2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Generating Stable and Collision-Free Policies through Lyapunov Function Learning

Alexandre Coulombe[1] and Hsiu-Chin Lin[2]

***Abstract*— The need for rapid and reliable robot deployment is on the rise. Imitation Learning (IL) has become popular for producing motion planning policies from a set of demonstrations. However, many methods in IL are not guaranteed to produce stable policies. The generated policy may not converge to the robot target, reducing reliability, and may collide with its environment, reducing the safety of the system. Stable Estimator of Dynamic Systems (SEDS) produces stable policies by constraining the Lyapunov stability criteria during learning, but the Lyapunov candidate function had to be manually selected. In this work, we propose a novel method for learning a Lyapunov function and a collision-free policy using a single neural network model. The method can be equipped with an obstacle avoidance module for convex object pairs to guarantee no collisions. We demonstrated our method is capable of finding policies in several simulation environments and transfer to a real-world scenario.**

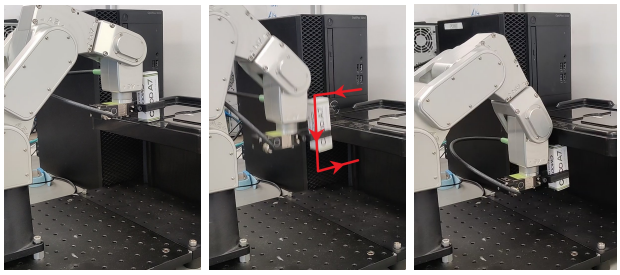
***Index Terms*— Imitation Learning, Lyapunov stability, Obstacle Avoidance, Motion Planning, Neural networks**

I. INTRODUCTION

With the growing number of robots in industry and in human environments, the need for ease of deployment and safety in robotics becomes increasingly apparent. Robots are skilled in routine and repetitive tasks. However, it is hard to handle a new task without tedious modelling, designing, and programming. Furthermore, safety is crucial to deploying robotic systems since a mistake may injure the human or damage the robot itself. This is particularly important when robots need to maneuver in environments with obstacles, such as the example in Figure 1.

Trajectory optimization is a well-studied method for motion planning. The method finds a trajectory by minimizing some objective functions while satisfying its physical limits [1], [2]. In the presence of obstacles, collision avoidance can be achieved by imposing a minimum distance between the robot and the obstacles as inequality constraints [3], [4]. While trajectory optimization is popular for motion planning, the solution is only valid for a specific pair of initial and target states. Re-planning is a time-consuming process; therefore, it is not suitable for real-time planning. Additionally, the planned path is not immune to perturbations.

In recent years, Imitation Learning (IL), the data-driven approach for motion planning, has shown promising results.



(a) Initial Position (b) Policy Rollout (c) Goal Position

Fig. 1: Mecademic Meca500 performing the shelf manipulation task: the robot (a) starts from an initial position, (b) follows the policy rollout, and (c) reaches the goal position.

IL learns a model to predict the desired action given the current input, and therefore, produces a global solution as opposed to trajectory optimization. Many machine learning approaches have been applied, including supervised learning [5], [6], Inverse Reinforcement Learning (IRL) [7], and Generative Adversarial Imitation Learning (GAIL) [8], [9]. However, these methods have no guarantees that the policy will be stable when deployed.

Stability is crucial for robotic systems. Prior work proposed the Stable Estimator of Dynamic Systems (SEDS), a data-driven approach with stability constraints to generate stable policies [10], [11]. However, [10], [11] relies on a chosen Lyapunov candidate function to be valid for the demonstrations. Another prior work constrains the Lyapunov stability conditions on neural networks and learns the Lyapunov function with the policy from training data using mixed-integer programming [12]. On the Reinforcement Learning (RL) front, prior work introduces the stability constraint during training by widening the region of attraction of a Lyapunov function [13]. However, the challenge is finding a suitable Lyapunov function.

In the presence of obstacles, methods such as virtual potential fields can modulate the policy to repel the robot from obstacles [14]. However, these create local minima where the robot is stuck. Recent work avoids extrema by modulating space around obstacles so that the robot moves around them, but it only works for point representations of the robot [15]–[17].

In this paper, we propose a novel method for learning both the Lyapunov function and a collision-free policy *together* with a single neural network. To produce a stable policy, we train the network using a constrained optimization formulation with stability conditions as constraints. We deploy the

[1]Alexandre Coulombe is with the Department of Electrical and Computer Engineering, McGill University, Canada alexandre.coulombe@mail.mcgill.ca

[2]Hsiu-Chin Lin is with the School of Computer Science and the Department of Electrical and Computer Engineering, McGill University, Canada hsiu-chin.lin@cs.mcgill.ca

This work is sponsored by Mitacs Accelerate IT23788.

