Composable Model-Free RL for Navigation with Input-Affine Systems

Xinhuan Sang*[0009−0002−0340−082X], Abdelrahman Abdelgawad[0009−0005−1433−0072] and Roberto Tron[0000−0002−6676−8595]

Boston University, Boston MA 02215, USA
leosang@bu.edu, aaoaa@bu.edu, tron@bu.edu

Abstract As autonomous robots are increasingly deployed in real-world environments, they must learn to navigate safely in real time. However, anticipating and learning all possible system behaviors is infeasible. To address this, we propose a composable, model-free Reinforcement Learning method that *1)* learns a value function and an optimal policy for each individual element in the system; and *2)* combines them to achieve collision avoidance and goal-directed behavior. The lynchpin of our approach is that the system and environment are unknown and nonlinear, but are assumed to evolve in *continuous time* and be *input-affine* (many electromechanical systems of practical interest can be considered in this category). Based on these assumptions, we make three contributions. First, we derive the Hamilton-Jacobi-Bellman (HJB) equation for the *value function* V, and show that the corresponding *advantage function* A *is a quadratic function* of the input actions u and optimal policy u^*. Second, we propose a novel model-free actor-critic method to *learn static or moving obstacles via optimal policies and value functions*, which uses gradient descent, and is based on the quadratic form of the advantage function mentioned above. Third, we show that our models for individual reach and avoid tasks can be *composed online through Quadratically Constrained Quadratic Programs (QCQP)*, with formal guarantees on the avoidance of obstacles (modeled as level sets of the individual value functions), providing a model-free alternative to the popular model-based Control Lyapunov and Barrier Functions methods from the nonlinear control literature. We show the effectiveness of our method in simulations compared with a baseline obtained by applying Proximal Policy Optimization (PPO) to a discrete-time approximation of the system.

Keywords: Collision Avoidance · Reinforcement Learning · Modular Learning.

1 Introduction

Robots are increasingly transitioning from controlled environments such as laboratories, factories, and warehouses into complex, dynamic conditions of daily life. However, it is infeasible to anticipate and train robots to handle every possible combination of goals and obstacle configurations that they may encounter in such

settings. This makes it a pressing challenge in robotics to develop control methods that enable robots to operate safely and collision-free, while still achieving their goals in such environments.

Reinforcement Learning (RL) offers a promising data-driven approach to address this challenge, allowing an agent to learn value functions and optimal policies through direct interaction with the system [13]. In many cases, model-free RL has demonstrated the ability to learn complex behaviors that would be difficult to design manually without prior knowledge of system dynamics or environmental models [5]. RL has been successfully applied to control problems with relatively simple dynamics but complex free configuration spaces (e.g., multi-UAV path planning when facing multiple threats [17] and motion planning in an environment with multiple, possibly moving obstacles [8]). However, these successes have focused on discrete state-space formulations, and the learned policies typically apply only to specific task-environment configurations. When the environment or task changes significantly, retraining is usually required.

On the other hand, control theory, particularly linear and nonlinear control, provides provably safe feedback controllers with strong theoretical guarantees. However, these methods typically require accurate system models. An effective class of such controllers relies on Control Lyapunov Functions (CLFs) and Control Barrier Functions (CBFs), which measure progress toward goals and safety relative to obstacles, respectively. By solving a Quadratic Program (QP), it is possible to find control inputs that simultaneously ensure convergence and safety [1]. The CLF-CBF approach is robust under mild regularity assumptions (e.g., Lipschitz continuity) and naturally scales to complex environments by adding one linear constraint for each obstacle. Although conceptually similar to the traditional potential field methods [11], CLF-CBF QPs offer broader applicability and superior empirical performance. Applications include locomotion control for bipedal robots [7] and integration with sampling-based path planning [16]. Nevertheless, these methods rely heavily on explicit expressions for system dynamics and the environment, limiting their practical deployment in uncertain situations, or where an analytical model is difficult to obtain from first principles.

This study introduces a new approach that combines the strengths of both paradigms by developing a model-free CLF-CBF-like control method for input-affine systems in continuous time and space. A core concept in RL is the value function, which estimates the expected discounted cumulative reward or cost starting from a given state following optimal control [15]. We link this notion to CLFs and CBFs in control theory and propose learning CLF-CBF-like actor-critic RL models. To support this, we derive a form of the Hamilton-Jacobi-Bellman (HJB) equation specific to input-affine systems, applicable to many electromechanical platforms of practical interest. Our derivation links the HJB residual to the advantage function and shows that it is quadratic in the optimal policy, providing a compact, model-free method to express the terms required in the CBF constraints for the online composition of policies.

As shown in later sections, any function-approximation framework that provides gradient predictions during training can be used with the proposed method.

We employ Gaussian Processes (GPs) [10] as an interpolation method over sparse data to encode continuous, smooth value functions, and optimal policies across the state space. However, our method can also be applied to other learning frameworks (e.g., Neural Networks), provided that certain necessary conditions are met. During training, we learn individual value functions (critics) and optimal policies (actors) for goals and obstacles separately. At runtime, these learned results are composed via a QP by incorporating them as linear constraints based on the current configuration of goals and obstacles, enabling real-time collision-free navigation.

To summarize, our approach has three main contributions:

- We derive a novel advantage function definition based on the HJB residual for input-affine systems. Form an actor-critic learning method based on its quadratic nature with input u and optimal policy u^*.
- From a control-theoretic perspective, our method eliminates the need for explicit system and environment models by employing a novel continuous-time model-free actor-critic learning method.
- From the reinforcement learning side, generalization is enhanced by modularizing the learning process around composable environmental elements. It provides a formal collision-avoidance guarantee (based on the learned model) via CBF-like constraints for each obstacle.

2 Preliminary Concepts

This section provides an overview of the fundamental concepts in control theory and Reinforcement Learning on which our proposed approach is based.

