Exploring Learning for Intercepting Projectiles with a Robot-Held Stick

John E. G. Baxter[1], Torin Adamson[1], Satomi Sugaya[1] and Lydia Tapia[1]

***Abstract*— For many tasks, including table tennis, catching, and sword fighting, a critical step is intercepting the incoming object with a robot arm or held tool. Solutions to robot arm interception via learning, specifically reinforcement learning (RL), have become prevalent, as they provide robust solutions to the robot arm interception problem, even for high degree of freedom robotic systems. Despite numerous solutions, there has been little exploration into the factors of learning that impact solution quality. Thus, there is little insight into what problem features lead to better learning success. In this paper, we explore the parameters that impact solution quality. We find that link position observations outperform joint angle observations in terms of learning speed, performance, ability to utilize more than one frame of observation, and generalization to situations not trained for. These results are immediately applicable to RL for robot arm interception tasks.**

I. Introduction

Interception of an object is a key problem in robotics, and many important tasks rely upon successful interception. For example, interception may be involved in catching [1]–[7], batting [8]–[20], parrying [21]–[25], or otherwise controlling an object through robotic motion [26], [27]. Interception is difficult to perform, as it combines inverse kinematics and prediction of dynamic projectiles, which in turn must rely on potentially inaccurate sensor data. Deep RL [28] is one approach used to train algorithms to handle such difficult systems, and feature selection has a critical effect on learning. This work evaluates the impact of using different representations of robot configuration as observation feature choices, and the influence of these features on the performance of the resulting interception algorithm.

Interception of projectiles, and other physical objects, by robotic means has been considered before in several contexts. For moving objects, inverse kinematics alone is not sufficient, as the target interception point changes over time. A model of the object's motion is required to calculate the future interception point, but creating accurate models is difficult for real world objects. As an alternative approach, machine learning can find a direct mapping from observations of obstacles to robot motions, a policy, incorporating real world dynamics into the solution [13], [15]–[21], [24], [27]. As we show in this work, input feature choice can greatly influence

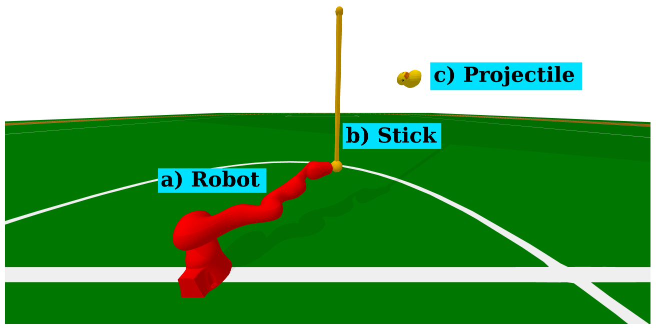


Fig. 1: The simulation setup used in this work's interception learning. A fixed base 7-DOF KUKA robot arm, a) Robot, is affixed with a stick, b) Stick. Projectiles, c) Projectile, are thrown at the robot one at a time. A deep RL network controls the robot joints by observing the robot and projectile and setting joint accelerations. The effects of varying the type of observations are examined in this work. The policy receives reward if the stick hits the projectile, and penalty if the projectile hits the robot or ground, or if the stick hits the robot arm.

learned performance in terms of success rate of interception. However, a study on these input feature choices for RL-based interception learning, surprisingly, does not to the best of our knowledge exist in the literature.

Through an analysis of the effect of selecting between joint angles ('Joints') or link positions ('Links') as features for an RL-based interception task, we contribute quantitative measures on the effect of feature choice on policy performance. Figure 1 shows the interception simulation setup, in which a robot with a stick attached to an end effector must hit a projectile with the stick without the projectile hitting the robot or the ground, or the robot hitting itself with the stick. To test the robustness of the resulting systems to situations not encountered in training, the policies are subjected to variations of horizontal projectile speeds, projectile position observation noise, and perturbed object trajectories. We find that not only can feature choice improve training rates, it can also produce policies more resilient to untrained-for observation noise and projectile dynamics. We also release simulation and learning code, as well as the trained policies for use with that code on our website. A video with demonstrations is attached.

