Adaptive Reinforcement Learning Based Robotic Arm Trajectory Planning for Obstacle Avoidance

Barath Balamurugan, Adithya Rajendran
MS in Robotics
{balamurugan.b, rajendran.a}@northeastern.edu

***Abstract*—**This paper presents a reinforcement learning-based approach for robotic arm trajectory planning that integrates three key methodologies: Proximal Policy Optimization (PPO), Semi-Markov Decision Processes (SMDP), and Model Predictive Control (MPC). The system aims to enable robotic arms to navigate complex environments, avoid obstacles, and reach target positions efficiently. We implement a PyBullet simulation environment with customizable physics parameters and a carefully designed reward function that encourages desirable behaviors. Our approach uses temporal abstraction through SMDP to handle actions with varying durations and combines the global planning capabilities of reinforcement learning with the local precision of MPC. Experimental results show that the hybrid PPO + SMDP + MPC approach outperforms both standalone PPO and PPO + SMDP in terms of success rate, reward accumulation, and trajectory efficiency. This research demonstrates how the combination of complementary control methodologies can effectively address the challenges of robotic arm trajectory planning.

***Index Terms*—**Reinforcement Learning, Proximal Policy Optimization, Semi-Markov Decision Process, Model Predictive Control, Robotic Arm, Trajectory Planning, Obstacle Avoidance

I. Introduction

Robotic arm trajectory planning presents significant challenges, particularly in environments with obstacles, dynamic constraints, and precise target requirements. Traditional approaches often struggle with the complexity and high dimensionality of the problem space. Reinforcement learning (RL) offers a promising alternative by enabling robots to learn optimal behaviors through experience, but standard RL implementations can be sample inefficient and struggle with continuous control tasks.

This research addresses these challenges by integrating three powerful methodologies:

- **Proximal Policy Optimization (PPO)**: A widely-used, sample-efficient reinforcement learning algorithm that belongs to the family of policy gradient methods. PPO strikes a balance between performance and stability by constraining policy updates through a clipped objective function. This prevents the new policy from deviating too far from the previous one, reducing the risk of performance collapse. PPO enables stable learning by incorporating both exploration and exploitation effectively, and is well-suited for continuous control problems like robotic trajectory planning due to its ability to handle high-dimensional action spaces and dynamically changing environments.
- **Semi-Markov Decision Processes (SMDP)**: An extension of the traditional Markov Decision Process (MDP) framework that allows actions to take variable amounts of time to complete. Unlike standard MDPs, where actions are assumed to result in transitions at fixed time steps, SMDPs are designed to model temporally extended actions, also known as macro-actions or options. This makes SMDPs particularly suitable for real-world scenarios like robotic control, where some actions—such as moving to a waypoint or avoiding an obstacle—may span multiple time steps. The SMDP framework enables reinforcement learning agents to reason not just about which action to take, but also how long it will take to execute, leading to more realistic and efficient long-term planning in dynamic environments.
- **Model Predictive Control (MPC)**: A powerful and flexible control strategy that optimizes control actions by predicting future states of a system over a finite time window, known as the receding horizon. At each time step, MPC solves an optimization problem to generate a sequence of control actions that minimize a cost function subject to system dynamics and constraints (e.g., collision avoidance, velocity limits). Only the first control input is applied before the process repeats at the next timestep using updated state information. This approach allows the

controller to continuously adapt to changes in the environment, making MPC well-suited for real-time robotic applications where precision, safety, and responsiveness are critical—especially in dynamic settings involving moving obstacles and uncertain disturbances.

By combining these approaches, we aim to leverage the global planning capabilities of reinforcement learning with the local precision and real-time responsiveness of Model Predictive Control (MPC). Temporal abstraction is achieved through the use of Semi-Markov Decision Processes (SMDP), enabling the agent to execute macro-actions that span multiple timesteps and thus improve learning efficiency in complex tasks. The system is initially developed and tested in a 2D environment, where the fundamental perception-action loop, state representation, and reward structure are validated. Following this, the framework is extended to a 3D workspace using the PyBullet simulation environment, which offers realistic physics and visual feedback, allowing us to evaluate the full spatial and dynamic behavior of the robotic arm in more realistic scenarios.

Our main contributions include:

- An integrated reinforcement learning framework for robotic arm trajectory planning that combines Proximal Policy Optimization (PPO), Semi-Markov Decision Processes (SMDP), and Model Predictive Control (MPC).
- A carefully designed reward function that balances goal achievement, obstacle avoidance, and motion efficiency, providing the agent with clear and consistent learning signals to accelerate policy convergence.
- A curriculum learning approach that gradually increases task difficulty during training.
- Comparative analysis of different control methodologies (PPO, PPO + SMDP, PPO + SMDP + MPC) on robotic arm tasks.

II. Related Work

Robotic arm control has a rich history in both robotics and artificial intelligence. Classical approaches have typically relied on techniques such as inverse kinematics [9], sampling-based motion planning algorithms like RRT and PRM [10], and optimal control theory [11]. While effective in structured environments, these methods often require precise models of the robot and its surroundings, which may be difficult to obtain or maintain in real-world scenarios with dynamic changes or uncertainty.

