FlashPlan: A Fast Neural Motion Planner for Robotic Ball Catching

Abstract— **Time-critical robotic tasks such as ball catching require fast motion planning, as planning delays directly reduce the available control window prior to ball interception. Classical sampling-based motion planners can generate collision-free trajectories, but often exhibit high and variable planning latency, limiting their suitability for athletic robotic applications. This paper presents *FlashPlan*, a neural motion planner designed for time-sensitive robotic tasks. FlashPlan directly predicts compact, collision-free, time-normalized joint-space trajectories in a single forward pass, enabling low and consistent planning latency. To ensure dynamic feasibility, we introduce a torque-constrained time-scaling method that computes the fastest control duration for the predicted trajectory while respecting joint torque limits. Extensive benchmarking against classical planners demonstrates significantly reduced and more consistent planning latency. Experiments further show that the increased available control window translates to improved ball-catching success rates. These results highlight the importance of latency-consistent motion planning for dynamic and athletic robotic manipulation tasks.**

I. Introduction

Robotic athletic tasks such as ball catching require rapid perception, motion planning, and control under strict time constraints. Unlike quasi-static manipulation, interception tasks are inherently time-sensitive: excessive planning latency directly reduces the available control window and degrades interception performance, making fast planning a key requirement for successful ball catching. As a result, motion planners for robotic ball catching must not only generate collision-free motions but must also do so with low and consistent latency. Robotic ball interception and table tennis have long served as benchmarks for studying fast perception–planning–action loops in robotics [1]–[4]. Beyond table tennis, dynamic interception and batting systems have demonstrated the tight coupling between trajectory prediction, motion generation, and actuation timing in high-speed manipulation tasks [5]. These works highlight a fundamental principle: in dynamic interception, computational latency directly translates into reduced physical execution time.

Classical sampling-based motion planners, including rapidly-exploring random trees (RRT) [6] and probabilistic roadmaps (PRM) [7], have been widely used for collision-free motion planning in high-dimensional configuration spaces [8]–[10]. Optimization-based approaches such as CHOMP [11] and STOMP [12] further refine trajectories through gradient-based updates in configuration space. While these methods are probabilistically complete and highly flexible, their planning time is inherently instance-dependent and influenced by obstacle geometry, start–goal separation, and stochastic sampling behavior [13]. Although average planning performance can be strong, worst-case latency remains inconsistent. Such variability poses a significant challenge for time-critical interception tasks, where worst-case planning delay is often more consequential than average runtime.

Recent work has explored learning-based motion planning to accelerate planning performance. These approaches include learning sampling distributions to guide classical planners [14], predicting latent representations of feasible paths [15], and generating trajectory proposals that are subsequently refined by optimization backends [16]. While these methods substantially improve average planning efficiency, they typically retain iterative search, collision checking, or refinement procedures. Consequently, planning latency remains problem-dependent and may vary across similar task instances. For highly time-critical dynamic manipulation tasks, such variability limits practical deployment.

Alternative approaches formulate dynamic interception as a reactive control or policy learning problem, directly mapping sensory observations to joint commands using deep reinforcement learning or imitation learning [17], [18]. These methods enable fast inference but may struggle to generalize across diverse obstacle configurations in high-dimensional manipulation spaces. Moreover, purely reactive policies do not provide structured trajectory representations that can be systematically time-scaled under dynamic constraints.

In this work, we propose *FlashPlan*, a neural motion planner designed explicitly for time-critical robotic tasks. In contrast to prior learning-based planners that guide or warm-start iterative search, FlashPlan eliminates search entirely and formulates motion generation as a feedforward prediction problem. Given the current joint configuration and an environment representation, FlashPlan directly predicts a compact, collision-free, time-normalized joint-space trajectory in a single forward pass. This design enables low and consistent planning latency independent of obstacle complexity.

To support physically feasible control, we integrate a torque-constrained time-scaling formulation that computes the fastest dynamically feasible control time for the predicted motion. We validate the proposed motion planning method in extensive simulation evaluation using panning benchmarks. We also demonstrate the performance of our proposed method in a simulated

robotic ball catching setup, and improved ball catching capability is shown under strict timing constraints.

The contributions of this paper are twofold: (a) a single-pass neural motion planner that generates collision-free, time-normalized joint-space trajectories with low and consistent latency; (b) a torque-constrained time-scaling formulation that computes the fastest dynamically feasible execution time for the planned motion.

II. The System & Problem Formulation

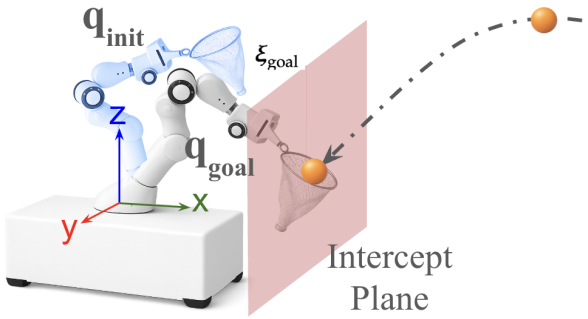

Fig. 1: Illustration of the interception pose ξ and the joint-space configurations $\mathbf{q}_{\text{init}}$ & $\mathbf{q}_{\text{goal}}$.