2023 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3503-2365-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICRA48891.2023.10160494

trained policy with an obstacle avoidance module, augmented by our method to deal with convex objects. The proposed work is validated in simulation and robotic hardware with direct sim-to-real transfer. The main contributions include:

- We extend [10] by learning a Lyapunov function from demonstrations instead of manually choosing a Lyapunov candidate function (Sec. III-B).
- We extend the obstacle avoidance work in [15]–[17] to treat robots and obstacles as *convex objects* (Sec. III-C).
- We show that policies developed in simulation can be transferred to a real-world system reliably (Sec. IV).

II. BACKGROUND

In this section, background knowledge for Lyapunov stability theory and collision avoidance techniques are provided.

A. Lyapunov Function

From [18], Lyapunov stability dictates that a system with d degree-of-freedom is locally stable if, for a region around the equilibrium $\mathbf{x}_g \in \mathbb{R}^d$, there is a Lyapunov function $V(\cdot): \mathbb{R}^d \to \mathbb{R}$ that satisfies the conditions:

$$V(\mathbf{x}[n]) > 0, \qquad \forall \mathbf{x}[n] \in X, \mathbf{x}[n] \neq \mathbf{x}_g \quad (1)$$

$$V(\mathbf{x}[n+1]) \leq (1-\varepsilon)V(\mathbf{x}[n]), \qquad \forall \mathbf{x}[n] \in X, \mathbf{x}[n] \neq \mathbf{x}_g \quad (2)$$

$$V(\mathbf{x}_g) = 0 \quad (3)$$

where $\varepsilon > 0$ is a positive scalar and $X = \{\mathbf{x}[0] \mid V(\mathbf{x}[0]) \leq \rho\}$ is the stable region around $\mathbf{x}_g$, where $\rho > 0$ is some positive value. (1) and (3) specify that $V(\cdot)$ must output a positive value for all positions, except for 0 at $\mathbf{x}_g$. (2) states that the system must evolve to positions with lower values, with the lowest value at $\mathbf{x}_g$. Thus, the system will converge to the equilibrium point $\mathbf{x}_g$ if a Lyapunov function can be found.

A common Lyapunov candidate function is the quadratic Lyapunov function

$$V(\mathbf{x}[n]) = \|\mathbf{x}[n] - \mathbf{x}_g\|^2 \quad (4)$$

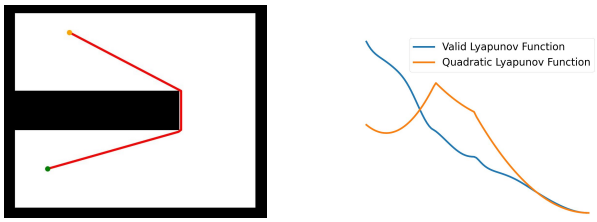
The above Lyapunov candidate function imposes constraints on the movement towards the goal. However, this may not be ideal in some scenarios, such as moving around obstacles. An example is shown in Figure 2. The quadratic Lyapunov candidate function (orange) violates the constraint in (2), since avoiding the obstacles implies moving away from the goal, while a valid Lyapunov function (blue) is monotonically decreasing along the demonstration path (red).

Instead of explicitly specifying the Lyapunov candidate function, prior work [12] learns a function that satisfies the Lyapunov conditions in (1)-(3). In this work, we will adapt this structure to learn the Lyapunov function.

B. Collision Avoidance

Given the distance between the robot and the obstacle, a policy $\pi(\mathbf{x}[n])$ can be *modulated* to move around the obstacles [15]–[17] with the following formulation:

$$\dot{\mathbf{x}}[n] = \mathbf{M}(\mathbf{x}[n])\,\pi(\mathbf{x}[n]) \quad (5)$$



(a) A demonstration (red) with an initial (orange) and a goal (green) states while avoiding obstacles (black).

(b) Outputs of a quadratic Lyapunov function (orange) and a valid Lyapunov function (blue) for the demonstrations.

Fig. 2: An example of a trajectory and its Lyapunov function outputs.

where $\mathbf{M}(\mathbf{x}[n])$ is the *modulation matrix* of the obstacle. The modulation matrix conserves the existing extrema of the system dynamics and does not introduce new extrema while the matrix is full rank [16]. The modulation matrix $\mathbf{M}(\mathbf{x}[n])$ is formed by the basis matrix $\mathbf{E}(\mathbf{x}[n])$ and eigenvalue diagonal matrix $\mathbf{D}(\mathbf{x}[n])$

$$\begin{aligned}\mathbf{M}(\mathbf{x}[n]) &= \mathbf{E}(\mathbf{x}[n])\,\mathbf{D}(\mathbf{x}[n])\,\mathbf{E}(\mathbf{x}[n])^{-1} \\ \mathbf{E}(\mathbf{x}[n]) &= [r(\mathbf{x}[n]), e_1(\mathbf{x}[n]), \ldots, e_{d-1}(\mathbf{x}[n])] \\ \mathbf{D}(\mathbf{x}[n]) &= \mathbf{diag}(\lambda_r(\mathbf{x}[n]), \lambda_e(\mathbf{x}[n]), \ldots, \lambda_e(\mathbf{x}[n]))\end{aligned} \quad (6)$$