2.1 Input-Affine System

Input-affine systems represent a class of nonlinear systems where the dynamics are linear-affine in the control input [3]:

$$\dot{x} = F(x, u) = f(x) + g(x)u, \tag{1}$$

where $x \in X$ is the state of the system, $u \in U$ is the control input, and the mapping $F(x, u)$ that defines the dynamics is composed of the *drift* vector field $f(x)$, and the *control* vector field $g(x)$. A key consequence of having a dynamics in the form of Eq. 1 is that the derivative (with respect to time) of any function V evaluated along the trajectories of the system, that is, its *Lie derivative*, will be linear-affine with respect to the control inputs $u(t)$, that is,

$$\dot{V} = \frac{d}{dt}V\big(x(t)\big) = \nabla_x V^\intercal \dot{x} = \nabla_x V^\intercal F(x, u) = \nabla_x V^\intercal f(x) + \nabla_x V^\intercal g(x)u. \tag{2}$$

While all the forms in Eq. 2 are mathematically equivalent, in the following we will choose different forms depending on what is most useful in the context.

2.2 CBF Constraint

Given a CBF $H(x) : \mathcal{D} \to \mathbb{R}$, define the associated safe set

$$\mathcal{C} \doteq \{x \in \mathcal{D} \mid H(x) \geq 0\}. \tag{3}$$

We say that $\mathcal{C}$ is *forward invariant* for the (closed-loop) dynamics if any trajectory starting in $\mathcal{C}$ remains in $\mathcal{C}$ for all future times (i.e., $x(0) \in \mathcal{C} \Rightarrow x(t) \in \mathcal{C}$ for all $t \geq 0$, over the interval of existence of the solution). Showing that $\mathcal{C}$ is forward invariant is equivalent to showing that $H(x(0)) \geq 0 \Rightarrow H(x(t)) \geq 0$ for all $t \geq 0$; a sufficient condition to achieve this is given by the *CBF constraint* [1]

$$\dot{H} + c_h H = \nabla_x H^\intercal \dot{x} + c_h H \geq 0. \tag{4}$$

The constant $c_h \geq 0$ controls how conservative the constraint is; for example, $c_h = 0$ corresponds to the most conservative case where H is never allowed to decrease.

2.3 Advantage Function

In the field of discrete time RL [2, 14], the *value function* $V(x)$ represents the total expected reward/cost when starting at state x and following optimal control $u^*(x)$. The *state-action value function* $Q(x, u)$ represents the total expected reward/cost when starting in state x and taking action u and then followed by optimal control u^*. The *advantage function* $A(x, u)$ represents the difference

$$A(x, u) = Q(x, u) - V(x). \tag{5}$$

By definition, we have:

$$A(x, u^*) = \min_u A(x, u) = \min_u Q(x, u) - V(x) = Q(x, u^*) - V(x) = 0. \tag{6}$$

3 Theoretical Derivation

3.1 Hamilton–Jacobi–Bellman Equation

The Hamilton–Jacobi–Bellman (HJB) equation is a fundamental result in optimal control theory that provides a necessary condition for optimality in continuous-time, continuous-state decision-making problems [4], formulated as a nonlinear partial differential equation. This section provides an HJB derivation for a time-discounted setting under state-dependent termination conditions.

We consider the trajectories of the system $x(t)$ (episodes) uniquely determined by an input signal $u(t)$ and an initial condition $x(t_0) = x_0$, terminating at time $T(x_0, u)$ according to user-defined state conditions (e.g., hitting an obstacle, reaching the goal, or reaching an out-of-bound region). Consider a discount factor λ and no terminal cost. The value function V is then defined as

$$V(x_0, t_0) = \min_{u \in \mathcal{U}} \int_{t_0}^{T(x_0,u)} e^{-\lambda(t-t_0)} C(x(t), u(t)) dt \tag{7}$$

where $\mathcal{U}$ is the set of right-continuous control signals defined for $t \geq t_0$, $x(t)$ is the solution to Eq. 1 uniquely determined by u and x_0, $C(x,u)$ is a user-defined function that defines the cost of taking the infinitesimal action u at location x infinitesimal cost function.[1] Starting from a later state $x_h \doteq x(t_0+h)$, the HJB equation is similar:

$$V(x_h, t_0+h) = \min_{u\in\mathcal{U}} \int_{t_0+h}^{T(x_0,u)} e^{-\lambda(t-(t_0+h))} C(x(t),u(t))dt \tag{8}$$

Breaking the integral, and substituting the second term with Eq. 8 we obtain:

$$V(x_0,t_0) = \min_{u\in\mathcal{U}} \left\{ \int_{t_0}^{t_0+h} e^{-\lambda(t-t_0)} C(x(t),u(t))dt + e^{-\lambda h} V(x_h, t_0+h) \right\} \tag{9}$$

Subtracting $V(x_0,t_0) = e^{\lambda 0} V(x_0,t_0)$ on both sides we get:

$$0 = \min_{u\in\mathcal{U}} \left\{ \int_{t_0}^{t_0+h} e^{-\lambda(t-t_0)} C(x(t),u(t))dt + e^{-\lambda h} V(x_h,t_0+h) - e^{\lambda 0} V(x_0,t_0) \right\}. \tag{10}$$