A. Robot Manipulator

We consider a ball-catching problem using a robot manipulator as shown in Fig. 1. The robot is *Franka Emika Panda*, a 7-DoF over-actuated robotic manipulator. The joint velocity limits are 2.62 rad/s for joints 1–4 (joint indices start from the base) and 3.14 rad/s for joints 5–7. The maximum allowable joint torques are 87 Nm for joints 1–4 and 20 Nm for joints 5–7. These actuator constraints define the feasible dynamic envelope of the system and are enforced by the low-level controller during motion execution.

B. Problem Statement

We formulate the ball-catching task as a dynamic interception problem. Assuming a perception subsystem estimates and predicts the ball trajectory, an interception point on a predefined interception plane is determined. Let $\mathbf{p}_{\text{int}} \in \mathbb{R}^3$ denote the predicted interception position of the ball on the interception plane. Let $\boldsymbol{\theta} = [\phi, \theta, \psi]^\top$ denote the desired end-effector (ball catcher) orientation expressed in roll, pitch, and yaw angles inferred from the incoming ball's velocity direction at $\mathbf{p}_{\text{int}}$. The catcher plane is aligned such that its surface normal coincides with the incoming ball's velocity vector, ensuring perpendicular impact and maximizing the effective capture area as shown in Fig. 1. The desired Cartesian end-effector pose is then defined as

$$\xi_{\text{goal}} = \begin{bmatrix} \mathbf{p}_{\text{int}} \\ \boldsymbol{\theta} \end{bmatrix} \in \mathbb{R}^6, \tag{1}$$

which fully specifies the position and orientation of the end-effector at interception. Inverse kinematics is used to map ξ to a joint-space goal configuration $\mathbf{q}_{\text{goal}} \in \mathbb{R}^7$.

Given the current joint configuration $\mathbf{q}_{\text{init}} \in \mathbb{R}^7$, the ball-catching problem is defined as computing joint torques $\boldsymbol{\tau}(t)$ such that the robot reaches ξ_{goal} before the ball arrives at the interception plane, while satisfying the manipulator dynamics

$$\boldsymbol{\tau}(t) = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}(t) + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}(t) + \mathbf{g}(\mathbf{q}), \tag{2}$$

where $\mathbf{M}(\mathbf{q})$ is the inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ represents Coriolis and centrifugal effects, and $\mathbf{g}(\mathbf{q})$ denotes the gravity vector.

To plan a motion that moves the robot from $\mathbf{q}_{\text{init}}$ to $\mathbf{q}_{\text{goal}}$, the motion planner must plan a collision-free motion in minimal time that enables the robot to reach $\mathbf{q}_{\text{goal}}$ to successfully catch the ball while avoiding collisions with the robot base. Crucially, motion planning must be completed as fast as possible with a consistent time budget so that sufficient time can be allocated for physical actuation and lower-level control.

III. Motion Planning Method

We propose a new motion planner, FlashPlan, that operates in robot joint space. At each planning instance, the robot is provided with $\mathbf{q}_{\text{goal}}$ along with a representation of the surrounding environment.

A. Environment Representation

To enable collision-free motion planning, FlashPlan incorporates two complementary environment representations: a structured context encoding and a spatial occupancy map. The context encoding captures task-related information, including the initial and goal joint configurations and obstacle descriptions. Each obstacle is represented by its Cartesian center location and 3D bounding box dimensions, and the combined context is arranged as a fixed-size 2D tensor suitable for a neural network.

In addition to the context representation, the environment is encoded as a 2D top-down occupancy grid that captures the spatial layout of obstacles in the workspace. This occupancy map provides a spatially structured representation of the environment and enables the planner to reason about obstacle locations relative to the robot. By separating structured task information from spatial environment information, FlashPlan can effectively leverage convolutional neural networks to extract high-level features from both representations.

B. Inputs and Outputs

FlashPlan takes $\mathbf{q}_{\text{init}}$, $\mathbf{q}_{\text{goal}}$, and a representation of the surrounding environment as input. The output of FlashPlan is a compact, time-normalized joint-space trajectory represented by a fixed number of path-points as illustrated in Fig. 3. Each path-point is defined as

