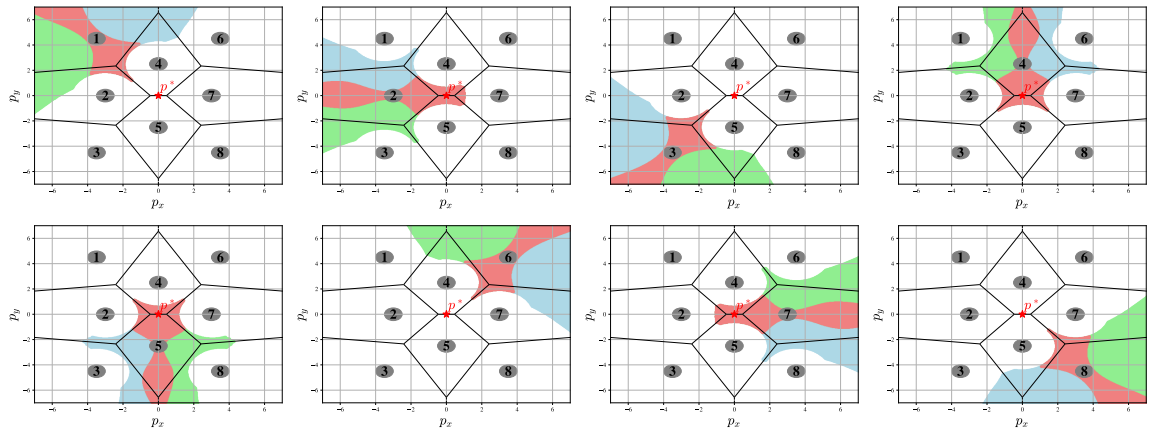


Figure 7: Visualization of the relaxed Voronoi partition for eight obstacles. For each obstacle $\ell \in \{1, 2, 3, 4, 5, 6, 7, 8\}$, the blue & red and the green & red areas denote the sets $\mathcal{M}^{\text{ext}}_{0,\ell}$ and $\mathcal{M}^{\text{ext}}_{1,\ell}$, respectively. The red area denotes the intersection $\mathcal{M}^{\text{ext}}_{0,\ell} \cap \mathcal{M}^{\text{ext}}_{1,\ell}$. The black lines denote the Voronoi vertices, the gray circles denote the obstacles, and the red circle denotes the set-point position p^*.

$$\begin{bmatrix} z^+ \\ u^+ \\ \tau^+ \\ m^+ \end{bmatrix} = G_{\text{nrm}}(x) := \begin{bmatrix} z \\ \pi_{\text{nrm}}(z+m) \\ 0 \\ G_\varepsilon \end{bmatrix} \quad x \in D_{\text{nrm}}, \tag{32}$$

where π_{nrm} is the policy corresponding to the normal agent, the flow set is given by

$$C_{\text{nrm}} := \{x \in X_{\text{nrm}} : \tau \leq \Delta t\}, \tag{33}$$

and the jump set by

$$D_{\text{nrm}} := \{x \in X_{\text{nrm}} : \tau \geq \Delta t\}, \tag{34}$$

and the noise jump map by

$$G_\varepsilon = \begin{bmatrix} [-\varepsilon_p, \varepsilon_p] \\ [-\varepsilon_p, \varepsilon_p] \\ [-\varepsilon_\xi, \varepsilon_\xi] \\ [-\varepsilon_\xi, \varepsilon_\xi] \end{bmatrix}, \tag{35}$$

where $\varepsilon_p \in \mathbb{R}_{\geq 0}$ is the magnitude of the measurement noise on the vehicle's perceived position and $\varepsilon_\xi \in \mathbb{R}_{\geq 0}$ is the magnitude of the measurement noise on the vehicle's perceived orientation. The flow map (31) captures the evolution of the vehicle and the timer, and the jump map (32) captures the controller being updated every Δt seconds. Furthermore, the observation given to the controller is perturbed by the measurement noise m. Lastly, the applied measurement noise jumps according to (35). The system flows whenever the timer's value is less or equal to the sampling time Δt, as shown in the flow set (33). The system jumps whenever the timer's value is greater or equal to the sampling time Δt, as shown in the jump set (34).