Assuming time-invariant dynamics $\dot{x} = F(x,u)$, dividing both sides by h, adding and removing $e^{-\lambda h} V(x_0,t_0)$, and taking the limit $h \to 0$, derivatives appear as follows:

$$\begin{aligned}
0 &= \lim_{h\to 0} \frac{1}{h} \min_{u\in\mathcal{U}} \left\{ \int_{t_0}^{t_0+h} e^{-\lambda(t-t_0)} C(x(t),u(t))dt + e^{-\lambda h} V(x_h,t_0+h) - e^{\lambda 0} V(x_0,t_0) \right\} \\
&= \lim_{h\to 0} \frac{1}{h} \min_{u\in\mathcal{U}} \left\{ \int_{t_0}^{t_0+h} e^{-\lambda(t-t_0)} C(x(t),u(t))dt + \right. \\
&\qquad \left. e^{-\lambda h}\big(V(x_h,t_0+h) - V(x_0,t_0)\big) + \big(e^{-\lambda h} - e^{\lambda 0}\big) V(x_0,t_0) \right\} \\
&= \min_{u\in\mathcal{U}} \left\{ e^{-\lambda(0)} C(x_0,u_0) + e^{-\lambda 0} \left.\frac{d}{dt} V(x,t)\right|_{t=t_0} + \left.\frac{d}{dt} e^{-\lambda t}\right|_{t=0} V(x_0,t_0) \right\}
\end{aligned} \tag{11}$$

With time-invariant dynamics, the result of Eq. 7 depends only on the starting state x_0 and not on the initial time t_0. Thus, we can substitute the term $\frac{d}{dt} V(x,t)|_{t=t_0}$ with Eq. 2 and drop the dependency on a specific time t_0:

$$0 = \min_u \left\{ C(x,u) - \lambda V(x) + \nabla_x V(x)^\intercal F(x,u) \right\}. \tag{12}$$

3.2 Special Form of the Advantage Function

By analogy between the HJB and the corresponding Bellman equation in discrete time [2, 14] and between Eq. 12 and Eq. 6 we define the *differential advantage function* $A(x,u)$ as the argument of the min operator of Eq. 12:

$$A(x,u) = C(x,u) - \lambda V(x) + \nabla_x V(x)^\intercal F(x,u). \tag{13}$$

[1] The definition and derivations are valid with a terminal cost after minor modifications, but this is not used in this paper.

With this definition, the HJB equation becomes a special condition on the advantage function and for the discrete time case in Eq. 6, for the optimal control u^*, we have

$$A(x, u^*) = \min_u A(x, u) = 0. \tag{14}$$

Furthermore, under practical conditions of the unknown dynamics F and user-defined cost C, the advantage $A(x, u)$ takes a form that facilitates learning.

Proposition 1. *Assume an unknown input-affine dynamics of form Eq. 1, and a quadratic cost function $C(x, u)$ of form*

$$C(x, u) = \frac{1}{2}(u - u_c(x))^\intercal R(x)(u - u_c(x)) + q_c(x), \tag{15}$$

where $R(x)$ is positive semidefinite, and $u_c(x)$ is a vector-valued function. Then the differential advantage function $A(x, u)$ is quadratic in the optimal control $u^*(x)$, specifically:

$$A(x, u) = \frac{1}{2}\big(u - u^*(x)\big)^\intercal R(x)\big(u - u^*(x)\big). \tag{16}$$

Proof. With the assumptions above and using $f(x)$ and $g(x)$ as placeholders for the unknown dynamics, Eq. 13 becomes:

$$A(x, u) = C(x, u) - \lambda V(x) + \nabla_x V^\intercal f(x) + \nabla_x V^\intercal g(x)u. \tag{17}$$

Because the terms on the right-hand side are either quadratic in u (i.e., $C(x, u)$) or linear in u (i.e., $\nabla_x V^\intercal g(x)u$), the advantage function $A(x, u)$ must also be quadratic in u, i.e., generically:

$$A(x, u) = (u - u_a(x))^\intercal S(x)(u - u_a(x)) + D(x)(u - u_a(x)) + E(x). \tag{18}$$

From Eq. 14, the minimum of $A(x, u)$ is zero and the minimum is achieved at $u = u^*(x)$, hence $E(x), D(x) = 0$. Moreover, matching only the quadratic term in Eq. 17, which is $C(x, u)$, yields $S(x) = \frac{1}{2}R(x)$. The claim follows.

4 Actor-Critic Learning for Single Goals and Obstacles

Based on our special form of the advantage function for input-affine systems, we construct our proposed actor-critic algorithm that we will use to model individual goals or obstacles (before composing them through the QCQP). Following the definition of actor-critic algorithms [14], we intuitively have an actor represented by the optimal policy $u^*(x)$, and a critic represented by the value function $V(x)$ [6]. For simplicity of exposition, we use estimates of the value function $\hat{V}(x)$, of the gradient of the value function $\nabla_x \hat{V}(x)$, and of the optimal policy $\hat{u}^*(x)$ that are linear in the parameters:

$$\hat{V}(x) = \mathcal{M}_V(x)\theta_V, \tag{19a}$$

$$\nabla_x \hat{V}(x) = \mathcal{M}'_V(x)\theta_V, \tag{19b}$$

$$\hat{u}^*(x) = \mathcal{M}_u(x)\theta_{u^*}, \tag{19c}$$

where θ_V and θ_{u^*} are the parameters and data used by the value function model $\mathcal{M}_V$ and optimal policy model $\mathcal{M}_u$, respectively. Note that the model for obtaining the gradient of the value function, $\mathcal{M}'_V$, is based on the same parameters and data as in the model $\mathcal{M}_V$.