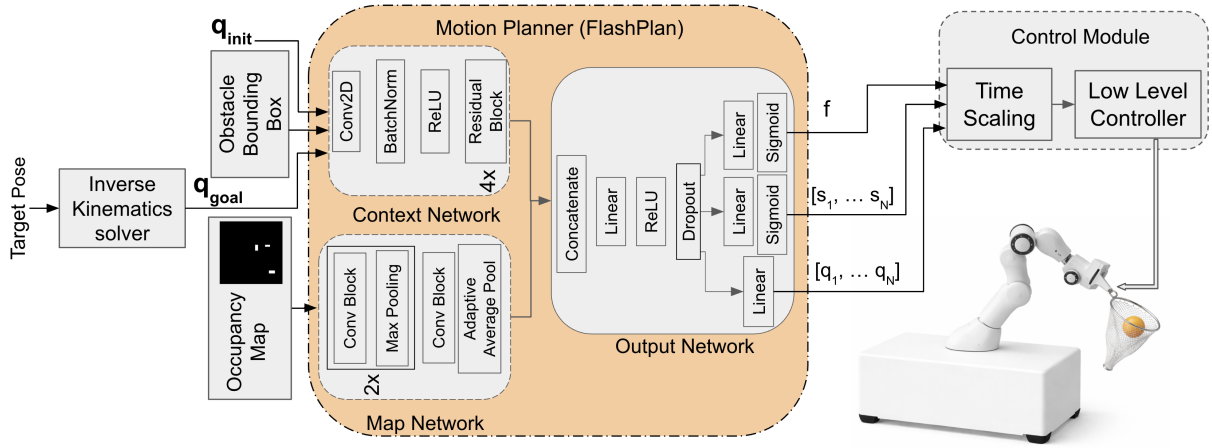

Fig. 2: robot motion planning and control pipeline.

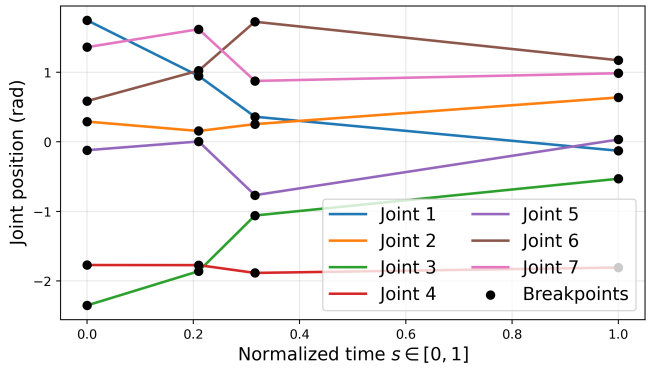

Fig. 3: Example time-normalized joint-space trajectory represented using a small number of path-points. The solid curves show the piecewise-linear joint trajectories for all seven joints of the Franka Emika Panda arm as a function of normalized time $s \in [0, 1]$. Circular markers indicate the path-points $(s_i, \mathbf{q}_i)$. A continuous trajectory is reconstructed by interpolating between these path-points.

a pair $(s_i, \mathbf{q}_i)$, where $s_i \in [0, 1]$ denotes a normalized time instance and $\mathbf{q}_i \in \mathbb{R}^7$ denotes a joint configuration. The planner predicts up to N path-points, yielding the trajectory to be

$$\mathcal{T} = \{(s_i, \mathbf{q}_i)\}_{i=1}^{N} \tag{3}$$

The normalized time parameterization decouples motion generation from absolute execution time, allowing the execution time to be scaled according to joint velocity and torque limits. Similar to prior learning-based planners such as MPNet [15], FlashPlan also includes a binary feasibility classifier that predicts whether a collision-free path exists between $\mathbf{q}_{\text{init}}$ and $\mathbf{q}_{\text{goal}}$ given the environment representation. This head approximates the classical motion-planning feasibility query while maintaining constant inference time.

C. Network Architecture

FlashPlan employs a neural architecture that separately encodes the context representation and the occupancy map before jointly predicting a compact motion. The planner consists of a *context network*, a *map network*, and a shared *output network*, all trained end-to-end.

The context network encodes the context representation. It consists of an initial convolutional block followed by a stack of residual blocks. A *residual block* [19] consists of two convolutional layers with batch normalization and ReLU activations, together with an identity skip connection:

$$\mathbf{y} = \mathbf{x} + \mathcal{F}(\mathbf{x}), \tag{4}$$

where $\mathcal{F}$ denotes the stacked convolutional transformations. Dropout is applied after the second activation for regularization. This design enables the network to learn higher-level relational features that capture the interaction between the robot state, task objective, and nearby obstacles, producing a compact context embedding.

In parallel, the map network processes a 2D top-down occupancy grid representing the spatial layout of obstacles in the workspace. As illustrated in Fig. 2, the map network applies repeated convolutional blocks interleaved with spatial downsampling to progressively aggregate global environment information. A *convolutional block* is defined as:

$$\text{Conv2D} \rightarrow \text{BatchNorm} \rightarrow \text{ReLU}.$$

An adaptive average pooling layer produces a fixed-dimensional map embedding, allowing the network to handle varying obstacle configurations while preserving spatial structure relevant to collision avoidance.

The context and map embeddings are concatenated and passed to a shared output network. The output network consists of a shared multilayer perceptron backbone followed by three task-specific heads. These heads jointly predict: (i) a feasibility score $f \in [0, 1]$

indicating whether a collision-free motion exists, (ii) a set of normalized time indices $\{s_i\}_{i=1}^N$ with $s_i \in [0,1]$, and (iii) the corresponding joint-space path-point configurations $\{\mathbf{q}_i\}_{i=1}^N$. Each pair $(s_i, \mathbf{q}_i)$ defines a waypoint of a time-normalized joint-space trajectory.