In recent years, reinforcement learning (RL) has emerged as a promising alternative for robotic control tasks [12]. Among RL methods, policy gradient algorithms—notably Proximal Policy Optimization (PPO) [6]—have demonstrated strong performance in continuous control settings due to their balance between sample efficiency and training stability. However, traditional RL frameworks generally assume that actions occur at fixed intervals, which limits their applicability in scenarios where actions may naturally span variable durations.

To address this, Semi-Markov Decision Processes (SMDPs) [7] extend the standard Markov Decision Process (MDP) by allowing actions to last for variable amounts of time. This temporal abstraction enables agents to learn macro-actions, which are especially beneficial in robotic tasks requiring long sequences of coordinated movements. The options framework [14] is one such implementation that facilitates learning and execution of temporally extended actions and has seen success in robotic applications [13].

On the other hand, Model Predictive Control (MPC) [8] remains a cornerstone in classical control theory and has been successfully applied to robotic manipulation tasks [15]. More recent efforts have investigated hybrid models that integrate MPC with learning-based techniques [16], capitalizing on the adaptability of RL and the real-time precision of MPC.

While the integration of RL and MPC has been studied in various contexts [17], our work presents a novel combination of PPO, SMDP, and MPC specifically tailored for robotic arm trajectory planning. This hybrid framework leverages the global planning capability of PPO, the temporal abstraction of SMDP, and the fine-grained real-time correction of MPC, addressing the individual limitations of each component and enabling robust operation in dynamic and uncertain environments.

III. System Architecture

Our system consists of three main components: A. Environment Simulation, B. Reinforcement Learning Agent, and C. Control System. Fig. 1 illustrates the high-level architecture and information flow.

A. Environment

The environment is built using PyBullet, open-source physics simulator that provides accurate rigid-body dynamics and real-time visualization. This platform enables realistic interaction modeling between the robotic arm, obstacles, and workspace components. It includes:

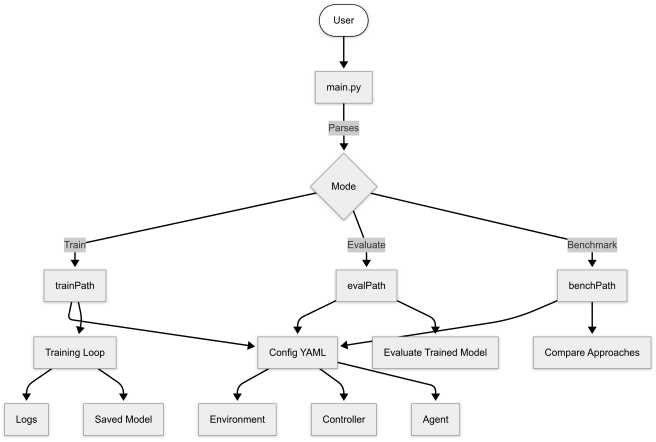

Fig. 1. High-level execution pipeline showing training, evaluation, and benchmarking paths.

- A KUKA robotic arm.
- Dynamic Obstacles with Customizable Positions and Sizes: Obstacles with varying positions and sizes are randomly initialized in each episode. This ensures that the agent generalizes its policy to a variety of workspace layouts. Obstacles are treated as rigid bodies with real-time collision detection.
- End-Effector position (3D): The Cartesian coordinates of the end effector in the world frame, used for trajectory tracking and reward computation.
- Goal Position (3D): A fixed or dynamic target position that the end effector must reach. This position can be randomized during curriculum learning.
- Obstacle Positions (3 × 3D): The environment includes up to three dynamic obstacles, each represented by its 3D position in the workspace.

State Space Definition

The agent's state at each timestep is represented as a structured tuple:

$$S_t = \{\text{End-Effector Position}, \text{Obstacle Position}, \text{Goal Position}, \text{Velocity}\}$$

More specifically, the numerical representation of this state includes:

- Position of the end effector in the 3D space.
- Position of the obstacle in 3D space.
- Goal position in a 3D space.
- End-effector velocity in 3D space.

Action Space

The agent's action at each timestep is defined as:

$$A_t = \{\text{Move to Waypoint}, \text{Avoid Obstacle}\}$$

Each action represents a temporally extended macro-action, suitable for modeling in a Semi-Markov Decision Process (SMDP). In our implementation:

- **Move to Waypoint**: Directs the end-effector toward an intermediate or goal position.
- **Avoid Obstacle**: Executes a motion sequence to bypass a detected obstacle in the workspace.

In the SMDP setting, each macro-action may span multiple timesteps and can be terminated based on state feedback or dynamic changes in the environment.

Depending on the setup, the action space can be:

- **Discrete**: Selecting between predefined macro-actions (e.g., move, avoid).
- **Continuous**: Outputting joint velocity commands or end-effector deltas along with a duration value.