where $r(\mathbf{x}[n])$ is a vector from the obstacle reference point to the robot reference point, and $e_i(\mathbf{x}[n])$ are vectors tangent to the obstacle surface. $\lambda_r(\mathbf{x}[n]), \lambda_e(\mathbf{x}[n])$ are related to the distance between the robot and the obstacle $\Gamma(\mathbf{x}[n])$ as:

$$\lambda_r(\mathbf{x}[n]) = 1 - \frac{1}{\Gamma(\mathbf{x}[n])}, \quad \lambda_e(\mathbf{x}[n]) = 1 + \frac{1}{\Gamma(\mathbf{x}[n])} \quad (7)$$

This method prevents motion from penetrating the surface of the obstacles while encouraging the movement in the tangential directions of the obstacles. In this work, we extend this method by treating convex obstacles with a convex hull representation of the robot.

C. Distance Measurement for Convex Objects

The collision avoidance technique in the previous section requires the signed distance between the robot and an obstacle. Assuming the robot and the obstacle can be described as a convex object, the distance between two convex objects $\mathbf{A}$ and $\mathbf{B}$ can be calculated with the Minkowski difference, $\mathbf{A} \ominus \mathbf{B} = \{\mathbf{a} - \mathbf{b} : \forall \mathbf{a} \in \mathbf{A}, \forall \mathbf{b} \in \mathbf{B}\}$.

To find the shortest distance between $\mathbf{A}$ and $\mathbf{B}$ without computing the entire Minkowski difference, Recursive Gilbert Johnson Keerthi (RGJK) [19] and Expanding Polytope Algorithm (EPA) [20] exploit its properties to obtain the shortest separating and penetrating distance respectively. Together, they give the signed distance $sd(\mathbf{A},\mathbf{B})$ between a pair of convex hulls as follows:

$$sd(\mathbf{A},\mathbf{B}) = RGJK(\mathbf{A},\mathbf{B}) - EPA(\mathbf{A},\mathbf{B}) \quad (8)$$

Many extensions were introduced to speed up this computation [21], [22].

III. PROPOSED APPROACH

We consider a task with d degree-of-freedom where $\mathbf{x}[\cdot] \in \mathbb{R}^d$ represents the state, and $\dot{\mathbf{x}}[\cdot] \in \mathbb{R}^d$ represents the velocities. Assuming we have full knowledge of the environment, our goal is to learn a policy $\dot{\mathbf{x}}[\cdot] = \pi(\mathbf{x}[\cdot]) : \mathbb{R}^d \to \mathbb{R}^d$ that outputs the velocities which guides the robot towards a target state $\mathbf{x}_g \in \mathbb{R}^d$ while enforcing stability on the motion.

A. Automated Demonstration Collection

The data is automatically generated via trajectory optimization. Given an initial position $\mathbf{x}_0$ and a target position $\mathbf{x}_g$, the objective of the optimization is to find a trajectory of positions $\mathbf{x}[\cdot]$ and a trajectory of velocities $\dot{\mathbf{x}}[\cdot]$, that travel from $\mathbf{x}_0$ and $\mathbf{x}_g$ with the shortest distance. In addition, the solution should be constrained by the position limits $\mathbf{x}^-, \mathbf{x}^+$ and the velocity limits $\dot{\mathbf{x}}^-, \dot{\mathbf{x}}^+$.

Trajectory optimization allows us to introduce additional constraints such as avoiding collisions with the environment. Assuming that the robot has N_{link} links and performs some tasks in an environment of $N_{obstacles}$ obstacles, we need to avoid collisions between each link of the robot and the obstacles. For collision avoidance, we model each object as a convex hull, since they are less prone to overestimating the volume of the object. For non-convex objects, the object can be represented as a collection of its convex components. The trajectory optimization is formulated as follows:

$$\begin{aligned} \min_{\mathbf{x}[\cdot],\dot{\mathbf{x}}[\cdot]} \quad & \sum_{n=0}^{N-1} \|\mathbf{x}[n+1] - \mathbf{x}[n]\|^2 \\ \text{subject to } & \mathbf{x}[n+1] = \mathbf{f}(\mathbf{x}[n], \dot{\mathbf{x}}[n]), \forall n \in [0, N-1] \\ & \mathbf{x}[0] = \mathbf{x}_0, \quad \mathbf{x}[N] = \mathbf{x}_g \\ & \mathbf{x}^- \leq \mathbf{x}[n] \leq \mathbf{x}^+ \\ & \dot{\mathbf{x}}^- \leq \dot{\mathbf{x}}[n] \leq \dot{\mathbf{x}}^+ \\ & sd(A_i, O_j) \geq d_{safe}, \forall i \in [1, N_{link}] \\ & \qquad \forall j \in [1, N_{obstacles}] \end{aligned} \tag{9}$$

where N is the number of collocation points, $\mathbf{f}$ is the forward dynamics of the robot, sd is the signed distance for convex hulls in (8), A_i is the convex hull of the i^{th} link of the robot, O_i is the convex hull of the i^{th} obstacle, and $d_{safe} > 0$ is the minimum distance that must not be violated for obstacles.