By predicting all trajectory parameters in a single forward pass, FlashPlan avoids iterative search and enables fast planning suitable for time-critical tasks such as robotic ball catching.

D. Training Data Generation

FlashPlan is trained entirely offline using expert demonstrations generated by a classical motion planner. We treat the sampling-based planner *RRTConnect* [8] as an expert that provides collision-free joint-space trajectories in cluttered environments. For each training sample, the environment is randomized by placing multiple box obstacles at random locations within the robot workspace, including configurations that partially obstruct the arm base. A random collision-free $\mathbf{q}_{\text{init}}$ and $\mathbf{q}_{\text{goal}}$ are sampled for each environment instance.

Given $(\mathbf{q}_{\text{init}}, \mathbf{q}_{\text{goal}})$ and the obstacle configurations, RRTConnect is invoked to compute a collision-free joint-space trajectory. When RRTConnect successfully finds a solution, the resulting trajectory is treated as a feasible expert demonstration. If the planner fails to find a solution, the sample is labeled as infeasible, and the trajectory waypoints are padded using $\mathbf{q}_{\text{init}}$ signifying no motion.

The joint-space trajectories produced by RRTConnect are inherently piecewise linear due to the tree-based nature of the planner. To obtain a compact trajectory representation suitable for learning, we apply the Ramer–Douglas–Peucker (RDP) algorithm ([20], [21]) to each expert trajectory. RDP identifies the path-points in the RRTConnect generated trajectories. These path-points are then reparameterized using normalized time $s \in [0,1]$, yielding a compact, time-normalized trajectory representation $\{(s_i, \mathbf{q}_i)\}_{i=1}^N$.

Using this procedure, a dataset of 10,000 samples is generated, covering both feasible and infeasible planning scenarios. The dataset is split into 80% training, 10% validation, and 10% testing sets. All neural network components of FlashPlan are trained using this dataset, while RRTConnect is used only during offline data generation and not at runtime.

E. Training Objective and Optimization

FlashPlan is trained using a weighted multi-task loss that jointly supervises feasibility prediction, trajectory timing, joint-space accuracy, and motion smoothness. The total training loss is defined as

$$\mathcal{L} = w_f \mathcal{L}_f + w_s \mathcal{L}_s + w_q \mathcal{L}_q + w_g \mathcal{L}_g + w_{\dot{q}} \mathcal{L}_{\dot{q}}, \qquad (5)$$

where each term corresponds to a specific prediction objective.

The feasibility loss $\mathcal{L}_f$ is defined as a binary cross-entropy loss between the ground-truth feasibility label and the predicted feasibility score f. This term encourages the network to distinguish between solvable and unsolvable planning instances. The normalized time loss $\mathcal{L}_s$ penalizes errors in the predicted path-point times instances $\{s_i\}$ using mean squared error (MSE). This loss encourages consistent temporal spacing of trajectory waypoints in normalized time. The joint configuration loss $\mathcal{L}_q$ is defined as the MSE between predicted and expert joint-space path-point configurations $\{\mathbf{q}_i\}$. This term encourages accurate reconstruction of the expert trajectory in joint space. To explicitly enforce goal reaching behavior, a goal loss $\mathcal{L}_g$ is applied to the last path-point configuration. This loss penalizes the deviation between the predicted last path-point joint configuration and the target configuration $\mathbf{q}_{\text{goal}}$, ensuring that the generated motion terminates at the desired intercept configuration. To promote smooth motions and reduce abrupt transitions between consecutive path-points, a velocity regularization term $\mathcal{L}_{\dot{q}}$ is introduced. This term penalizes the MSE between the first-order finite differences of predicted joint configurations and those of the expert trajectory.

In all experiments, the loss weights are set to $w_f = w_s = w_g = 1$ and $w_q = w_{\dot{q}} = 5$. The network is trained using the Adam optimizer to minimize the total loss.

IV. Motion Control Method

As the planned motion is independent of execution speed, a total execution time $T > 0$ is assigned, mapping normalized time to real time via

$$t = sT \quad , \quad t \in [0, T]. \qquad (6)$$

The execution joint trajectory in real time is therefore given by $\mathbf{q}(t) = \mathbf{q}(s)$. Using the chain rule, joint velocities and accelerations scale with the execution time T as

$$\dot{\mathbf{q}}(t) = \frac{1}{T}\frac{d\mathbf{q}}{ds} \quad , \quad \ddot{\mathbf{q}}(t) = \frac{1}{T^2}\frac{d^2\mathbf{q}}{ds^2}. \qquad (7)$$

Thus, executing the same geometric trajectory over a shorter duration results in higher joint velocities and accelerations, while longer execution times reduce the magnitudes.

The robot is controlled using a torque-based low-level controller. For a 7-DoF robotic manipulator, the joint torques required to execute the trajectory are given by the robot kinematic model described in (2). Since inertial and Coriolis terms scale inversely with T^2, while gravity torques are independent of T, shorter execution times lead to higher torque demands, whereas longer execution times reduce dynamic torques.