The state for simulating the sample-and-hold MultiHyRL agent is given by $x = (z, u, \tau, m, q, \lambda) \in X_{\text{Hy}}$, where $q \in \{0, 1\}$ and $\lambda \in \{1, 2, \ldots, N\}$ are the logic parameters from the MultiHyRL algorithm. For the MultiHyRL agent, the hybrid system $\mathcal{H}_{\text{Hy}} = (C_{\text{Hy}}, F_{\text{Hy}}, D_{\text{Hy}}, G_{\text{Hy}})$ is given by

$$\begin{bmatrix} \dot{z} \\ \dot{u} \\ \dot{\tau} \\ \dot{m} \\ \dot{q} \\ \dot{\lambda} \end{bmatrix} = F_{\text{Hy}}(x) := \begin{bmatrix} f(z,u) \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad x \in C_{\text{Hy}}, \tag{36}$$

$$\begin{bmatrix} z^+ \\ u^+ \\ \tau^+ \\ m^+ \\ q^+ \\ \lambda^+ \end{bmatrix} = G_{\text{Hy}}(x) := \begin{bmatrix} z \\ \begin{cases} \pi_0(z+m) & \text{if} \quad q = 0 \\ \pi_1(z+m) & \text{if} \quad q = 1 \end{cases} \\ 0 \\ G_\varepsilon \\ \begin{cases} 0 & \text{if} \quad x + \rho(m) \in D^{\text{hyst}}_{0,\lambda} \\ 1 & \text{if} \quad x + \rho(m) \in D^{\text{hyst}}_{1,\lambda} \end{cases} \\ \Gamma(x+\rho(m)) \quad \text{if} \quad x + \rho(m) \in \overline{D \setminus D^{\text{fcs}}_\lambda} \end{bmatrix} \quad x \in D_{\text{Hy}}, \tag{37}$$

where the MultiHyRL algorithm is used to find the control policies π_0 and π_1, the jump sets $D^{\text{hyst}}_{0,\lambda}$, $D^{\text{hyst}}_{1,\lambda}$, and D^{fcs}_λ, and $\rho(m)$ is the perturbation on the state defined as

$$\rho(m) = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\top. \tag{38}$$

The jump map Γ is given by (13). The flow set is given by

$$C_{\text{Hy}} := \{x \in X_{\text{Hy}} : \tau \leq \Delta t\}, \tag{39}$$

and the jump set by

$$D_{\text{Hy}} := \{x \in X_{\text{Hy}} : \tau \geq \Delta t\}. \tag{40}$$

Note that by implementing the MultiHyRL agent as a sample-and-hold controller, we can only update our logic variables q and λ during timer jumps.

C Additional Simulations

This section provides the additional simulations mentioned in the main text. Three additional sets of simulations are performed to highlight the effectiveness of the MultiHyRL algorithm: moving obstacles, obstacles with various sizes, and a Capture the Flag between the normal and MultiHyRL agents. As mentioned in the main text, the controllers are implemented with a sample-and-hold approach; discussed in Appendix B.4.

C.1 Moving Obstacles

In this set of simulations, the obstacles are dynamic, have the same radius, have a spacing between each obstacle greater than $\frac{1}{2}\sqrt{2}w_{\text{bew}}$, and their velocity is less than 1, such that the vehicle can outrun an obstacle. Figure 8 shows still frames from the simulation in which the set-point position changes from $p^* = (p^*_x, p^*_y) = (3, 0.1)$ to $p^* = (3, -0.1)$ every ten seconds. The normal and MultiHyRL agents are deployed in the same environment in the presence of the *exact same* measurement noise signal, given by (15). The measurement noise signal consists of a positional measurement noise of magnitude $\varepsilon_p = 0.2$ and no orientational measurement noise, namely, $\varepsilon_\xi = 0$. Figure 8 shows that both the normal and MultiHyRL agents are effective in environments with dynamic obstacles. However, the MultiHyRL agent outperforms the normal agent as the normal agent crashes into some obstacles in the presence of the positional measurement noise. In contrast, the MultiHyRL does not crash into any obstacle in the presence of positional measurement noise, thanks to the MultiHyRL agent's robustness properties.