II. Related Work

All interception approaches we found in the literature rely on a pipeline which takes observation data, performs some processing, and produces motor commands for the robot. We

*Tapia, Adamson, and Sugaya are partially supported by the National Science Foundation under Grant Number IIS-1553266. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

[1]John E. G. Baxter automata@unm.edu, Torin Adamson toriadam@unm.edu, Satomi Sugaya satomi@cs.unm.edu and (corresponding) Lydia Tapia tapia@cs.unm.edu are with the Department of Computer Science, University of New Mexico, Albuquerque, NM 87131, USA http://www.tapialab.science

will distinguish between three broad interception pipeline categories. First, there are pure expert systems which do not perform any learning. Second, systems may use supervised learning to learn the motions of the object to be intercepted, learn how to position the robot to intercept, and/or learn to predict the often complicated behavior that occurs in response to the impact of interception. Finally, interception may be handled by RL, in which a function transforming observations to actions is learned by optimizing for reward.

There are many interception problems that have been addressed by expert systems. An early system for table tennis used a complex model of ball trajectory and calculated motor controls and higher level decisions from stereo observations [8], while a later work used a racket rebound model together with an aerodynamics model to calculate how to strike balls to hit chosen target locations [14]. Faster, simpler ball models have been compensated for with high speed vision and manipulator speeds, such as one that uses joints with high inertia to move the arm into general position while faster joints continuously track predictions of ball interception points [11]. Expert systems have also been used for catching, creating direct mappings between ball trajectory observations and robot trajectory generation [1], catching with minimal impact [6] or minimal robot energy expenditure [3], and catching without prehensile grasp [2]. Interesting expert systems for robot sword fighting skills have also been developed, which detect a human's attack and generate parries by inverse kinematics [22], [23]. Despite successes, expert systems have difficultly adjusting themselves to unexpected deviations due to reliance on accurate models of the motion of the robot and the object of interception. Learning algorithms, by contrast, are well suited to addressing these issues.

Supervised learning has been used to extrapolate the trajectories of objects to be intercepted, generate interception actions, and predict the behavior of objects after interception. In table tennis, one work implemented a mapping between ball observations and the ball hitting plane, encoding training data with a KD-tree [9], while in a later work a locally weighted regression was used to learn ball behavior, and iterative learning control used to achieve desired stroke motions [12]. Another table tennis work considered learning how likely a robot is able to successfully land an incoming ball at various parts of the table [10]. Learning has also been applied to robot catching, considering the problems of catching non-spherical thrown objects [5], catching multiple balls with a flexible link robot [4], and early anticipation of ball trajectory before it leaves the thrower's hand [7]. In another instance of interception learning, a system of multiple neural networks handled the task of a virtual character performing sword blocks, learning from motion capture data [25]. Although supervised learning can be a straightforward and successful, it does require a lot of data be collected in order to train, which may be difficult to acquire. By contrast, RL can teach itself, collecting training data in the process of performing its task.

RL, the type of learning considered in this paper, optimizes a policy incrementally based on reward and punishment feedback. Deep RL has used self-play to learn policies for dynamically complex interception tasks with multiple humanoid agents, including soccer goal kicking and defending [27]. In table tennis, RL has been used to learn when enough observations have been made and the time to act [13], how to weigh motion primitives for generalizing human strokes [15], and adjust robot trajectories to better return the ball [17]. Hierarchical RL systems for learning how to play table tennis have been developed [19], as well as RL algorithms for learning table tennis without a model of ball trajectory [20]. In virtual sword fighting RL, simulated defenders with sticks have been trained to defend against computer generated stick attacks [21] and recorded human stick swings [24].

The problem setup most similar to ours is in [24], which uses RL to control an arm intercepting free-flying objects. There, a virtual character's arm with four actively controlled degrees of freedom moved by discrete joint velocities was used to intercept recorded human stick swing motions, and focused on the development of the interception method. In this work we consider a robot arm with seven degrees of freedom moved by discrete joint accelerations to intercept randomly generated projectiles moving under the effect of gravity, and focus on examining the effects of observation feature type on learning and on resilience to situations not encountered in training.

III. Preliminaries

In this section we present the interception learning problem, detail the network architecture, give details of the simulation used for training and evaluation, and explain episode initialization.

A. Learning Problem

Our deep RL problem is formalized as a Partially Observable Markov Decision Process (POMDP), given by the tuple (S, O, A, T, R, γ). S is the simulation state space. This includes the global Cartesian coordinates of the projectile with origin at the robot base, the orientation of the projectile, the linear and angular velocities of the projectile, and the joint angle positions and velocities of the robot arm. O is the observation space, which varies according to experimental setup, to be described in further detail in Subsection IV-A. In short, we consider robot observations as joint angles, link positions, or both, and provide one or two time steps of observation. A is the action space, one discrete acceleration value (1, 0 or -1 radians per second squared) per joint. In preliminary tests, using discrete velocity actions in place of discrete acceleration actions lead to poor exploration of the state space (specifically the robot would jitter mostly in place, failing to ever encounter projectiles from most directions). T is the transition function from a state and action pair to a new state, here this function is the PyBullet physics simulation. R is the reward function, which is 0 on all non-terminal time steps. There are four terminal events: *stick hit projectile*, *projectile hit robot*, *stick hit robot*, and *projectile hit ground*. We give a +1 reward on a *stick hit projectile* terminal event, and -1 rewards on the other