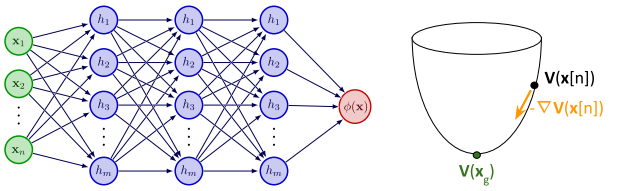
With this information, the formulation in (9) can be passed to an interior point optimization solver, such as IPOPT [23]. This can be run multiple times from different initial positions to produce a set of demonstrations.

B. Value Function and Policy Learning

We assume that data are generated using Sec. III-A as a set of positions $\mathbf{x}[\cdot]$ and velocities $\dot{\mathbf{x}}[\cdot]$. Our goal is to learn a policy π that predicts the most suitable velocity given the current state $\dot{\mathbf{x}}[\cdot] = \pi(\mathbf{x}[\cdot])$ and satisfies the Lyapunov stability conditions. The proposed supervised learning method aims to shape a Lyapunov function $\tilde{V}(\cdot) : \mathbb{R}^d \to \mathbb{R}$.

We follow the formulation in [12] to structure a Lyapunov candidate function as follows:

$$\tilde{V}(\mathbf{x}[n]) = \phi(\mathbf{x}[n] - \mathbf{x}_g, \theta_V) - \phi(\mathbf{0}, \theta_V) + \|\mathbf{x}[n] - \mathbf{x}_g\| \tag{10}$$



(a) ANN for the Lyapunov function (b) Policy

Fig. 3: Learning Lyapunov policy: (a) an ANN is learned (b) the action is the negative gradient of the Lyapunov function

where $\phi(\cdot, \theta_V) : \mathbb{R}^d \to \mathbb{R}$ is an Artificial Neural Network (ANN) parameterized by θ_V. This formulation imposes that the value at the target state is 0, which satisfies (3).

The policy can be taken to be related to the negative gradient of the learned Lyapunov function

$$\pi(\mathbf{x}[n]) \propto -\nabla \tilde{V}(\mathbf{x}[n]) \tag{11}$$

The idea is illustrated in Fig. 3. The Lyapunov function $\tilde{V}(\mathbf{x}[n])$ is nonlinear with respect to positions $\mathbf{x}[n]$, so we deploy an ANN to learn the relationship (Fig. 3a). Once a Lyapunov function is learned, the negative gradient $-\nabla \tilde{V}(\mathbf{x}[n])$ will guide the robot toward the target (Fig. 3b).

To train the neural network, the objective is to minimize the discrepancy between the policy output $-\nabla \tilde{V}(\mathbf{x}[n])$ and the demonstrated action $\dot{\mathbf{x}}[n]$. This discrepancy $e(n)$ can be taken from the dot product between two vectors.

$$-\nabla \tilde{V}(\mathbf{x}[n]) \cdot \dot{\mathbf{x}}[n] = \|\nabla \tilde{V}(\mathbf{x}[n])\| \|\dot{\mathbf{x}}[n]\| \cos(e(n))$$

Dividing both sides by the magnitude of the two vectors and find the inverse cosine, the objective function becomes

$$\min_{\theta_V} \sum_{n=0}^{N-1} \left\| \arccos\left(\frac{-\nabla \tilde{V}(\mathbf{x}[n]) \cdot \dot{\mathbf{x}}[n]}{\|\nabla \tilde{V}(\mathbf{x}[n])\| \|\dot{\mathbf{x}}[n]\|} \right) \right\| \tag{12}$$

In order to generate a valid Lyapunov function, the stability conditions in (1) and (2) are added to the objective in (12). The problem is converted into:

$$\begin{aligned} \min_{\theta_V} \quad & \sum_{n=0}^{N-1} \left\| \arccos\left(\frac{-\nabla \tilde{V}(\mathbf{x}[n]) \cdot \dot{\mathbf{x}}[n]}{\|\nabla \tilde{V}(\mathbf{x}[n])\| \|\dot{\mathbf{x}}[n]\|} \right) \right\| \\ \text{subject to } & \tilde{V}(\mathbf{x}[n]) > 0, \forall \mathbf{x}[n] \neq \mathbf{x}_g, \\ & \tilde{V}(\mathbf{x}[n+1]) - (1-\varepsilon)\tilde{V}(\mathbf{x}[n]) \leq 0 \end{aligned} \tag{13}$$