The parameters $R(x)$, $u_c(x)$, and $q_c(x)$ in the cost function are part of the design of the learning algorithm (i.e., determining how the instantaneous cost is determined). In this case, we chose $R(x) = I$, $q_c(x) = 0$, and $u_c(x) = 0$. Then we have that the cost function Eq. 15 simplifies, and we define two different models for the advantage function, one from Eq. 13 based on critic $\hat{V}(x)$, and one from Eq. 16 based on actor $\hat{u}^*(x)$:

$$C(x_i, u) = \frac{1}{2} u^\intercal u, \tag{20a}$$

$$\hat{A}_{critic}(x_i, u) = \frac{1}{2} u^\intercal u + \nabla_x \hat{V}(x_i)^\intercal \dot{x} - \lambda \hat{V}(x_i). \tag{20b}$$

$$\hat{A}_{actor}(x_i, u) = \frac{1}{2} \|u - \hat{u}^*(x_i)\|^2, \tag{20c}$$

We define loss as the difference between the *critic* and *actor* advantage functions:

$$Loss = \frac{1}{2} (\hat{A}_{actor}(x, u) - \hat{A}_{critic}(x, u))^2. \tag{21}$$

We update θ_V and θ_{u^*} using gradient descent:

$$\frac{\partial Loss}{\partial \theta_V} = (\hat{A}_{actor} - \hat{A}_{critic})(\lambda \mathcal{M}_V(x_i) - \mathcal{M}'_V(x_i)\dot{x}), \tag{22a}$$

$$\frac{\partial Loss}{\partial \theta_{u^*}} = -(\hat{A}_{actor} - \hat{A}_{critic})\big(u - \hat{u}^*(x)\big)\mathcal{M}_u(x_i), \tag{22b}$$

$$\theta_V \leftarrow \theta_V - \eta \frac{\partial Loss}{\partial \theta_V}, \tag{22c}$$

$$\theta_{u^*} \leftarrow \theta_{u^*} - \eta \frac{\partial Loss}{\partial \theta_{u^*}}. \tag{22d}$$

Remark 1. While we used the model $f(x), g(x)$ for the derivation, we do not need to know it for computing the gradient updates in Eq. 22. Similarly to how the discrete case assumes that we observe pairs of states $x[k], x[k+1]$, in our continuous-time case we assume that we observe pairs $x(t), \dot{x}(t)$.

5 Modular Training of Single Goals and Obstacles

A central design choice in our framework is to treat a navigation scene not as a single monolithic RL task but as a *composition* of reusable behaviors associated with individual *environment elements*, namely goals and obstacles. This mirrors the way CLF-CBF controllers scale to complex scenes by introducing one constraint per obstacle and combining them online in a QCQP, but without requiring explicit models of the system dynamics or obstacle geometry.

5.1 Element-wise Training in Local Coordinate

Let $\mathcal{E}$ denote the set of elements present in a scene: one goal element e_g and M obstacle elements $\{e_i\}_{i=1}^{M}$. For each element $e_j \in \mathcal{E}$, we define a state x_j, which includes the agent state (e.g., pose, velocity, and other available parameters relative to the element) and the element's parameters (e.g., shape descriptors when available). The state $x_j \in X_j$ is only relative to the goal/obstacle element, so the learned functions are not affected across global layouts by construction.

For *each* element e_j we train critic $V_j(x_j)$ and actor $u_j^*(x_j)$ using the continuous-time HJB residual from Eq. 17. The termination set $\mathcal{X}_{j,term} \subset X_j$ is the contact/reach set; we enforce $V_j(x) = 0$ for $x \in \mathcal{X}_{j,term}$. Importantly, *goals and obstacles are trained in the same manner*: we always define a *reaching* task whose termination set corresponds to entering the element. For the goal, termination occurs upon reaching the goal set. For an obstacle, termination occurs upon *contact* with the obstacle. Under this convention, $u_j^*(x_j)$ points in the locally optimal *approach-to-contact* direction, and $V_j(x_j)$ measures the discounted cost-to-contact. Therefore, the distinction between the *goal* and *obstacle* arises *only at deployment time* through how these learned functions are incorporated as linear constraints in the online QCQP.

5.2 Gaussian Process Implementation

We adopted Gaussian Processes (GPs) [10] as the function-approximation backbone to validate our theoretical development for three main reasons: *1)* GP regression provides a modeling framework in which predictions are expressed directly in terms of the data through kernel evaluations, making it straightforward to interpret how the dataset shapes the learned function. *2)* GP posteriors define a distribution over functions on a continuous input domain. With commonly used kernels, the posterior mean is continuous (and, depending on kernel regularity, sample paths can be smooth), which aligns naturally with our continuous-state setting. *3)* When the kernel is differentiable, derivatives of a GP are themselves GPs, and function values and derivative values are jointly Gaussian; therefore, gradients can be obtained in a principled and convenient manner from the same GP model.

In practice, a discrete subset of the state space X, marked as $X^\sharp \subset X$, is the state where we store the mean of the optimal policy μ_{u^*} and the value function μ_V. For state $x_i \in X$, we can obtain a prediction of the optimal policy $u_p^*(x_i)$ and the value function $\hat{V}(x_i)$ as Gaussian Processes:

$$J_{x_i} = K_{x_i,X^\sharp} K_{X^\sharp,X^\sharp}^{-1}, \tag{23a}$$

$$\hat{u}^*(x_i) = J_{x_i} \mu_{u^*}, \tag{23b}$$

$$\hat{V}(x_i) = J_{x_i} \mu_V, \tag{23c}$$

where $K_{x_i,X^\sharp}$ is the covariance between the prediction state x_i and the data location $X^\sharp$, and $K_{X^\sharp,X^\sharp}$ is the covariance matrix for all possible combinations

within the data location $X^\sharp$. The elements of the covariance matrix are defined by the kernel function k; in our case, we use the RBF kernel $k(x_i, x_j)$,

$$K_{i,j} = k(x_i, x_j) \quad x_i, x_j \in X^\sharp \tag{24}$$

Furthermore, with a differentiable kernel, function values and partial derivatives are jointly Gaussian. Therefore, we obtain gradients directly from the same GP posterior (we use the posterior mean in our implementation) to compute the Lie derivative of the value function [10]:

$$J'_{x_i} = K'_{x_i, X^\sharp} K^{-1}_{X^\sharp, X^\sharp}, \tag{25a}$$

$$K'_{x_i, X^\sharp} = \left[\tfrac{\partial}{\partial [x]_1} K(x_i, X^\sharp) \cdots \tfrac{\partial}{\partial [x]_m} K(x_i, X^\sharp) \right], \tag{25b}$$

$$\nabla_x \hat{V}(x_i) = J'_{x_i} \mu_V, \tag{25c}$$

where $\frac{\partial}{\partial [x]_j}$ is the partial derivative of the j^{th} dimension of state $x \in \mathbb{R}^m$. For K, this partial derivative is applied element-wise.