For a candidate execution time T, we evaluate torque feasibility by computing the required joint torques $\boldsymbol{\tau}(t_k; T)$ along the time-scaled trajectory at discrete samples $k \in \{1, \ldots, K\}$. We define the peak normalized

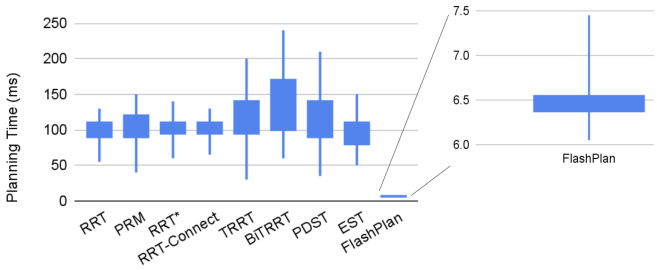

Fig. 4: Planning latency comparison between FlashPlan and sampling-based planners. FlashPlan achieves substantially lower and more consistent planning latency across identical planning queries.

torque usage

$$\Phi(T) \triangleq \max \frac{|\tau_i(t_k;T)|}{\tau_{i,\max}}, \quad i \in [1,2,\ldots,7] \tag{8}$$

which represents the maximum fraction of the torque limit used by any joint at any time for a complete execution duration of T. Here $\tau_{\max}$ represents the maximum physical torque that can be generated by the joint motors. The trajectory is torque-feasible if $\Phi(T) \leq 1$.

We choose the fastest feasible execution time by solving

$$T^\star = \min_{T>0} T \quad \text{s.t.} \quad \Phi(T) \leq 1. \tag{9}$$

Because $\dot{\mathbf{q}}(t)$ scales as $1/T$ and $\ddot{\mathbf{q}}(t)$ scales as $1/T^2$, the required dynamic torques decrease monotonically as T increases, making $\Phi(T)$ a monotonically decreasing function of T. This monotonicity enables an efficient binary search to compute $T^\star$ with negligible runtime overhead.

The time-scaled joint-space trajectory is executed on the robot using a torque-based low-level controller. Specifically, a joint-space PD controller tracks the desired joint positions and velocities derived from the scaled trajectory and outputs torque commands at each control cycle.

V. Simulation Results

We construct a simulation environment in PyBullet, a Python interface to the Bullet physics engine, using a Franka Emika Panda robotic manipulator model. The Orocos KDL inverse kinematics solver [22] is used to compute the joint-space goal configuration $\mathbf{q}_{\text{goal}}$ from the Cartesian interception pose ξ. FlashPlan is trained on an NVIDIA A6000 GPU. Although FlashPlan supports both CPU and GPU inference, all evaluation experiments are performed on an Intel Xeon Gold 6226R CPU to ensure a fair comparison with classical sampling-based planners, which are CPU-based. We evaluate FlashPlan using offline planning benchmarks and simulated robotic ball-catching experiments. The simulation environment enables controlled evaluation,

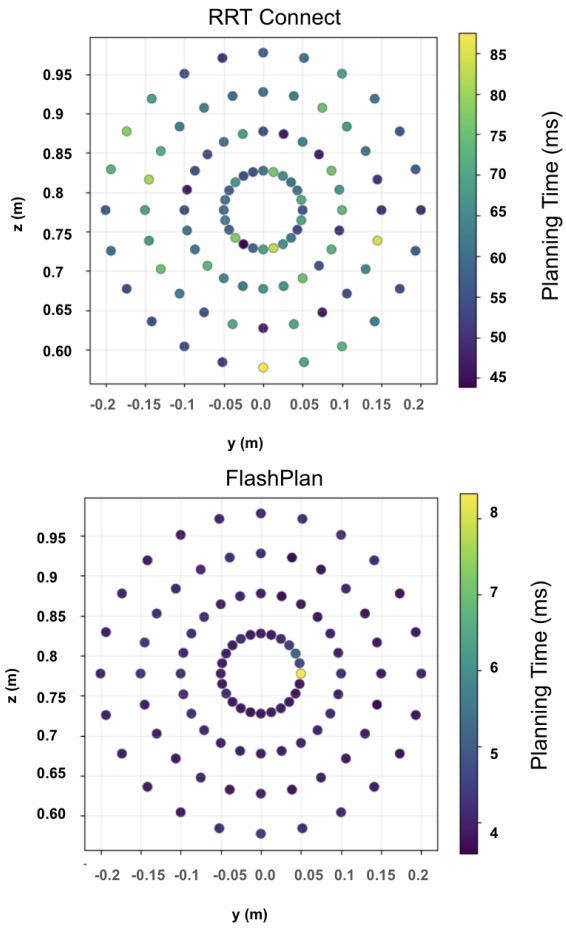

Fig. 5: Comparison between the RRT Connect method (top) and our proposed motion planner (bottom) on planning time.

where perception noise and low-level control uncertainties can be abstracted, allowing a focused assessment of the motion-planning component only. First, we compare FlashPlan against classical sampling-based motion planners in terms of planning time and consistency across the workspace. We then analyze the qualitative behavior of the learned collision-free trajectory estimation components. Finally, we demonstrate the FlashPlan on a robotic ball-catching task.