C.2 Varying Obstacle Size

In this set of simulations, the obstacles satisfy the properties listed in Section 3.1 except that the radius of the obstacles varies. The same measurement noise and obstacle setting used in Section 5 are considered with varying obstacle radius. Figure 9 shows the result of these simulations. The same discussion described in Section 5 applies here. In essence, the hysteresis and obstacle focus mechanism of the MultiHyRL agent allows it to reach the set point for all the scenarios considered, contrary to the normal agent, which gets stuck for several initial conditions.

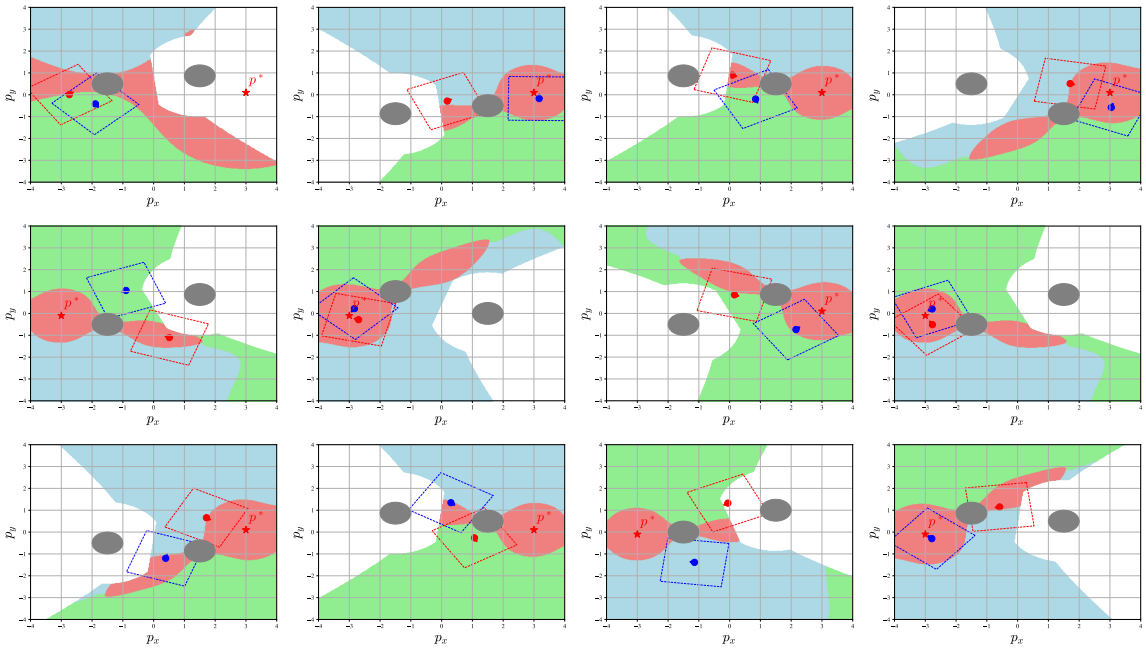
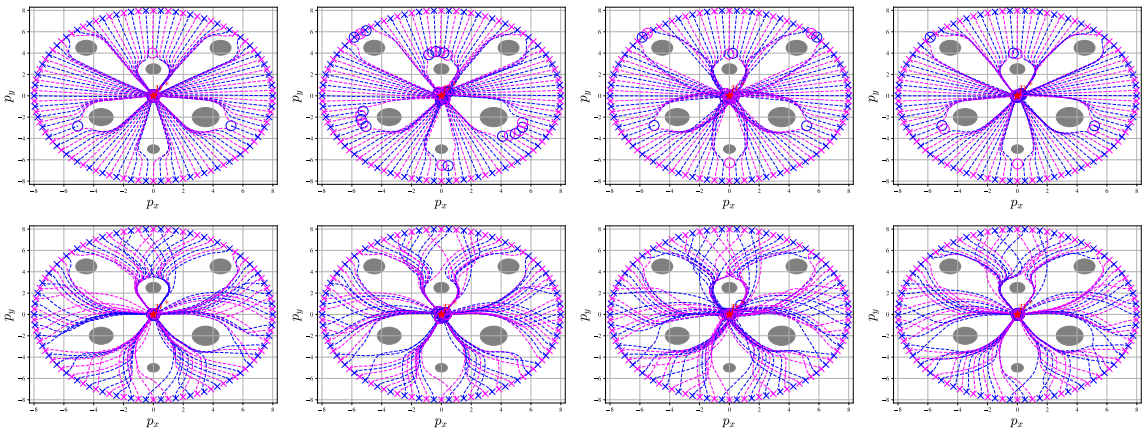