Substituting $\mathcal{M}_\mathcal{V} = J_{x_i}$, $\mathcal{M}'_V = J'_{x_i}$, and $\mathcal{M}_u = J_{x_i}$ into Eq. 22 yields gradient-descent updates for the stored base-point means (μ_V, μ_{u^*}).

5.3 Discrete-time Baseline using Proximal Policy Optimization (PPO)

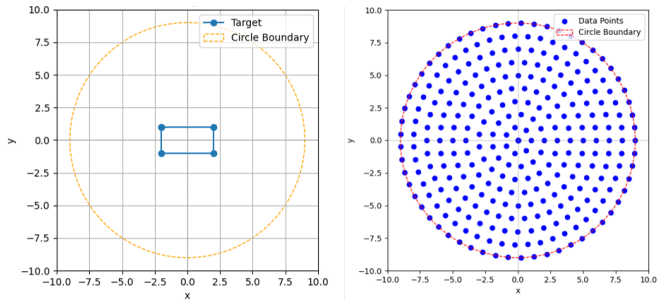
To further evaluate our system, we compared it with Proximal Policy Optimization (PPO) [12]. PPO is an on-policy actor–critic algorithm that alternates between

- collecting rollouts under the current stochastic policy $u_\theta(x)$ and
- updating θ by maximizing a clipped surrogate objective while fitting a critic $V_\phi(x)$ to predict the discounted return [12].

We use PPO to train a critic and an actor neural network. We use the Stable-Baselines3 implementation of PPO [9], which performs multiple epochs of minibatch updates per rollout buffer, including common hyperparameters such as the rollout horizon n_{steps}, GAE λ_{GAE}, and clipping range [9]. The PPO critic $V_\phi(x)$ estimates the expected discounted return $\mathbb{E}[\sum_{k\geq 0} \gamma^k r_k]$. Under our choice $r_k = -\Delta t\, C(x_k, u_k)$ (plus terminal shaping), V_ϕ represents a negative cost-to-go (up to discretization and shaping). The optimal control action u^* is obtained as the mean action of the policy π.

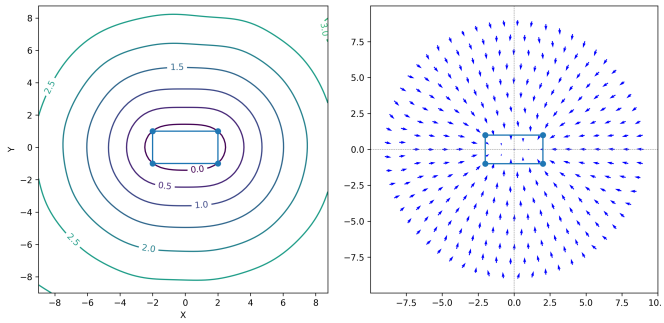
5.4 Simulation Results for Single Static and Moving Elements

Simulation setup. Here, we show the results of learning individual elements. To emphasize that our approach does not require an explicit model of the element's shape or dynamics, we consider navigation tasks in $\mathbb{R}^2$ in which the learner observes only a *relative position* state, while key dynamical effects remain unobserved. Each environmental element e_j (a goal region or obstacle) induces a local navigation objective. To make these behaviors interpretable, after training, we evaluated each actor-critic pair on a dense grid around the element and visualized (i) the learned value $\hat{V}_j(x_j)$ as a contour map and (ii) the learned control $\hat{u}^*_j(x_j)$ as a vector field in the coordinates of the element.



(a) Range and element (b) Model data points

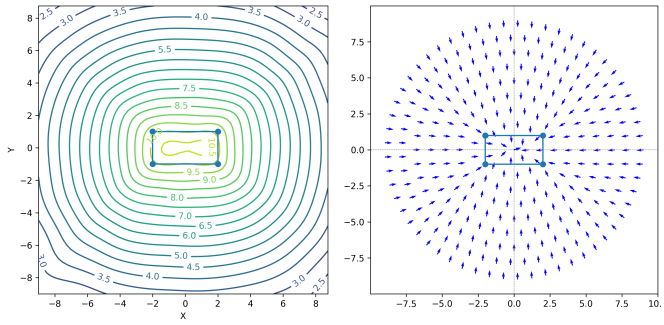
Figure 1: Training range setup for a typical 2-dimensional navigation scenario: (a) shows a circular state space (orange dashed circle) around the element (blue rectangle) and (b) shows the $X^\sharp$ used to store μ_V and μ_{u^*} as blue dots.



(a) $V(x)$ (b) $u^*(x)$

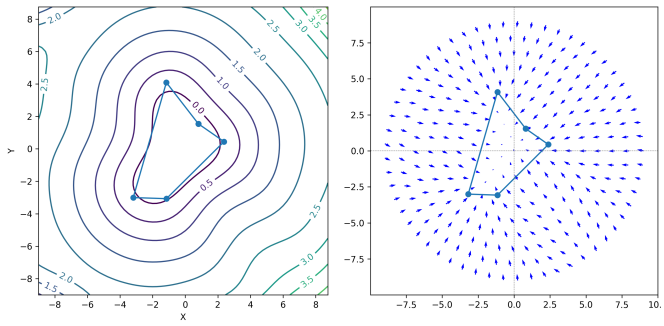
Figure 2: GP training results for 4×2 stationary rectangle element with 20000 epochs and maximum 100 steps for each epoch: (a) shows the contours of the learned value function and (b) shows the learned control vector field.