B. Reinforcement Learning Agent

The RL agent is based on the **Proximal Policy Optimization (PPO)** algorithm, a sample-efficient policy gradient method known for its training stability and robustness in continuous control tasks. The agent architecture is composed of the following components:

- **Actor network**: Outputs action distributions over the defined action space, enabling stochastic policy exploration.
- **Critic network**: Estimates the value function $V(s)$ to guide the policy update via advantage estimation.
- **Rolling horizon planning**: A short-term trajectory optimization strategy where the agent re-evaluates its plan continuously over a fixed time window. This improves adaptability in dynamic environments with changing obstacle positions.

C. Control System

The control system employs a **hybrid architecture** that integrates the high-level decision-making of the reinforcement learning (RL) policy with the fine-grained trajectory refinement of **Model Predictive Control (MPC)**.

- **RL Policy:** Used for *global planning* and navigating toward the goal in free-space regions of the workspace.
- **MPC Module:** Activated for *precise positioning* near the goal and for *obstacle avoidance*, where small corrective motions are required.

The **MPC controller** follows a receding horizon approach, in which it solves an optimization problem over a short future time window at every control step. Only the first control input is applied, and the optimization is repeated using updated state information in the next timestep.

By combining the adaptability and exploration capability of RL with the precision and constraint satisfaction of MPC, the hybrid system ensures robust, efficient, and safe trajectory execution in complex environments.

IV. Methodology

A. Reinforcement Learning Formulation

We formulate the robotic arm trajectory planning problem as a reinforcement learning task defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$, where:

- $\mathcal{S}$ is the state space, capturing the robot's end-effector position, obstacle position, goal location, and velocity
- $\mathcal{A}$ is the action space, consisting of either joint velocity commands or temporally extended macro-actions (in the SMDP formulation)
- $\mathcal{P}$ is the transition probability distribution between states
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function providing feedback for goal-reaching, obstacle avoidance, and motion efficiency
- $\gamma \in [0, 1]$ is the discount factor for future rewards

The objective is to learn a stochastic policy $\pi(a|s)$ that maximizes the expected cumulative discounted reward:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{T} \gamma^t r_t \right] \tag{1}$$

where $\tau = (s_0, a_0, r_0, s_1, \ldots, s_T)$ represents a trajectory sampled from the policy π, and T denotes the episode horizon. In the SMDP setting, actions may span multiple timesteps.

B. PPO Implementation

Our reinforcement learning agent is trained using **Proximal Policy Optimization (PPO)**, a first-order policy gradient algorithm that improves training stability by limiting the magnitude of policy updates. PPO achieves this using a clipped objective function, which penalizes updates that move the policy too far from the previous iteration. The clipped surrogate objective used in PPO is defined as:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{2}$$

where:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the new and old policies
- $\hat{A}_t$ is the estimated advantage at timestep t
- ϵ is the clipping parameter (typically $\epsilon = 0.1$ or 0.2)

To compute $\hat{A}_t$, we use **Generalized Advantage Estimation (GAE)**, which balances bias and variance in advantage calculation. The advantage estimate is computed as:

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{3}$$

with the temporal-difference (TD) error defined as:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{4}$$

Here, γ is the discount factor and λ is the GAE parameter, typically set between 0.9 and 0.95. This formulation enables stable and efficient learning by controlling the update size while still benefiting from multi-step returns.

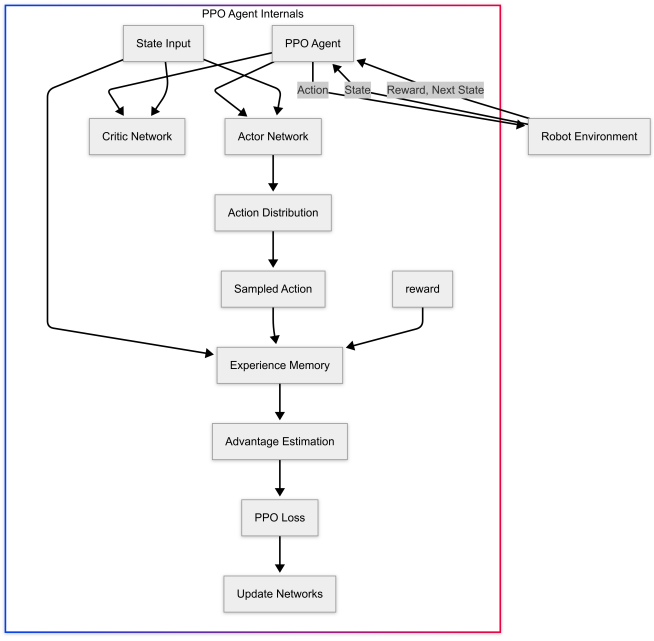

Fig. 2. Detailed PPO training flow, showing the interaction between agent, environment, and learning updates.