A. Planning Latency Comparison

We compare FlashPlan against several classical sampling-based motion planners: RRT [6], PRM [23], RRT-star [10], RRT-Connect [8], TRRT [24], BiTRRT [24], PDST [25] and EST [26]. All planners are evaluated on identical planning queries generated in randomized obstacle environments, and planning time is measured from query initialization to the generation of a collision-free joint-space trajectory.

Fig. 4 shows the distribution of planning times across methods. Classical planners exhibit median planning times in the range of approximately 90–160 ms, with

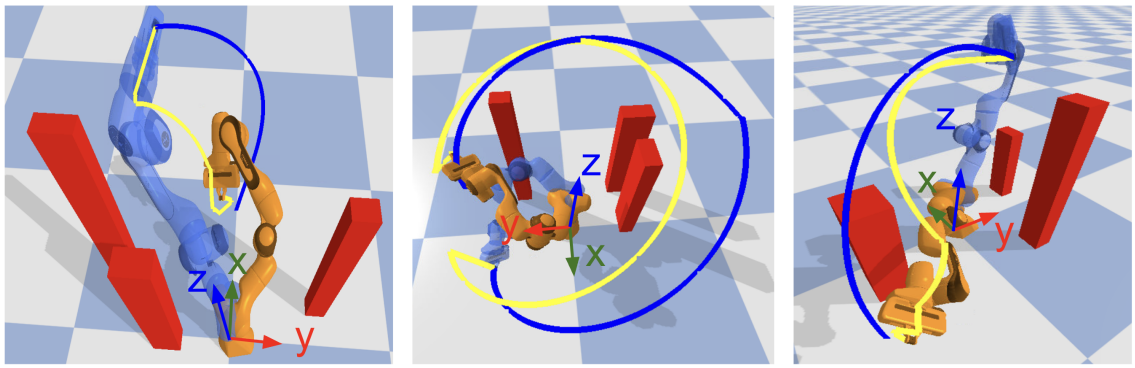

Fig. 6: Motion trajectories generated by RRT-Connect and FlashPlan across three representative planning scenarios. The faint blue robot indicates $\mathbf{q}_{\text{init}}$, while the orange robot denotes $\mathbf{q}_{\text{goal}}$. The red boxes represent obstacles. The blue line corresponds to the end-effector trajectory produced by RRT-Connect, and the yellow line corresponds to the trajectory planned by FlashPlan.

significant variance depending on the specific start–goal configuration and obstacle layout. In contrast, FlashPlan achieves a median planning time of approximately 6.5 ms with very low variance, demonstrating both substantially reduced latency.

To further analyze latency consistency across the workspace, Fig. 5 evaluates how planning time varies as a function of the intercept location on the predefined interception plane. In this experiment, the robot initial configuration $\mathbf{q}_{\text{init}}$ is fixed, and a dense set of Cartesian intercept positions is sampled uniformly over a circular region on the interception plane. For each planning query $(\mathbf{q}_{\text{init}}, \mathbf{q}_{\text{goal}})$, we measure the time required to generate a collision-free trajectory using either RRT-Connect and FlashPlan. The resulting planning times are visualized spatially. In this test, RRT-Connect exhibits strong spatial variability, with planning time fluctuating significantly depending on the intercept location. This variability arises from the stochastic sampling process and geometry-dependent exploration behavior inherent to classical planners. In contrast, FlashPlan maintains consistently low and uniform planning latency across all tested locations, since trajectory generation is performed in a single forward pass independent of workspace geometry.

This spatial consistency is particularly important for time-critical interception tasks. Because planning latency directly reduces the available control window, spatially variable planning time translates into unpredictable execution budgets. FlashPlan's location-independent inference time therefore provides a consistent reaction horizon, which is essential for reliable high-speed robotic ball catching.

B. Motion Planning Performance

Fig. 6 presents a qualitative comparison of end-effector trajectories generated by RRT-Connect and FlashPlan under identical obstacle configurations. Both planners produce collision-free paths that successfully avoid obstacles. FlashPlan generates trajectories that are geometrically comparable to those of RRT-Connect, indicating that the learned model effectively captures feasible motion patterns in cluttered environments. Unlike RRT-Connect, however, FlashPlan predicts the trajectory in a single forward pass without iterative sampling or tree expansion.

Fig. 7 shows a representative time-normalized joint-space trajectory generated by FlashPlan. The horizontal axis represents normalized time $s \in [0, 1]$, decoupled from physical execution time. All joint trajectories evolve smoothly and converge close to the desired goal configuration $\mathbf{q}_{\text{goal}}$ at $s = 1$. The close alignment between the predicted final joint values (at $s = 1$) and $\mathbf{q}_{\text{goal}}$ confirms the planned trajectory converges close to the joint goals.

We also evaluate the performance of the collision-free feasibility indicator head. Fig. 8 shows the training and validation accuracy over the course of training. The feasibility head achieves approximately 90% validation accuracy, indicating effective generalization to unseen obstacle configurations and start–goal pairs. The accuracy on our test set was 89.2%.