Stationary elements. Fig 1 and 2 show the basic training environment setup and GP results for a 2-dimensional rectangle element. Fig. 3 shows the baseline PPO training results for the same rectangle as the previous GP results. The proposed approach can also be easily trained on random polygons, as shown in Fig. 4. In Fig. 2 and 3, both GP and PPO learn qualitatively similar structures in the value function and control vector field. The primary distinction lies in the sign convention: our method minimizes cost (so V decreases toward contact), whereas PPO maximizes reward (so V_ϕ increases toward the obstacle). This agreement confirms that our continuous-time HJB formulation captures the same fundamental structure as standard discrete-time RL, with the PPO critic differing only in sign and scaling.



(a) $V_\phi(x)$ (b) $u_\theta(x)$

Figure 3: PPO training results for 4×2 stationary rectangle element with 20000 epochs and maximum 100 steps for each epoch: (a) shows the contours of the learned value function and (b) shows the mean policy control vector field.



(a) $V(x)$ (b) $u^*(x)$

Figure 4: GP training results for a stationary random polygon with 20000 epochs and maximum 100 steps for each epoch: (a) shows the contours of the learned value function and (b) shows the learned control vector field.

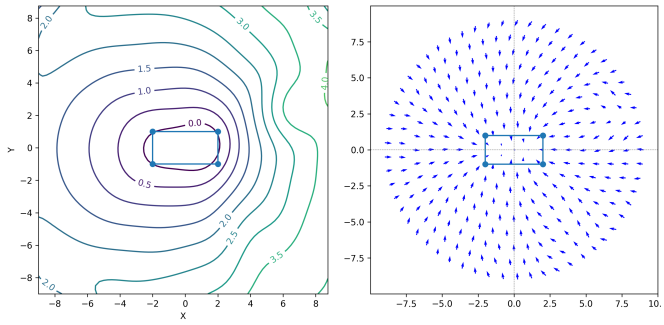
Moving elements. We *deliberately omitted* the relative velocity (and any element velocity) from the learner's state. In the simulation, the agent was controlled using a bounded planar input $u \in \mathbb{R}^2$ (with $\|u\| \leq u_{\max}$), and the relative dynamics take the generic form of an input-affine system Eq. 1 (constant velocity motion), which however remains unknown to the learning algorithms.

The goal of these results is to determine whether the effect of the hidden motion can be captured in the learned value function and policy.

These plots allow us to directly inspect how the learned critic and actor encode the hidden motion of an element.

The results for the moving elements are shown in Fig. 5.

Intuitively, when e is moving, the hidden drift must be compensated by the policy, and this compensation appears as systematic asymmetries in the learned

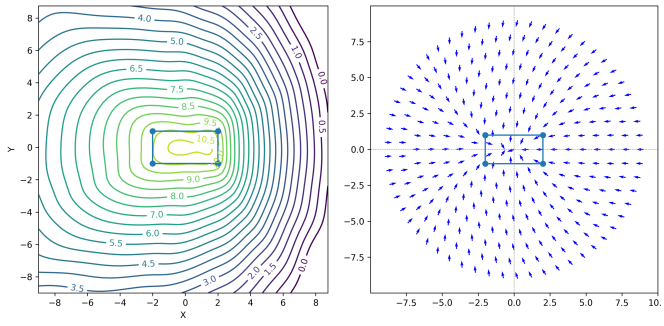


(a) $V(x)$ (b) $u^*(x)$

Figure 5: GP training results for a 4×2 rectangle moving at a speed of 0.8 along the positive x axis with 20000 epochs and maximum 100 steps for each epoch: (a) shows the contours of the learned value function and (b) shows the learned control vector field.

fields (e.g., stronger repulsive behavior on the "upstream" side of a moving element).

As shown in Fig. 5a that the value function clearly shows that the area in front of the moving element has a significantly lower cumulative cost, indicating that this area is much more dangerous (closer to the element) than the back of the element when serving as an obstacle. However, unlike the PPO critic learning results shown in Fig. 6a, the boundary where $V_j(x) = 0$ of the value function learned by our GP model extends beyond the true boundary of the element, whereas the PPO boundary adheres closely to the true boundary of the element.



(a) $V_\phi(x)$ (b) $u_\theta(x)$

Figure 6: PPO training results for a 4×2 rectangle moving at a speed of 0.8 along the positive x axis with 20000 epochs and a maximum of 100 steps for each epoch: (a) shows the contours of the learned value function and (b) shows the mean policy control vector field.

6 Modular Composition with Multiple Elements

In this section we propose a method to integrate multiple learned models into a single control system with collision-avoidance and goal-reaching behaviors via a QCQP.

Recall from Sec. 5.1, for an element $e_j \in \mathcal{E}$, the value function and the optimal policy estimation are denoted as $\hat{V}_j(x)$ and $\hat{u}_j^*(x)$. For the remainder of this section, we omit the hat from the estimates to simplify notation.

6.1 CLF-CBF-based Online Composition

We propose the use of the value function V_j as the CBF function $H(x)$ in the CBF constraint Eq. 4:

$$\dot{V}_j + c_j V_j = \nabla_x V_j^\intercal \dot{x} + c_j V_j \geq 0 \tag{26}$$

In practice, it is useful to consider a variation of the constraint above that uses a *sharpened* version ${V_j}^q$ of the CBF (to avoid problems where ∇V_j vanishes near $V_j = 0$, $0 < q < 1$) and a $V_{\min}$ level set (i.e., $V^q \geq V_{\min}$ to add a safety buffer around the obstacle), yielding

$$\dot{{V_j}^q} + c_j({V_j}^q - V_{\min}) = q{V_j}^{(q-1)} \nabla_x V_j^\intercal \dot{x} + c_j(V_j^q - V_{\min}) \geq 0; \tag{27}$$

Note that $\dot{V}_j = \nabla_x V_j^\intercal \dot{x}$ is a function affine in action u; in our setting, this expression cannot be evaluated directly, but can be obtained from our actor-critic model. Specifically, we obtain $\nabla_x V_j^\intercal \dot{x}$ from Eq. 20b and substitute the approximation $\hat{A}_{critic} = \hat{A}_{actor}$ to obtain:

$$\begin{aligned}
\nabla_x V(x)^\intercal \dot{x} =& \hat{A}_{critic}(x,u) - \frac{1}{2} u^\intercal u + \lambda V(x) \\
=& \frac{1}{2}\big(u - u^*(x)\big)^\intercal \big(u - u^*(x)\big) - \frac{1}{2} u^\intercal u + \lambda V(x) \\
=& \frac{1}{2}\|u^*(x)\|^2 - u^*(x)^\intercal u + \lambda V(x);
\end{aligned} \tag{28}$$

As expected, this expression is affine in u.