C. SMDP Extension

To support **temporal abstraction**, we extend the standard Markov Decision Process (MDP) to a **Semi-Markov Decision Process (SMDP)** formulation. In this framework, actions are no longer assumed to take a fixed duration; instead, each action is executed over a variable time span.

The policy is now factorized to output both an action and a duration:

$$\pi(a, d \mid s) = \pi_a(a \mid s) \cdot \pi_d(d \mid s) \tag{5}$$

where:

- $\pi_a(a \mid s)$ is the standard action policy
- $\pi_d(d \mid s)$ is a learned duration policy that determines how long to execute the chosen action

This enables the agent to learn *macro-actions*—temporally extended behaviors such as "move to waypoint" or "avoid obstacle"—that can span multiple timesteps.

The reward function is modified to account for the temporal nature of actions:

$$R(s, a, d, s') = \sum_{i=0}^{d-1} \gamma^i r(s_i, a) \tag{6}$$

Here, s_i represents intermediate states encountered during the execution of action a for duration d, and γ is the discount factor. This adjustment ensures that longer actions are evaluated based on their cumulative contribution to the task, rather than per-step performance alone.

By learning both the *action* and the *duration*, the SMDP extension allows the agent to operate over multiple time scales, improving planning efficiency and behavioral smoothness in complex environments.

D. MPC Integration

To enhance local precision and safety in dynamic environments, we integrate a **Model Predictive Control (MPC)** module with the reinforcement learning policy in a hybrid control architecture.

The MPC controller optimizes a control sequence over a finite prediction horizon H by solving the following optimization problem:

$$\min_{a_{0:H-1}} \sum_{t=0}^{H-1} c(s_t, a_t) + c_f(s_H) \tag{7}$$

subject to the system dynamics:

$$s_{t+1} = f(s_t, a_t)$$

Here:

- $c(s_t, a_t)$ is the **stage cost**, penalizing deviation from desired trajectories or proximity to obstacles.
- $c_f(s_H)$ is the **terminal cost**, penalizing final-state error at the end of the horizon.
- f is the system's dynamics function, either learned or analytically defined.

Reward Function Design

The reward function plays a central role in shaping the agent's behavior by providing feedback that aligns with the task objectives. We design a composite reward signal composed of multiple components, each targeting a specific aspect of desired behavior:

$$r = r_{\text{dist}} + r_{\text{prog}} + r_{\text{goal}} + r_{\text{time}} + r_{\text{coll}} + r_{\text{move}} + r_{\text{expl}} \tag{8}$$

where each term is defined as:

- **Distance Penalty:**

  $$r_{\text{dist}} = -10 \cdot \|p_{ee} - p_{goal}\|$$

  Penalizes the Euclidean distance between the end-effector position p_{ee} and the goal position p_{goal}.
- **Progress Reward:**

  $$r_{\text{prog}} = 30 \cdot (d_{\text{prev}} - d_{\text{curr}})$$

  Rewards progress toward the goal, where d_{prev} and d_{curr} are the previous and current distances to the goal, respectively.
- **Goal Bonus:**

  $$r_{\text{goal}} = 500 \cdot \mathbf{1}[d_{\text{curr}} < \delta_{\text{goal}}]$$

  Provides a large positive reward upon reaching the goal within a threshold δ_{goal}.
- **Time Penalty:**

  $$r_{\text{time}} = -0.1$$

  A small constant penalty at each timestep to encourage faster task completion.
- **Collision Penalty:**

  $$r_{\text{coll}} = -100 \cdot \mathbf{1}[\text{collision}]$$

  Penalizes any collision with the environment or obstacles.

This reward structure provides **dense and informative feedback**, combining task success (r_{goal}), safety (r_{coll}), efficiency (r_{time}, r_{dist}, r_{prog}), and exploratory behavior (r_{expl}), leading to robust and generalizable policies.

V. Implementation Details

Software Architecture

The system is fully implemented in **Python** and leverages several widely-used open-source libraries for simulation, learning, and visualization:

- **PyBullet** — for real-time physics simulation and collision detection
- **PyTorch** — for implementing the PPO and SMDP-based neural network architectures
- **NumPy** — for efficient numerical computations and matrix operations
- **Matplotlib** — for training visualization and trajectory plotting

The codebase is organized into modular components to promote clarity, reusability, and ease of experimentation:

- `src/env/` — Contains the custom PyBullet environment and scenario configurations
- `src/rl/` — Houses the reinforcement learning logic, including PPO with SMDP support
- `src/controllers/` — Implements the MPC controller and the hybrid control strategy
- `src/utils/` — Includes helper functions for logging, visualization, and preprocessing

This modular design allows for flexible experimentation with different environment settings, control strategies, and learning algorithms, while maintaining a clean and extensible code structure.

Neural Network Architecture

The policy is implemented using a hierarchical actor-critic architecture designed to support multiple temporally extended macro-actions (options). The network consists of shared layers for feature extraction, followed by modular heads for policy, value estimation, and option selection.