To make the optimization easier for implementation in tools such as PyTorch [24], the formulation is modified to be an augmented Lagrangian. Also, the trigonometry is simplified to be a difference by rearranging the terms of the dot product and inserting the desired angle. These changes produce the following optimization:

$$\begin{aligned} \min_{\theta_V} \quad & \sum_{n=0}^{N-1} \left(1 + \frac{\nabla \tilde{V}(\mathbf{x}[n]) \cdot \dot{\mathbf{x}}[n]}{\|\nabla \tilde{V}(\mathbf{x}[n])\| \|\dot{\mathbf{x}}[n]\|} \right) \\ & + \lambda_1 \max\left(-\tilde{V}(\mathbf{x}[n]), 0 \right) \\ & + \lambda_2 \max\left(\tilde{V}(\mathbf{x}[n+1]) - (1-\varepsilon)\tilde{V}(\mathbf{x}[n]), 0 \right) \end{aligned} \tag{14}$$

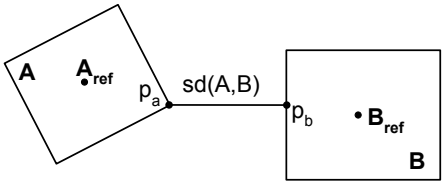


Fig. 4: The reference points on the convex hulls used to get the distance function output.

where λ_1 and λ_2 are the Lagrange multipliers. The model will be a valid Lyapunov function for the region around the provided demonstrations. Outside the demonstration regions, there is no guarantee on the stability of the model. Termination before the constraints are satisfied will result in an invalid Lyapunov function. Finally, to ensure that the policy output is within the physical limits, the output is scaled by

$$\pi(\mathbf{x}[n]) = \frac{-\nabla\tilde{V}(\mathbf{x}[n])}{\|\nabla\tilde{V}(\mathbf{x}[n])\|}\dot{\mathbf{x}}_{max} \tag{15}$$

C. Obstacle Avoidance for Convex object pairs

The learning approach introduced in the previous section learns a policy that reproduces the demonstrated data with stability guarantee. However, it has no information about obstacles in the environment. To further guarantee a collision-free policy, the obstacle avoidance method from Section II-B was improved upon. While [17] treats the robot as a *point*, we augmented the method for handling *convex representations*. We represent each link of the robot with the convex hull of the link, and the obstacles are convex objects.

For convex to convex comparison, the signed distance between the convex objects is required. An example is shown in Fig. 4. Assuming $\mathbf{A}$ and $\mathbf{B}$ are two convex objects, $\mathbf{A}_{ref}$ and $\mathbf{B}_{ref}$ are the reference position of object $\mathbf{A}$ and $\mathbf{B}$ respectively, p_a is the closest point on $\mathbf{A}$ to $\mathbf{B}$, and p_b is the closest point on $\mathbf{B}$ to $\mathbf{A}$. The signed distance $sd(\mathbf{A},\mathbf{B})$ between $\mathbf{A}$ and $\mathbf{B}$ can be computed from (8).

For handling convex objects, we define the distance function for the modulation matrix between convex objects as

$$\Gamma(\mathbf{x}[n]) = \frac{\|\mathbf{A}_{ref} - \mathbf{B}_{ref}\|}{\|\mathbf{A}_{ref} - \mathbf{B}_{ref}\| - sd(\mathbf{A},\mathbf{B})} \tag{16}$$

In (16), the signed distance $sd(\mathbf{A},\mathbf{B})$ is used to form a ratio. When $sd(\mathbf{A},\mathbf{B}) = 0$, the two objects collide. The result is $\Gamma(\mathbf{x}[n]) = 1$, so that $\lambda_r(\mathbf{x}[n]) = 0$ in (7) and makes (5) block motion towards the obstacle. When $sd(\mathbf{A},\mathbf{B}) \geq 0$, $\Gamma(\mathbf{x}[n]) \geq 1$, which gives the desired eigenvalues for the obstacle modulation when they are not intersecting.

For the basis vectors in (6), $r(\mathbf{x}[n]) = p_a - p_b/\|p_a - p_b\|$ is taken to be the normal direction in which to impede robot motion and $e_i(\mathbf{x}[n])$ are chosen to be orthogonal to the normal. In 2D, this can simply be the normal rotated by 90°. In 3D, this can be done using spherical coordinates and taking $\hat{r} = r(\mathbf{x}[n])$, and $\hat{\theta}$ and $\hat{\phi}$ to be the tangents.

A summary of our proposed method is illustrated in Fig. 5.

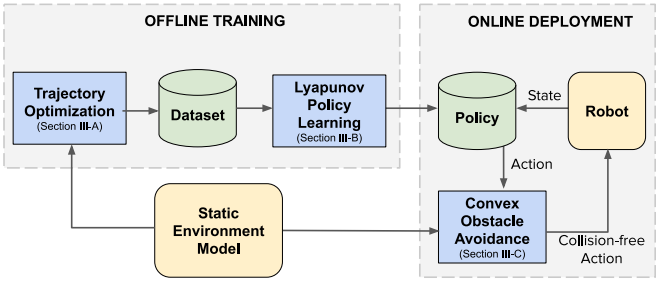


Fig. 5: A summary of our proposed method

IV. EXPERIMENTS

We evaluate the method on several scenarios, including (1) 2D examples, (2) a manipulation task in simulation, and (3) the same manipulation task on a real robot. For each case, a set of demonstrations are collected through the trajectory optimization method from Section III-A. We used the obstacle avoidance module in all experiments to show the difference on learned policies.