Figure 8: Still frames of the dynamic obstacle avoidance simulation in the presence of positional measurement noise of magnitude $\varepsilon_p = 0.2$ for the normal and MultiHyRL agents. The logic variables q and λ are randomly initialized for the MultiHyRL Agent. The normal agent and its bird's-eye view image are represented by the red triangle with a semicircle and the red box, respectively. The MultiHyRL agent and its bird's-eye view image are represented by the blue triangle with a semicircle and the blue box, respectively. For the MultiHyRL agent, the extended overlapping sets $\mathcal{M}_{0,\ell}^{\text{ext}}$ and $\mathcal{M}_{1,\ell}^{\text{ext}}$ are drawn for the currently focused obstacle $\ell \in \{1, 2, \ldots, N\}$. The red $\star$ denotes the set-point position p^* for the current still frame.

C.3 Capture the Flag

In this set of simulations, the normal and MultiHyRL agents play a simplified game of Capture the Flag against each other. However, in this game, measurement noise and obstacles are present. The agents are given the same observations as discussed in Section 3.1, namely, the planar position error with respect to a set-point position, the orientation error, and a bird's-eye view image. The rules of the game are as follows. Each player has a flag that is located at their base. At the start of the game, each player starts in their base. Each player's set-point position is set to the opposing player's flag. If player A grabs player B's flag, player A's set-point position is set to player A's base, and player B's set-point position is set to player A's position. If player B tags player A before player A reaches its base, player A is reset to its base, and player B's flag is returned. If player A reaches its base untagged, player A scores a point, and player B's flag is returned to player B's base. If the players collide and no one holds a flag, they are reset to their bases. Figure 10 shows still frames from a game of Capture the Flag with measurement noise. The normal and MultiHyRL agents undergo the *exact same* measurement noise signal, given by (15). The measurement noise signal consists of a positional measurement noise of magnitude $\varepsilon_p = 0.2$ and no orientational measurement noise, namely, $\varepsilon_\xi = 0$. Figure 10 shows that the normal agent gets stuck numerous times in front of an obstacle, which allows the MultiHyRL agent to score points easily. It can also be seen that the MultiHyRL agent seems unaffected by the noise thanks to the MultiHyRL agent's robustness properties.



(a) $\varepsilon_p = \varepsilon_\xi = 0$. (b) $\varepsilon_p = 0.2$ and $\varepsilon_\xi = 0$. (c) $\varepsilon_p = 0$ and $\varepsilon_\xi = 0.3$. (d) $\varepsilon_p = 0.05$ and $\varepsilon_\xi = 0.1$.

Figure 9: Visualization of the state trajectories under different measurement noise settings for the normal agent (row 1) versus the MultiHyRL agent (row 2) in a static obstacle environment with varying obstacle radii. The logic variables q and λ are randomly initialized for the MultiHyRL agent. The trajectories alternate colors for readability. The gray circles denote obstacles, the $\times$'s denote the initial conditions, the $\circ$'s denote the terminal positions, and the red $\star$ denotes the set-point position p^*.