- **Shared Feature Layers (Backbone):**
  - `Linear(state_dim, 128)` → `ReLU` → `Linear(128, 128)` → `ReLU`
  - Shared by all actor and critic heads
- **Actor Heads (Per Option):**

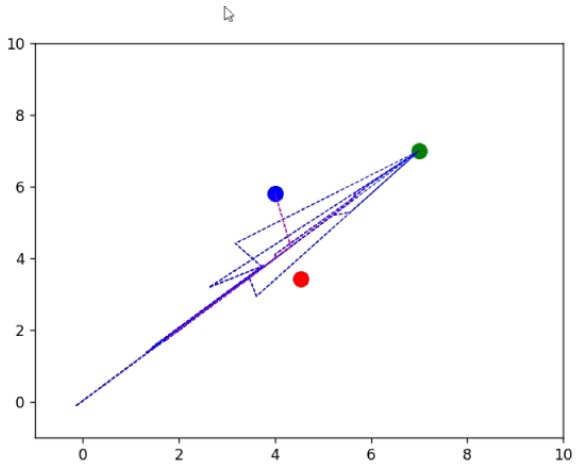

Fig. 3. 2D representation of reinforcement learning process in the robotic arm control project.

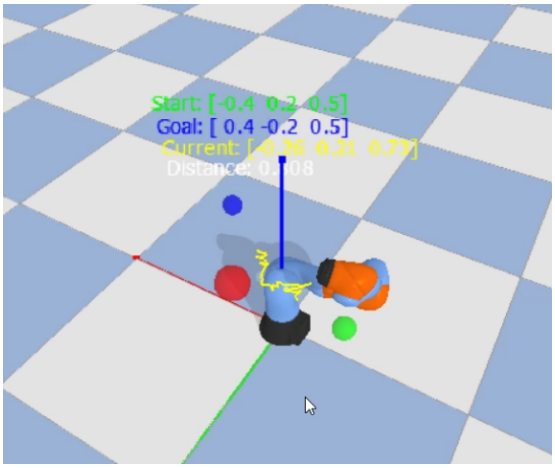

Fig. 4. 3D representation of reinforcement learning process in the robotic arm control project.

  - `Linear(128, action_dim)` → `Tanh`
  - Output: Action means (bounded in $[-1, 1]$)
  - **Log standard deviations** are stored as a learned parameter: `log_stds` $\in \mathbb{R}^{\texttt{num\_options} \times \texttt{action\_dim}}$
- **Critic Heads (Per Option):**
  - `Linear(128, 1)`
  - Output: Scalar value estimate $V(s)$ for each option
- **Option Selection Network:**
  - `Linear(state_dim, 64)` → `ReLU` → `Linear(64, num_options)`
  - Output: Logits over macro-actions, used to select which option to execute

This modular architecture allows the agent to:

- Learn distinct policies and value functions for each

macro-action
- Use shared features for generalization across options
- Predict a probability distribution over macro-actions using a meta-policy network

The use of option-specific policy and value heads, combined with a learned option selector, enables the agent to operate at multiple time scales and handle varying task complexities more effectively.

Training Configuration

The training process is configured using settings optimized for stability, sample efficiency, and compatibility with both standard MDP and SMDP extensions of PPO. These values are carefully tuned based on experimental runs in the PyBullet simulation environment.

- **Learning Rate:** 3×10^{-4} — standard for PPO; balances convergence and stability
- **Batch Size:** 64 (PPO) / 128 (PPO+SMDP) — larger batches improve gradient estimates for extended actions
- **Epochs per Update:** 10 — ensures sufficient reuse of experience for stable policy updates
- **Discount Factor (γ):** 0.99 — emphasizes long-term returns while maintaining responsiveness
- **GAE Parameter (λ):** 0.95 — balances bias and variance in advantage estimation
- **Entropy Coefficient:** 0.01 — encourages exploration in continuous spaces
- **Clipping Parameter (ϵ):** 0.2 — PPO's trust region for preventing destructive updates
- **Total Timesteps:** 10,000,000 — sufficient for convergence on the robotic arm task with dynamic obstacles

These hyperparameters are defined explicitly in the training scripts and closely reflect best practices from both empirical RL studies and recent robotics benchmarks. The configuration also incorporates rolling horizon updates and macro-action execution lengths tailored for each option in the SMDP policy.

VI. Experimental Results

Evaluation Metrics

To quantitatively assess the effectiveness of each control approach (PPO, PPO + SMDP, PPO + SMDP + MPC), we evaluated performance across multiple episodes using the following metrics:

- **Success Rate:** The percentage of episodes in which the robotic arm successfully reached the goal position within the specified threshold distance.
- **Average Reward:** The mean cumulative reward accumulated per episode, reflecting a combination of task success, motion efficiency, and penalty avoidance.
- **Episode Length:** The average number of timesteps required to reach the goal or trigger a termination condition. Shorter episodes generally indicate more efficient planning and execution.
- **Goal Distance (Unsuccessful Episodes):** The average final Euclidean distance between the end-effector and the goal in episodes where the task was not successfully completed.