A. Evaluation Criteria

We use SEDS [10] and GAIL [8] as our baselines. The performance of a method is evaluated based on its convergence to the goal, its ability to avoid collisions, and its ability to satisfy the Lyapunov stability conditions.

B. Experiment 1: 2D examples

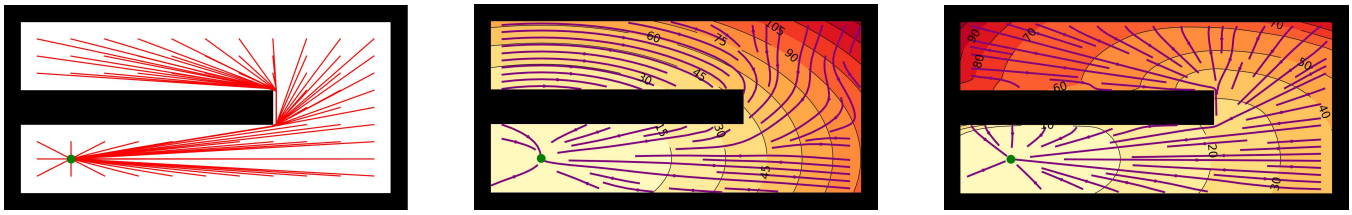
To demonstrate the necessity of learning a Lyapunov candidate function, the method was first validated on 2D examples where the issues can be directly visualized. We designed two scenarios: (a) the hallway (Fig. 6) and (b) the cross-shaped (Fig. 7). The target position is marked as the green dot.

We used the automatic data collection method in Section III-A to collect 75 and 2818 demonstrations for the hallway and cross environments respectively, where the initial position is a valid position from a grid on the environment around the target position. The collected demonstrations can be viewed as the red arrows in Fig. 6a and in Fig. 7a.

For comparison, we trained policies with SEDS which employs a quadratic Lyapunov function. The results are shown in Fig. 6b and Fig. 7b, where the background color is the Lyapunov function values and the purple streamlines are the vector fields. We can see that the top left corner fails in Fig. 6b. A similar phenomenon can be observed at the top right quadrant of Fig. 7b. This is due to the demonstrations moving away from the target, which conflicts with the quadratic Lyapunov conditions.

In contrast, our proposed method finds a solution that brings the system to the target position for both cases (Fig. 6c and Fig. 7c). In terms of the Lyapunov conditions, we see clearly that the policy motion field is in the direction of decreasing values, thus satisfying the criteria for stability.

In terms of collision avoidance, due to the Lyapunov function not allowing the baseline to move around the obstacles without violating the stability conditions, SEDS has regions where the policy would generate motions that

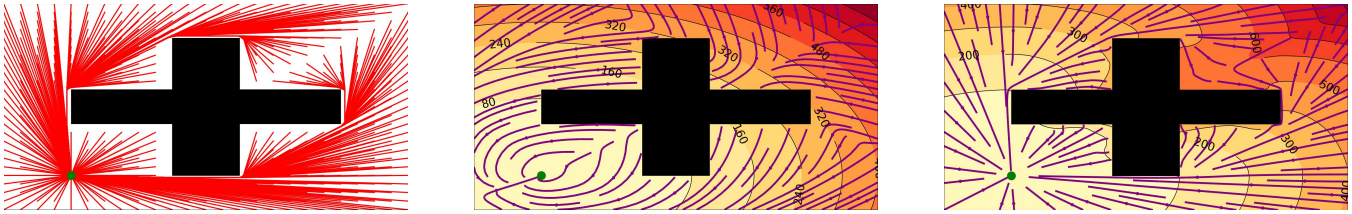


(a) Demonstrations (red arrows) with the goal (green dot).

(b) Values and vector field from SEDS with obstacle avoidance.

(c) Values and vector field from our method with obstacle avoidance.

Fig. 6: 2D hallway environment. The red arrows denote the velocities. The background colour gradient and contour lines show the Lyapunov values for the given position, and the purple streamlines show the motion field of the policy.



(a) Demonstrations (red arrows) with the goal (green dot).

(b) Values and vector field from SEDS with obstacle avoidance

(c) Values and Vector field from our Method with obstacle avoidance

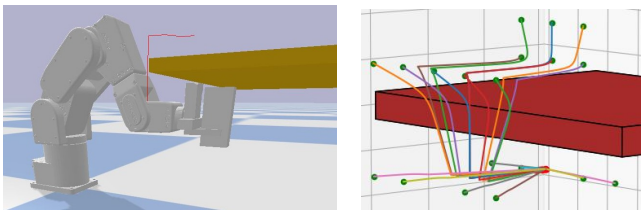
Fig. 7: 2D cross environment. The red arrows denote the velocities. The background colour gradient and contour lines show the Lyapunov values for the given position, and the purple streamlines show the motion field of the policy.

collide with the obstacles in both environments. However, our method finds a Lyapunov function that allows the policy to avoid collisions while keeping the stability conditions.