C. Robotic Ball Catching Performance

We evaluate FlashPlan on simulated robotic ball catching task and compare it against two classical sampling-based planners, RRT-Connect and EST, which demonstrated the lowest planning times among baseline methods (Fig. 4). The balls were thrown from a distance of 4 meters. All methods share the same inverse kinematics solver, torque-constrained time scaling procedure, and low-level controller. The only difference lies in the motion planning component.

For each episode, a ball trajectory is generated and an interception point is estimated at the interception plane. Let t_{arr} denote the estimated time-to-ball-arrival computed after perception (we assume a fixed perception latency of 100ms), and let t_{plan} denote the planning

TABLE I: Robotic ball catching performance comparison under simulated throws.

Motion Planner	SR (%)	Average $t_{control}$ (ms)	$v_{x,75}$ (m/s)	$v_{x,50}$ (m/s)
EST	82.8	221.6	6.77	7.25
RRT-Connect	87.35	237.2	6.90	7.48
FlashPlan	**95.2**	**330.6**	**8.51**	**8.86**

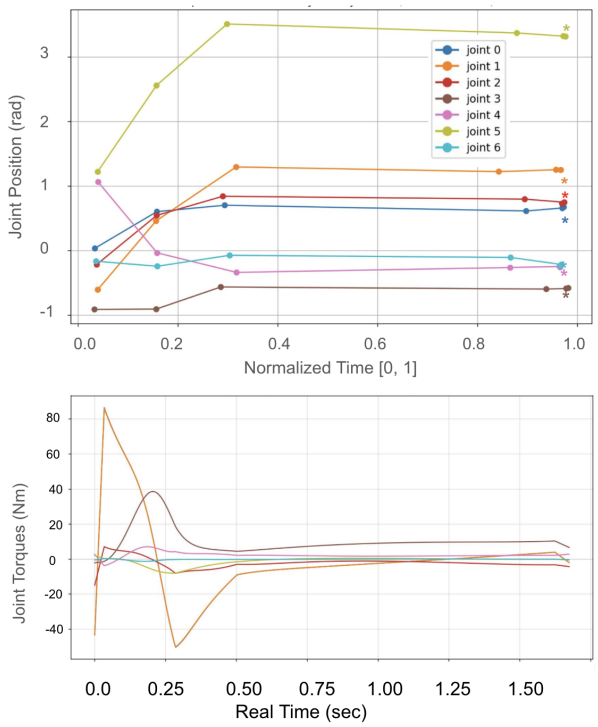

Fig. 7: Time-normalized joint-space trajectory generated by FlashPlan, and the corresponding joint torques. Each curve represents one of the seven joints. The asterisks (*) denote the desired joint-space goal configuration $\mathbf{q}_{\text{goal}}$. The predicted trajectory converges to the specified goal at $s = 1$, demonstrating accurate terminal constraint satisfaction.

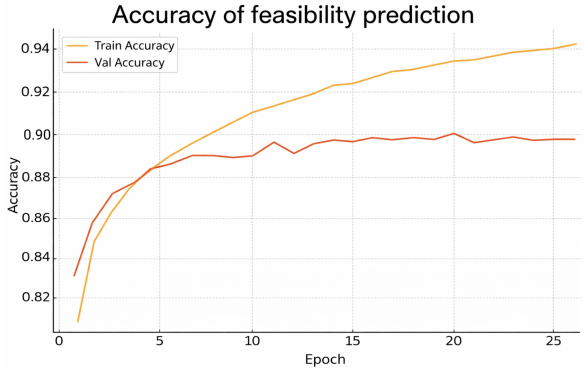

Fig. 8: Training and validation accuracy of the collision-free feasibility prediction head.

latency. The time window available for motion control is defined as $t_{\text{control}} = t_{\text{arr}} - t_{\text{plan}}$. This quantity represents the execution time budget available to the controller prior to ball arrival at the interception plane.

To characterize robustness to faster incoming balls, we measure the ball flying speed $|v_x|$ at the interception plane along the x axis towards the robot. Specifically, $v_{x,75}$ and $v_{x,50}$ denote the highest approach speeds for which the planner achieves at least 75% and 50% catch success, respectively. These thresholds quantify the effective velocity envelope within which interception remains reliable.

As shown in Table I, FlashPlan achieves the highest success rate (SR) of 95.2% while providing a substantially larger control window (330.6 ms) compared to EST (221.6 ms) and RRT-Connect (237.2 ms). The increased control time directly translates into improved interception capability: FlashPlan sustains successful catches at significantly higher approach speeds, increasing $v_{x,75}$ to 8.51 m/s and $v_{x,50}$ to 8.86 m/s. In contrast, the classical planners exhibit narrower velocity margins due to longer and more variable planning times. A snapshot of ball catching task is shown in Fig. 9.

These results demonstrate that consistent, low-latency motion planning enlarges the effective control window and expands the feasible interception velocity range. For time-critical dynamic manipulation tasks such as robotic ball catching, reducing planning latency is therefore not merely computationally beneficial, but physically consequential.