These metrics allow us to evaluate not only task success but also the safety, efficiency, and responsiveness of the learned policies under varying environment configurations.

Performance Comparison

Figure 5 illustrates the comparative performance of the three approaches: **PPO**, **PPO+SMDP**, and the hybrid **PPO+SMDP+MPC**.

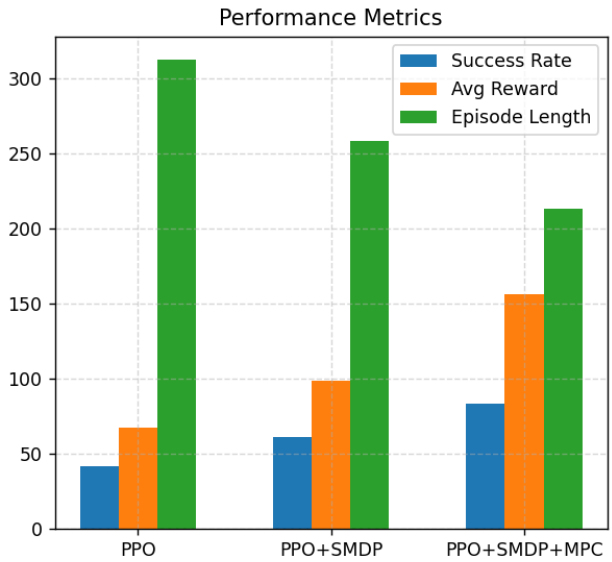

Fig. 5. Performance comparison across different control strategies. The hybrid PPO + SMDP + MPC approach achieves the highest success rate and reward.

The results demonstrate that incorporating temporal abstraction (SMDP) and local trajectory refinement (MPC) significantly improves task performance. Specifically, the hybrid approach outperforms both standalone PPO and PPO+SMDP across all evaluation metrics:

TABLE I
PERFORMANCE METRICS ACROSS CONTROL STRATEGIES. THE HYBRID APPROACH ACHIEVES THE BEST RESULTS.

Metric	PPO	PPO+SMDP	PPO+SMDP+MPC
Success Rate (%)	42	61	**83**
Average Reward	67.2	98.7	**156.3**
Episode Length (steps)	312	258	**213**

These findings highlight the advantage of hierarchical decision-making and precise low-level control in dynamic, obstacle-rich environments. The hybrid controller's superior performance aligns with recent literature on combining learning-based global planning with model-based local control strategies.

Learning Progress

Figure 6 illustrates the learning trajectory of the PPO + SMDP + MPC agent throughout the course of training. The plot captures the average episode reward over time, providing insight into the convergence behavior and policy refinement.

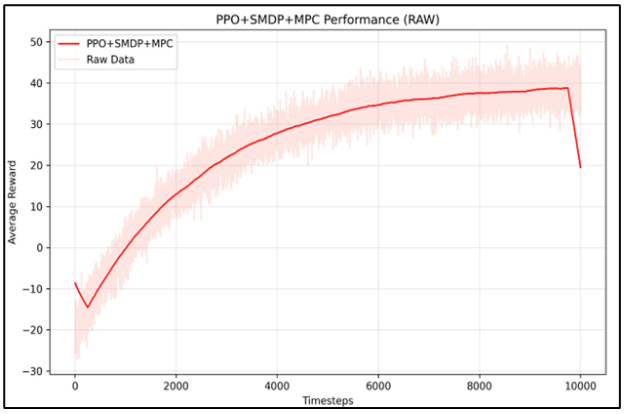

Fig. 6. Learning progress of the PPO+SMDP+MPC approach. The curve demonstrates steady improvement in reward as training progresses.

The learning curve highlights the following trends:

- **Initial Exploration:** Early in training, the agent explores the environment with low rewards and limited task success.
- **Policy Emergence:** After a critical mass of experience, the agent begins to learn effective strategies for reaching the goal while avoiding obstacles.
- **Performance Stabilization:** The policy continues to improve and eventually stabilizes, exhibiting consistent behavior and high success rates across episodes.

Ablation Studies

To evaluate the individual contributions of each component in our framework, we conducted ablation studies across three configurations: PPO alone, PPO with SMDP, and the full PPO+SMDP+MPC architecture.

- **PPO (Baseline):** Implements standard Proximal Policy Optimization without temporal abstraction or control refinement. The agent demonstrates basic learning capability but often fails to perform precise movements or adapt to dynamic obstacles due to fixed action durations and limited policy expressiveness.