At runtime, we evaluate each element-specific model in the current relative state to obtain $V_j(x_j), u_j^*(x_j)$. The goal is to choose action u such that Eq. 4 is always satisfied, thus guaranteeing safety (i.e., $V_j \geq 0$). We then solve the following small QCQP that plays the same *architectural role* as CLF-CBF QPs: it selects a control input that aims for goal-seeking behavior while enforcing per-obstacle safety constraints, and a maximum speed u_{max}:

$$u^{\text{QCQP}} \arg\min_u \ \|u - u_g\|^2 \tag{29a}$$

$$\text{subject to } q{V_j}^{(q-1)}\ {u_j^*}^\intercal\, u \leq c_j(V_j^q - V_{\min}) - q{V_j}^{(q-1)} \left(\frac{1}{2}\|u_j^*\|^2 + \lambda V(x)\right)$$

$$j \in \{1, \ldots, M\}, \tag{29b}$$

$$\|u\| \leq u_{\max}. \tag{29c}$$

Intuitively, each obstacle limits the component of the action along the learned approach-to-contact direction, thereby preventing the controller from choosing actions that would aggressively decrease the corresponding V_j.

Remark 2 (Relationship with CLF-CBF QPs). Classical CLF-CBF QPs enforce constraints on Lie derivatives (and therefore require the drift and control vector fields $f(x), g(x)$) to guarantee stability/safety properties. In contrast, Eq. 29 is *model-free*: it uses only learned value functions and direction fields, yet it retains the key scalability property of optimization-based safety filters—adding an obstacle corresponds to adding one inequality constraint.

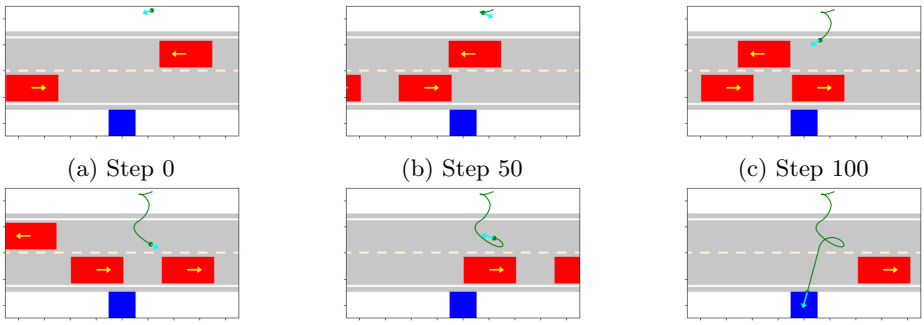
6.2 Simulation Results after Composing Multiple Moving Obstacles and a Static Goal

In this section, we present the training results in 2-D simulations to show the training results and collision avoidance via real-time composition.

Simulation Setup. Finally, to demonstrate compositionality, we created a library of actor-critic pairs trained on *isolated* elements (steady and moving). For each element configuration (shape and motion profile), we train an element-specific primitive consisting of a value function $V_j(x_j)$ and an optimal policy $u_j^*(x_j)$, using the same actor–critic procedure for both goals and obstacles.

We then deploy them in a *previously unseen* multi-element scenario: a road crossing setting with multiple moving vehicles and a goal region. It is a simulation of a street-crossing scenario to demonstrate the capabilities of our system. The red rectangles represent moving cars. The blue square represents the goal area. The green dot represents the agent.

The controller is synthesized online by composition and without any additional training, and the goal and obstacle are combined online by the QCQP controller of Sec. 6, which mediates goal-seeking and collision avoidance in real time.



(a) Step 0 (b) Step 50 (c) Step 100

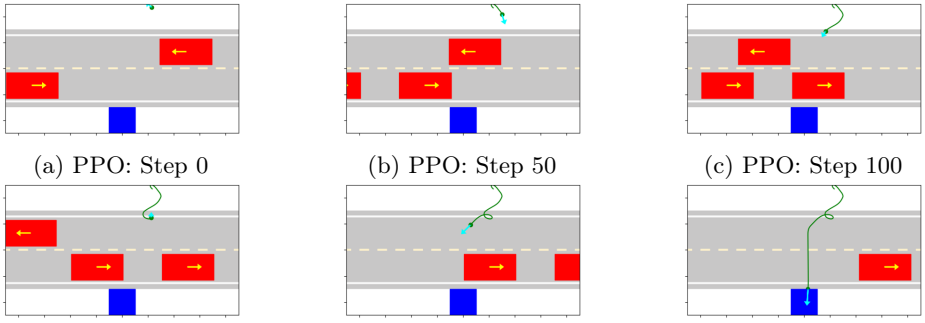
(d) Step 150 (e) Step 200 (f) Step 251

Figure 7: GP simulation results for a street crossing scenario with two-way traffic.

Simulation results. The simulation results obtained by composing the models trained via the GP are shown in Fig. 7

At the beginning of the simulation, the agent only sensed the car approaching from the right. However, if the agent attempts to overtake the car and cross the street, it deviates substantially from the target direction. Therefore, it stopped at the position closest to the goal, waited for the car to pass, and continued crossing the street. The agent performs a waiting-and-pass cycle for cars traveling in the opposite directions. After all cars have passed, the agent adheres to the goal-reaching controls without interference.