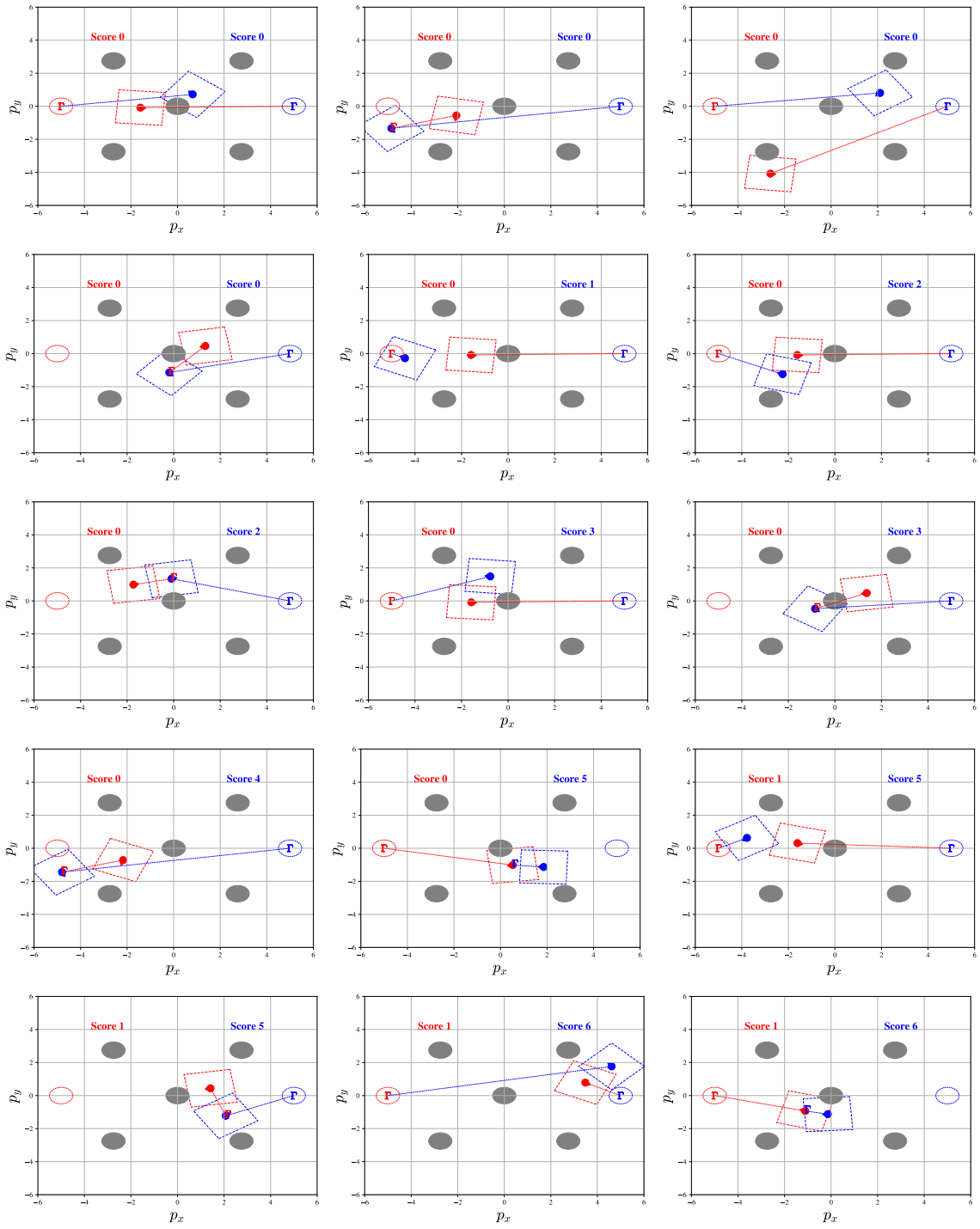


Figure 10: Still frames of the Capture the Flag simulations in the presence of positional measurement noise of magnitude $\varepsilon_p = 0.2$ for the normal and MultiHyRL agents. The logic variables q and λ are randomly initialized for the MultiHyRL Agent. The normal agent, its current set-point position, and its bird's-eye view image are represented by the red triangle with a semicircle, the red dotted line, and the red box, respectively. The MultiHyRL agent, its current set-point position, and its bird's-eye view image are represented by the blue triangle with a semicircle, the blue dotted line, and the blue box, respectively. The flags are denoted by Γ.