A similar conclusion can be drawn from Table I where our method has a lower MSE than the baseline.

	SEDS	Our Method
Hallway	2.87 ± 1.30	1.33 ± 1.24
Cross	2.59 ± 1.40	1.01 ± 1.01

TABLE I: Mean squared error (MSE) of various methods compared to the demonstration actions of each environment.



(a) Pybullet simulation

(b) Demonstrations

Fig. 8: Simulation and data for the shelf experiment. (a) Mecademic Meca500 in Pybullet simulation (b) Example data collected from trajectory optimization with multiple initial positions (green) and a target position (orange).

C. Experiment 2: Manipulation in Simulation

Next, we explore the performance of our method in a manipulation task. We use Pybullet [25] to simulate the environment and use the Mecademic Meca500 as the robot to perform the task. The task is to take an object from the top of a shelf to a target position under the shelf (see Fig. 8a).

We collected 1209 demonstrations by sampling a grid of start positions and select valid positions for trajectory optimization. Some example trajectories are shown in Fig. 8b. The network architecture used to learn the model is found in Appendix. The obstacle avoidance module was added so that the payload does not collide with the shelf.

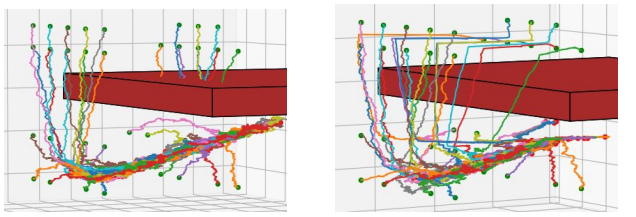
For this experiment, we compare our work with SEDS and an imitation learning method that does not have a stability guarantee. We have chosen GAIL since it is the state-of-the-art learning method. The results are summarized in Fig. 9.

In Fig 9a, we see that the GAIL policy forms spurious attractors, making it not converge to the desired state. Also, they lack knowledge of the environment, causing them to collide with obstacles. After adding our collision avoidance system to GAIL, as seen in Fig 9b, the policy is capable of avoiding the shelf. Nevertheless, the motion is not smooth since GAIL requires many demonstrations to converge.

The outcome of the SEDS baseline does not match the demonstrations and collides with the shelf (Fig. 9c). With collision avoidance, the motion looks closer to the demonstrations, but solely due to the modulation forcing the movement to avoid the obstacle (Fig. 9d).

From Fig. 9e and 9f, our method reproduces the motion with and without the obstacle avoidance module. In all cases, the policy rollouts reach the goal.

Fig. 10 is a visualization of our learnt Lyapunov function. The x-axis is the time-step n, and the y-axis is the Lyapunov function output $\tilde{V}(\mathbf{x}[n])$ normalized by the output at initial positions $\tilde{V}(\mathbf{x}[0])$. Fig. 10a and Fig. 10b are the outputs of $\tilde{V}(\mathbf{x}[n])$ with the demonstration data and with the policy rollout, respectively. We can see that the Lyapunov function



(a) GAIL without collision avoidance

(b) GAIL with collision avoidance

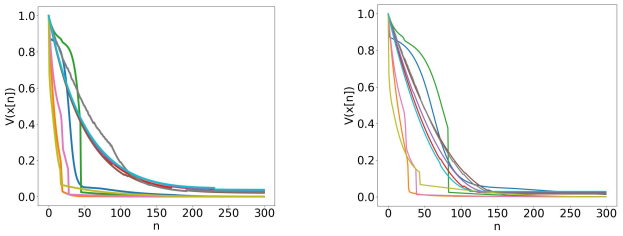
(c) SEDS without collision avoidance

(d) SEDS with collision avoidance

(e) Our method without collision avoidance

(f) Our method with collision avoidance

Fig. 9: Rollout from various initial positions (green points) to the goal position (orange points) using the learned policies



(a) Demonstrations

(b) Policy Rollout

Fig. 10: Trajectories of the learned Lyapunov function outputs $\tilde{V}(\mathbf{x}[.])$ normalized by the output at the initial positions $\tilde{V}(\mathbf{x}[0])$, where x-axis is the time-step n and y-axis is the Lyapunov function output $\tilde{V}(\mathbf{x}[n])$.

outputs are positive and monotonically decreasing until they reach 0 at the goal. This shows that our method found a suitable Lyapunov function that explains the data.

Lastly, we also look at the effect of perturbations while executing the policy (see the supplementary video. Standard imitation learning is not robust to external perturbations, especially when the robot is pushed to a state that was not previously observed in the training data. In contrast, our policy can robustly recover from external perturbations on the robot, since the policy learns a state-dependent vector field that is guaranteed to reach the goal.