We can apply the same composition strategies to the training obtained via PPO. The results are shown in Fig. 8 where the PPO-trained functions are tested in the same scenario as shown in Fig. 7. Although the actions selected by the two systems are not identical, both methods exhibit a similar high-level behavior, alternating between periods of dwelling in place and moving toward the goal.



(a) PPO: Step 0 (b) PPO: Step 50 (c) PPO: Step 100

(d) PPO: Step 150 (e) PPO: Step 200 (f) PPO: Step 252

Figure 8: PPO simulation results for a street crossing scenario with two-way traffic.

7 Discussion

As shown in the simulations, after training, our method and PPO recovered qualitatively similar structures in both the value function $V(x)$ and the induced optimal control field $u^*(x)$. This suggests that in our setting, our method and PPO can achieve similar training outcomes.

We draw two conclusions: *1)* First, although our framework is derived from a continuous-time HJB formulation, PPO remains a useful baseline because it can be trained on a discrete-time approximation of the same environment and objective, and its learned policy can be executed in the same online QP-based controller used during deployment. However, the PPO does not directly optimize our continuous-time HJB residual; instead, it optimizes a discrete-time surrogate

objective based on trajectory rollouts and advantage estimates. Bridging these views (e.g., developing policy-optimization updates that are native to continuous time) is an interesting direction for future work. *2)* Second, our modular and composable architecture is not tied to a specific learner: any method that provides a critic estimating a value function and an actor providing a state-dependent policy can be integrated into the same online QP to balance goal-reaching and safety in environments more complex than those encountered during training.

8 Conclusion and Future Work

We introduced a composable, model-free reinforcement learning framework for collision-avoidance navigation in continuous time for input-affine systems. The key idea is to train an *independent* actor-critic (a value function and corresponding policy field for each goal or obstacle element) using a continuous-time discounted HJB residual that only requires state trajectories, time derivatives (estimated from data), and termination events. At deployment time, these independently trained actor-critic pairs are combined online through linear constraints in a QCQP that plays the same architectural role as CLF-CBF QPs, seeking optimal goal-seeking behavior while enforcing per-obstacle safety constraints, and they scale gracefully by adding one constraint per obstacle.

Using Gaussian Processes as the framework backbone of our general function-approximation template, we demonstrated that the learned results recover interpretable value functions and optimal control fields for both stationary and moving elements, capturing the hidden drift factor $f(x)$ introduced by element motion even when the learner observes only the relative position. We further showed that composing results trained in isolation enables real-time navigation in previously unseen multi-element scenarios such as a crosswalk with two-way traffic. Finally, we compared against PPO trained on a discrete-time approximation of the same environment and found qualitatively similar learned structures and comparable collision-avoidance behavior under the same QCQP-based deployment controller.

Several directions remain for future research. On the learning side, it is important to study the stability and safety properties of the composed controller and to characterize the effect of approximation errors in learned results on closed-loop behavior. On the modeling side, scaling beyond low-dimensional relative states requires sparse or structured approximations and careful treatment of partial observability when the hidden dynamics are significant. Finally, a major advantage of Gaussian Processes is their ability to quantify uncertainty; we plan to treat the HJB residual as a measurement and propagate uncertainty over the stored base-point parameters via Kalman-filter-style covariance updates. Such calibrated uncertainty estimates can be integrated into the online QCQP (e.g., via risk-sensitive or chance-constrained formulations) to improve robustness in dynamic and uncertain environments.

Bibliography

[1] Ames, A.D., Xu, X., Grizzle, J.W., Tabuada, P.: Control barrier function based quadratic programs for safety critical systems. IEEE Transactions on Automatic Control **62**(8), 3861–3876 (2016)

[2] Baird, L.C.: Advantage updating. Tech. rep., Technical Report WL-TR-93-1146, Wright-Patterson Air Force Base Ohio: Wright ... (1993)

[3] Khalil, H.K., Grizzle, J.W.: Nonlinear systems, vol. 3. Prentice hall Upper Saddle River, NJ (2002)

[4] Kirk, D.E.: Optimal control theory: an introduction. Courier Corporation (2004)

[5] Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. The International Journal of Robotics Research **32**(11), 1238–1274 (2013)

[6] Konda, V., Tsitsiklis, J.: Actor-critic algorithms. Advances in neural information processing systems **12** (1999)

[7] Nguyen, Q., Hereid, A., Grizzle, J.W., Ames, A.D., Sreenath, K.: 3d dynamic walking on stepping stones with control barrier functions. In: 2016 IEEE 55th Conference on Decision and Control (CDC). pp. 827–834. IEEE (2016)

[8] Park, J.J., Kim, J.H., Song, J.B.: Path planning for a robot manipulator based on probabilistic roadmap and reinforcement learning. International Journal of Control, Automation, and Systems **5**(6), 674–680 (2007)

[9] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-baselines3: Reliable reinforcement learning implementations. Journal of machine learning research **22**(268), 1–8 (2021)

[10] Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. The MIT Press (2006)

[11] Rimon, E.: Exact robot navigation using artificial potential functions. Yale University (1990)

[12] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)

[13] Sutton, R.S., Barto, A.G., et al.: Reinforcement learning: An introduction, vol. 1. MIT press Cambridge (1998)

[14] Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. Advances in neural information processing systems **12** (1999)

[15] Watkins, C.J.C.H., et al.: Learning from delayed rewards (1989)

[16] Yang, G., Vang, B., Serlin, Z., Belta, C., Tron, R.: Sampling-based motion planning via control barrier functions. In: Proceedings of the 2019 3rd International Conference on Automation, Control and Robots. pp. 22–29 (2019)

[17] Zhang, B., Mao, Z., Liu, W., Liu, J.: Geometric reinforcement learning for path planning of uavs. Journal of Intelligent & Robotic Systems **77**, 391–409 (2015)