Total Time Steps	10^9
Learning Rate	5×10^{-5}
Gamma (Discount Factor)	0.99
GAE Lambda (Smoothing Parameter)	1
Initial KL Coefficient	0.2
Rollout Fragment Length	200
Training Batch Size	4000
SGD Minibatch Size	128
Shuffle Sequences	True
Number of SGD Iterations	30
Entropy Coefficient	0
PPO Clip Parameter	0.3
Value Function Clip Parameter	10
KL Target	0.01
Batch Mode	Truncate Episodes
Number of CPUs	5
Environments per CPU	50
Number of GPUs	0

TABLE I: Network parameters for our fully connected actor-critic RL networks. Optimization was performed by the PPO algorithm. In training 5 CPUs with 50 simulation environments each were used.

terminal events. An episode consists of any number of non-terminal time steps followed by a terminal event time step. Finally, $\gamma \in [0, 1]$ is the discount factor which determines preference for immediate versus long term reward, with larger γ indicating higher preference for long term reward.

B. Network Implementation

A deep RL network is implemented using the RLlib [29] RL library. Two parallel fully connected networks (actor and critic), with 4 hidden layers each with 256 neurons per hidden layer and ReLU activation for all but the output layer, are used, one (the actor) producing output features, and the other (the critic) outputting the value estimate. We use the Proximal Policy Optimization (PPO) [30] with a Generalized Advantage Estimator (GAE) [31] for policy optimization. Additional network parameters are given in Table I. The network learns a direct mapping from observations taken every time step to joint accelerations applied to the robot's joints every time step.

C. Simulation Details

PyBullet [32] is used for physics simulation. We use three models that come with PyBullet: a KUKA model for the robot arm, a scaled rubber duck model for the projectile, and a flat soccer field for the ground. We modeled the stick as a thin cylinder with spheres at each end. The stick is attached to the end effector of the robot arm. A time step of 0.01 seconds is used. Gravity in the downward direction of magnitude 9.8 m/s^2 is applied to both projectile and robot.

D. Episode Initialization

The robot arm is initialized at the start of each episode as follows. Joints are enumerated from 0 to 6, with 0 being the closest to the base along the kinematic chain. Joint 0 of the arm is set to a joint angle uniformly at random in the valid joint range, joint 1 is set to $-90°$, and all other joints are set to $0°$. This corresponds with an arm that is outstretched in a random valid direction, with the stick held upward perpendicular to the ground.

The projectile is initialized at the start of each episode as follows. The projectile is positioned 1 m above the ground and 2 m away from the robot in a random horizontal direction. The horizontal velocity of the projectile has a magnitude H_{Speed} and is directed toward the robot, this magnitude is equal to 3 m/s unless noted otherwise. The initial vertical velocity of the projectile is a uniform random value in the range of $[0, 2.5]$ m/s in the upward direction. The projectile's orientation is chosen uniformly at random. An angular velocity chosen uniformly in $[-10\pi, 10\pi]$ radians/s is applied to each axis.

IV. Methods

In order to evaluate the effect of observation type on learned interception performance, we define a set of parameters that are varied during training and post-training evaluation, perform a series of evaluations, and use metrics to assess performance of the interception task. We also detail the hardware used to run the evaluations.

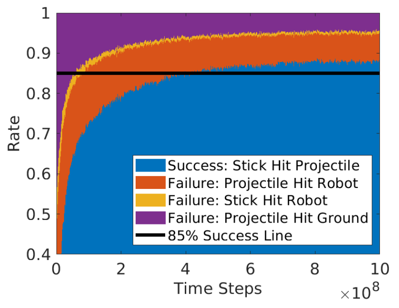
A. Varied Training Parameters

Input features (belonging to a observation space) are presented to the policy as a flat vector concatenation of floating point values. We logically divide input features into frames. Each frame contains the observations of one simulation time step. A frame consists of floating point data for a robot observation, a projectile observation, and an indicator of whether the projectile has been hit by the stick. We vary the robot observation type, to be described in the following paragraph. All projectile observations are global Cartesian coordinates of the center of the projectile at the observed time step. The indicator of whether the projectile has been hit by the stick is 0.0 except on a terminal time step where the stick hit the projectile, in which case it is 1.0. The indicator exists for a generalization to multiple projectile interception, not considered in this work.