D. Experiment 3: Manipulation with a real robot

We deploy the policy learnt from Experiment 2 on a robotic hardware, the Meca500 robot from Mecademic. Similar to the simulation, the task is to bring an object from anywhere in the shelf environment to a goal position below the shelf.

We start the robot from several initial positions and deploy the policy to bring the system to the goal. The policy is used as the motion planner and provides the physical robot the sequence of task-space via points toward the goal, where it is controlled through standard inverse kinematics control. Fig. 1 is an example of the outcome; where Fig. 1a shows an example of initial position, Fig. 1b is the policy rollout (red), and Fig. 1c shows that the robot reaches the target position.

The performance of the policy on the real-world robot can be viewed in the supplementary video. We see that the robot is able to perform the task without making contact with the shelf. In all cases, the policy performs the same in task-space on the real-world system as it did in simulation.

Our method can achieve sim-to-real transfer reliably. This is due to the fact that when the Lyapunov stability conditions are enforced, the movement is restricted to the directions of demonstrations and less likely to diverge. Also, our policy being kinematically planned, (i.e., mapping the positions to the velocities), which is less prone to modelling error.

V. CONCLUSION & DISCUSSION

We propose a novel method for learning a Lyapunov function and a policy using a single neural network through imitation learning. Our method is able to learn a policy that satisfies the Lyapunov stability conditions and reproduces the demonstrations. With our extension of the previous collision avoidance module, our method is capable of avoiding collisions between convex representations of the robot and environment obstacles. The policies were successfully applied to simulated environments and a real-world scenario.

In future work, the obstacle information will be incorporated into the learned neural network. This will remove the need for the obstacle avoidance module to prevent collisions and have the obstacles be learned by the network. With this improvement, the gradient of the Lyapunov function can have information for how to move away from obstacles and prevent colliding with them. This would make our method self-contained and capable of obstacle avoidance on its own.

APPENDIX

For our experiments, the activation function used for all the neurons is the tanh function. The architectures used can be found in Table A1. The information is the number of neurons in each layer: input layer, hidden layers, and output layer.

Experiment	Neural Network Architecture
Hallway	$(2, 128, 128, 128, 1)$
Cross	$(2, 128, 128, 128, 1)$
Shelf	$(3, 256, 256, 256, 256, 1)$

TABLE A1: The neural network structures used in the experiments.

REFERENCES

[1] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of guidance, control, and dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[2] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *IEEE international conference on robotics and automation*, 2011, pp. 4569–4574.

[3] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization." in *Robotics: science and systems*, vol. 9, no. 1. Citeseer, 2013, pp. 1–10.

[4] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[5] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[6] H.-C. Lin, M. Howard, and S. Vijayakumar, "A novel approach for representing and generalising periodic gaits," *Robotica*, vol. 32, no. 8, pp. 1225–1244, 2014.

[7] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, pp. 1–8.

[8] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, 2016.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, vol. 27, 2014.

[10] S. M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with gaussian mixture models," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 943–957, 2011.

[11] N. Figueroa and A. Billard, "Locally active globally stable dynamical systems: Theory, learning, and experiments," *The International Journal of Robotics Research*, 2022.

[12] H. Dai, B. Landry, L. Yang, M. Pavone, and R. Tedrake, "Lyapunov-stable neural-network control," *Robotics: Science and Systems*, 2021.

[13] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *Advances in neural information processing systems*, vol. 30, 2017.

[14] C. W. Warren, "Global path planning using artificial potential fields," in *IEEE International Conference on Robotics and Automation*, 1989, pp. 316–317.

[15] S. M. Khansari-Zadeh and A. Billard, "A dynamical system approach to realtime obstacle avoidance," *Autonomous Robots*, vol. 32, no. 4, pp. 433–454, 2012.

[16] L. Huber, A. Billard, and J.-J. Slotine, "Avoidance of convex and concave obstacles with convergence ensured through contraction," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1462–1469, 2019.

[17] L. Huber, J.-J. Slotine, and A. Billard, "Avoiding dense and dynamic obstacles in enclosed spaces: Application to moving in crowds," *IEEE Transactions on Robotics*, 2022.

[18] N. P. Bhatia and G. P. Szegö, *Stability theory of dynamical systems*. Springer Science and Business Media, 2002.

[19] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[20] G. Van Den Bergen, *Collision detection in interactive 3D environments*. CRC Press, 2003.

[21] C. J. Ong and E. G. Gilbert, "The Gilbert-Johnson-Keerthi distance algorithm: a fast version for incremental motions," vol. 2, pp. 1183–1189, 1997.

[22] A. Coulombe and H.-C. Lin, "High precision real time collision detection," *arXiv preprint arXiv:2007.12045*, 2020.

[23] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," vol. 32, pp. 8024–8035, 2019.

[25] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.