VI. Conclusion

This paper presented a neural motion planner designed for time-critical robotic interception tasks. By directly predicting compact, collision-free, time-normalized joint-space trajectories in a single forward pass, FlashPlan achieves low and consistent planning latency. A torque-constrained time-scaling formulation further ensures dynamic feasibility while maximizing the available execution window prior to interception. While the proposed approach shows strong performance for interception tasks, several limitations remain. First, the planner is trained offline on trajectories generated by a classical planner and therefore inherits the structural biases of the expert planner. Second, retraining is required when transferring the method to a different robot due to differences in physical dimensions. Finally, although torque-constrained time scaling ensures dynamic feasibility of motion control, tracking

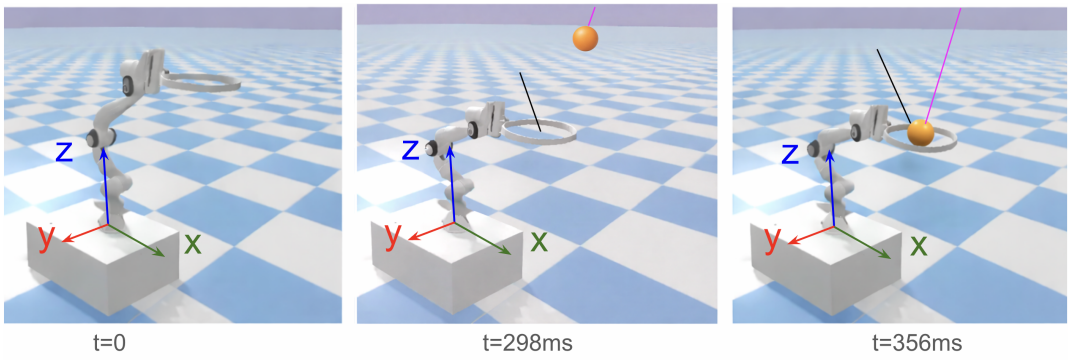

Fig. 9: Simulated robotic ball catching sequence. From left to right: (1) robot in initial joint configuration $\mathbf{q}_{\text{init}}$, (2) ball approaching the robot, and the robot approaching $\mathbf{q}_{\text{goal}}$ (3) successful capture of the ball.

performance ultimately depends on the tuning of the low-level controller and hardware limitations.

In future work, we aim to extend the current experimental setup to incorporate direct visual perception using high-speed cameras. While the present study assumes that the interception pose is estimated upstream, integrating raw visual observations would enable evaluation of the complete perception–planning–control pipeline under realistic sensing delays and noise. In particular, high-frame-rate cameras or event-based vision sensors could provide low-latency ball state estimation, further reducing reaction time and enabling interception at higher approach velocities. Beyond modular integration, we also plan to investigate end-to-end training strategies that learn a direct mapping from visual observations to motion generation. Such an approach would allow the neural motion planner to jointly optimize perception and trajectory prediction, potentially improving robustness to perception uncertainty and enhancing overall system performance in dynamic, high-speed manipulation tasks.

References

[1] R. L. Anderson, "A robot ping-pong player," *MIT AI Lab Memo*, 1988.

[2] M. Matsushima, T. Hashimoto, M. Takeuchi, and F. Miyazaki, "A learning approach to robotic table tennis," *IEEE Transactions on robotics*, vol. 21, no. 4, pp. 767–771, 2005.

[3] K. Mülling, J. Kober, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *The International Journal of Robotics Research*, 2013.

[4] Y. Gao, S. Schaal, and S. Levine, "Learning robot table tennis from demonstration and reinforcement learning," *IEEE Robotics and Automation Letters*, 2020.

[5] M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal, "Learning, planning, and control for quadruped locomotion over challenging terrain," *The International Journal of Robotics Research*, 2011.

[6] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[7] L. E. Kavraki, M. N. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," *IEEE Transactions on Robotics and automation*, vol. 14, no. 1, pp. 166–171, 1998.

[8] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[9] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *IJRR*, 2001.

[10] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, 2011.

[11] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE international conference on robotics and automation*, 2009.

[12] 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.

[13] K. Hauser, "The minimum constraint removal problem with applications to planning," *The International Journal of Robotics Research*, 2011.

[14] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[15] A. H. Qureshi and M. C. Yip, "Motion planning networks: : Bridging the gap between learning-based and classical motion planners," *IEEE Transactions on Robotics*, 2019.

[16] M. Dalal and et al., "Neural motion planner," *IEEE Robotics and Automation Letters*, 2023.

[17] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, 2016.

[18] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*, 2017.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[20] U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," *Computer Graphics and Image Processing*, 1972.

[21] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica*, 1973.

[22] H. Bruyninckx, "Open robot control software: the orocos project," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2001.

[23] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," in *IEEE International Conference on Robotics and Automation (ICRA)*, 1996.

[24] L. Jaillet, J. Cortés, and T. Siméon, "Transition-based rrt for path planning in continuous cost spaces," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008, pp. 2145–2150.

[25] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, 2012.

[26] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *IEEE International Conference on Robotics and Automation (ICRA)*, 1997.