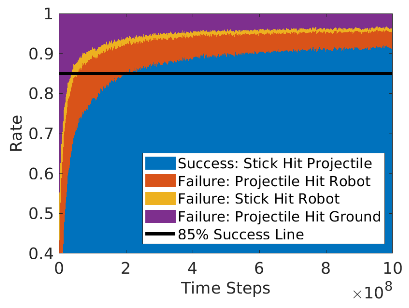
We consider three robot observation types: Joints, Links, and Both. The Joints type consists of 7 float joint angles corresponding to the 7 degrees of freedom of the robot. The Links type consists of 30 floats, corresponding to 10 links positions with 3 global Cartesian coordinates each. The Both type consists of the concatenation of the Joints and Links observations. We consider two frame counts: 1 Frame, where the input feature is based on the most recent simulation state, and 2 Frames, where the input feature is the concatenation of the observations of the previous and current simulation time steps. The previous frame of 2 Frames is initialized to the values of the current frame at time step 0.

B. Evaluations Performed

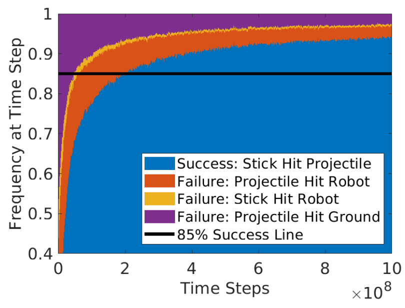
- **Training Evaluation:** For each robot observation type (Joints, Links, or Both) and frame count (1 Frame or 2 Frames) combination, we train 5 replicate policies for one billion time steps, recording evaluation metrics periodically. The purpose of this evaluation is to analyze



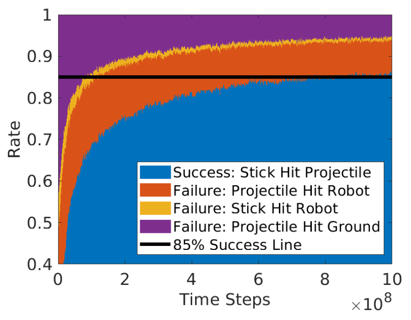
(a) Joints, 1 Frame



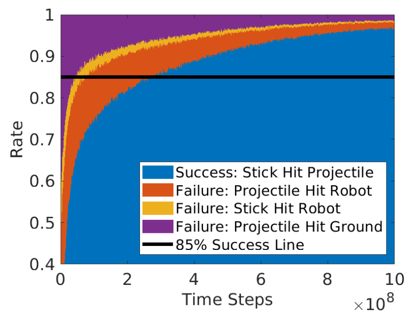
(b) Links, 1 Frame



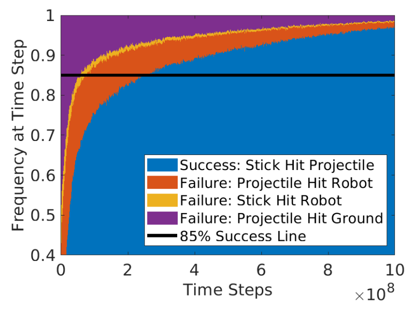
(c) Both, 1 Frame



(d) Joints, 2 Frames



(e) Links, 2 Frames



(f) Both, 2 Frames

Fig. 2: Training Evaluation, Success and Failure Rates. This figure presents success and failure rates of the six observation types considered in this paper over the course of RL training.

Final Events	Joints, 1 Frame	Links, 1 Frame	Both, 1 Frame	Joints, 2 Frames	Links, 2 Frames	Both, 2 Frames
Stick Hit Projectile	$88.8\% \pm 1.9\%$	$92.0\% \pm 1.0\%$	$94.6\% \pm 1.1\%$	$86.2\% \pm 2.6\%$	$96.9\% \pm 1.5\%$	$96.8\% \pm 0.6\%$
Projectile Hit Robot	$6.3\% \pm 1.4\%$	$4.0\% \pm 0.7\%$	$2.7\% \pm 0.6\%$	$7.8\% \pm 0.7\%$	$1.3\% \pm 0.9\%$	$1.6\% \pm 0.3\%$
Stick Hit Robot	$0.8\% \pm 0.2\%$	$0.8\% \pm 0.2\%$	$0.5\% \pm 0.3\%$	$0.6\% \pm 0.4\%$	$0.5\% \pm 0.1\%$	$0.4\% \pm 0.2\%$
Projectile Hit Ground	$4.1\% \pm 0.8\%$	$3.3\% \pm 0.5\%$	$2.3\% \pm 0.5\%$	$5.3\% \pm 1.7\%$	$1.4\% \pm 0.6\%$	$1.3\% \pm 0.3\%$
Steps, Success $\geq$ 85%	368.85M	198.55M	193.15M	609.25M	269.5M	244.25M