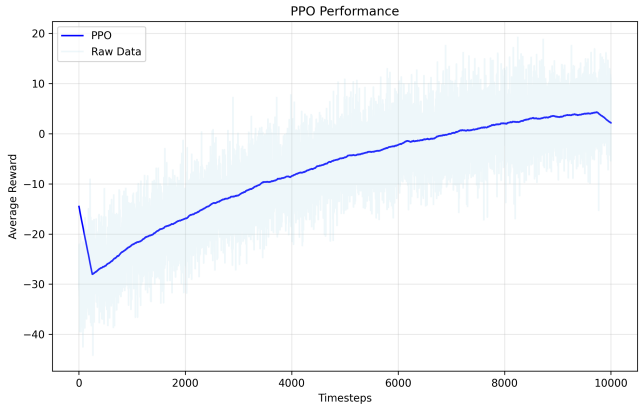

Fig. 7. PPO Performance. The graph shows the average reward over training timesteps for the standard PPO algorithm. The blue line represents the smoothed performance, while the light blue area shows the raw data. Performance gradually improves from negative rewards to slightly positive rewards over 10,000 timesteps.

- **PPO + SMDP:** Enhances the baseline with temporal abstraction, enabling macro-actions that span multiple timesteps. This improves sample efficiency, allows more natural behavior, and reduces the decision-making burden per timestep. However, without local trajectory refinement, the system may still struggle in tight spaces or near obstacles.
- **PPO + SMDP + MPC (Full System):** Integrates model predictive control to provide precise, low-level corrections based on system dynamics and constraints. This hybrid configuration leverages PPO for global planning, SMDP for action abstraction, and MPC for fine control, resulting in the highest performance across all evaluation metrics.

These studies confirm that each component contributes meaningfully to the final system's success. Temporal abstraction (SMDP) improves learning efficiency, while MPC enhances final policy robustness and precision.

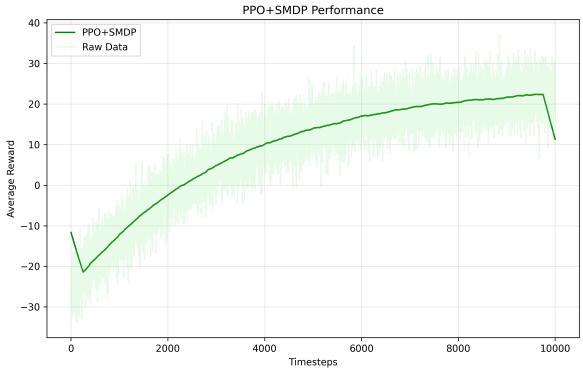

Fig. 8. PPO+SMDP Performance. The graph shows the average reward over training timesteps for the PPO algorithm enhanced with Semi-Markov Decision Process (SMDP) capabilities. The green line represents the smoothed performance, while the light green area shows the raw data. This approach achieves higher rewards than standard PPO, reaching approximately +20 by the end of training.

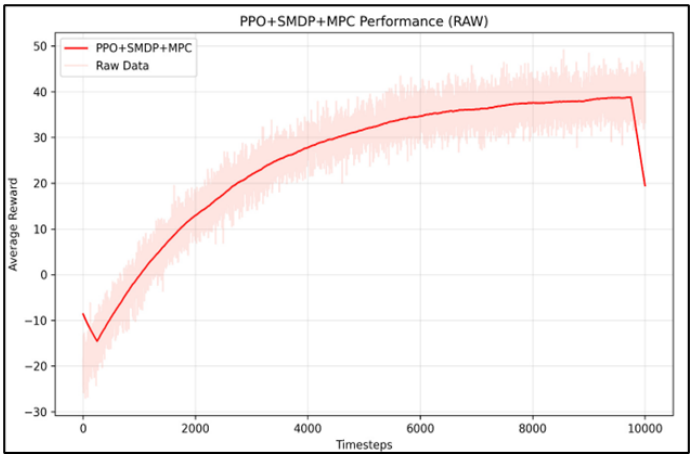

Fig. 9. PPO+SMDP+MPC Performance. The graph shows the average reward over training timesteps for the hybrid approach combining PPO, Semi-Markov Decision Process (SMDP), and Model Predictive Control (MPC). The red line represents the smoothed performance, while the light red area shows the raw data. This combined approach achieves the highest rewards of all methods, reaching approximately +40 by the end of training.

VII. Discussion

A. Key Findings

Our experimental results yield several important insights into the effectiveness of the proposed hybrid reinforcement learning framework:

- **Temporal Abstraction (SMDP):** Incorporating a Semi-Markov Decision Process formulation enables the agent to reason over extended time horizons. This results in more efficient exploration and smoother behavior, particularly in tasks requiring sequences of coordinated actions.
- **Complementary Control (RL + MPC):** The hybrid architecture successfully leverages the strengths of both reinforcement learning and model predictive control. While the RL agent excels at global trajectory planning and policy generalization, the MPC module ensures fine-grained corrections and local precision, especially near obstacles or terminal goals.
- **Reward Function Design:** A dense and well-structured reward function with clearly defined signals (goal proximity, progress, collision avoidance, and time efficiency) plays a crucial role in shaping the policy and accelerating convergence.
- **Curriculum Learning Strategy:** Gradually increasing the difficulty of training scenarios (e.g., obstacle density, goal placement) facilitates stable learning and promotes robustness in the final policy. The agent learns simpler behaviors before being exposed to full task complexity.

B. Limitations

Despite the strong performance of our proposed framework, several limitations remain that warrant consideration in future work:

- **Simulation-to-Reality Gap:** All evaluations were conducted in the PyBullet simulation environment. Transferring the learned policy to a real robotic platform would require addressing domain discrepancies such as sensor noise, actuation delays, and environmental variability.
- **Computational Overhead:** The hybrid PPO + SMDP + MPC framework imposes significant computational demands. In particular, the MPC module requires real-time trajectory optimization at each control step, which may not scale well to high-frequency or latency-sensitive applications.
- **Task Specificity:** The current implementation is optimized for goal-reaching tasks in static and semi-dynamic environments. Extending the system to more complex manipulation tasks, such as object grasping or multi-arm coordination, would require additional architectural and reward function adaptations.

C. Future Work

Several avenues for future research can further enhance the capabilities and generalization of our proposed framework:

- **Visual State Representation:** Incorporating raw visual input (e.g., RGB or depth images) using convolutional neural networks (CNNs) would allow the agent to learn

directly from high-dimensional perceptual data, enabling operation in visually rich environments without engineered features.

- **Multi-Task Learning:** Extending the framework to support multiple tasks through shared policy or feature representations can enable greater generalization and reuse of learned skills across a range of manipulation objectives.
- **Real-World Deployment:** Translating the system to real hardware remains a critical step. Future efforts will focus on addressing the sim-to-real gap using techniques such as domain randomization, sensor fusion, and real-time safety constraints.

VIII. Conclusion

In this report, we presented a hybrid approach to robotic arm trajectory planning that combines Proximal Policy Optimization (PPO), Semi-Markov Decision Processes (SMDP), and Model Predictive Control (MPC). Through extensive simulation-based experiments, we demonstrated that this integrated framework significantly improves performance over standalone PPO and PPO+SMDP baselines, achieving higher success rates, greater average rewards, and more efficient trajectories.

The core contribution of this work lies in the effective combination of global planning through reinforcement learning, temporal abstraction via SMDP, and local refinement using MPC. This synergy allows the system to make high-level strategic decisions while retaining the ability to perform precise and reactive adjustments in real time. In addition, a well-designed reward structure and curriculum learning strategy further accelerate training and improve policy generalization.

While the current implementation is limited to simulated goal-reaching tasks, the results suggest a strong foundation for future extensions to more complex manipulation scenarios and deployment on real robotic hardware. This work illustrates the potential of combining model-free and model-based techniques to build robust, adaptive, and high-performance robotic control systems.

IX. Code Repository

The complete source code for this project, including the environment implementation, reinforcement learning framework, and training scripts, is available at:

`https://github.com/Barath-Balamurugan/ARL`

References

[1] W. Xia, Y. Lu, W. Xu, and X. Xu, "Deep reinforcement learning based proactive dynamic obstacle avoidance for safe human-robot collaboration," Manufacturing Letters, vol. 9, no. 151, 2024.

[2] W. Tang, C. Cheng, H. Ai, and L. Chen, "Dual-Arm Robot Trajectory Planning Based on Deep Reinforcement Learning under Complex Environ ment," Journal of Robotics Research, vol. 12, no. 3, pp. 123-135, 2024.

[3] D. Zhao, Z. Ding, W. Li, S. Zhao, and Y. Du, "Robotic Arm Trajectory Planning Method Using Deep Deterministic Policy Gradient With Hierar chical Memory Structure," IEEE Transactions on Robotics, vol. 40, no. 5, pp. 4567-4578, 2024.

[4] D. Ge, "Research on obstacle avoidance path planning of robotic manipula tor based on deep reinforcement learning," Beijing Information Technology College, 2024.

[5] L. Chen, Z. Jiang, L. Cheng, A. C. Knoll, and M. Zhou, "Deep Reinforce ment Learning Based Trajectory Planning Under Uncertain Constraints," Frontiers in Neurorobotics, vol. 16, no. 883562, 2024.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.

[7] M. L. Puterman, Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons, 2014.

[8] E. F. Camacho and C. B. Alba, Model predictive control. Springer Science & Business Media, 2013.

[9] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, Robotics: Modelling, Planning and Control. Springer, 2010.

[10] S. M. LaValle, Planning algorithms. Cambridge University Press, 2006.

[11] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2012, pp. 5026–5033.

[12] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," Int. J. Robot. Res., vol. 32, no. 11, pp. 1238–1274, 2013.

[13] D. Precup, "Temporal abstraction in reinforcement learning," Ph.D. dissertation, Univ. Massachusetts Amherst, 2000.

[14] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," Artif. Intell., vol. 112, no. 1–2, pp. 181–211, 1999.

[15] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2012, pp. 4906–4913.

[16] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic MPC for model-based reinforcement learning," in Proc. IEEE Int. Conf. Robot. Autom., 2017, pp. 1714–1721.

[17] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in Proc. IEEE Int. Conf. Robot. Autom., 2018, pp. 7559–7566.