TABLE II: Training Evaluation, Statistics. This table shows mean and standard deviations for success and failure rates for the six observation types considered. It also shows the mean number of time steps to reach 85% success rate. Statistics are over 5 runs for each observation type.

learning rates and long-run performance of different robot observation types with different frame counts.

- **Post-Training Evaluations:** For each trained policy, we run without further training one hundred thousand time steps for a baseline (no simulation parameters changed from training) and situations not experienced in training, recording evaluation metrics.
  - **Projectile Horizontal Speed Variation:** We vary the horizontal speed of the projectile (abbreviated H_{Speed}). In training and for the baseline, H_{Speed} is fixed at 3 m/s. We examine the effect of instead varying H_{Speed} uniformly at random in the ranges [2, 4] m/s and [1, 5] m/s. This is done to assess the learned policies' generality to parabolic trajectories not encountered during training.
  - **Noisy Projectile Observations:** Here we apply, at every time step, Gaussian noise of zero mean to the global X, Y and Z coordinates of the observation of the projectile's true position. The baseline used in training is no observational noise. To explore the policies' resilience to untrained-for noisy observations we evaluate the trained policies with 0.05 m and 0.25 m standard deviation Gaussian noise applied to the projectile position observation.
  - **Non-Parabolic Projectile Trajectories:** We wish to evaluate the trained policies' generality to fundamentally different (non-parabolic) trajectories than the ones for which they were trained. We do this by applying Gaussian perturbations with zero mean to the projectile's X, Y and Z components of linear velocity every time step. The baseline is a parabolic trajectory as encountered in training. We consider 0.05 m/s and 0.25 m/s standard deviation perturbations to projectile linear velocity.

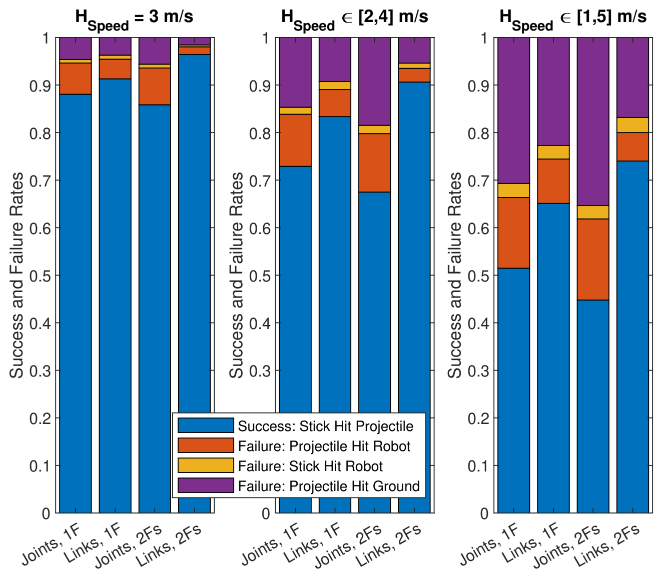


Fig. 3: Projectile Horizontal Speed Variation Evaluation. Stacked bar plots show mean success and failure rates under the effect of different H_{Speed} ranges. Here '1F' is short for '1 Frame' and 2Fs is short for '2 Frames'.

Success Rate Ratio: H_{Speed} Variation / Baseline		
	$H_{Speed} \in [2,4]\ m/s$	$H_{Speed} \in [1,5]\ m/s$
Joints, 1F	$82.77\% \pm 3.19\%$	$58.42\% \pm 5.10\%$
Links, 1F	$91.30\% \pm 0.61\%$	$71.30\% \pm 1.81\%$
Both, 1F	$90.69\% \pm 1.68\%$	$70.51\% \pm 2.56\%$
Joints, 2Fs	$78.62\% \pm 0.82\%$	$52.17\% \pm 1.34\%$
Links, 2Fs	$93.96 \pm 1.13\%$	$76.70\% \pm 4.17\%$
Both, 2Fs	$93.24\% \pm 1.79\%$	$74.97\% \pm 2.14\%$

TABLE III: Projectile Horizontal Speed Variation Evaluation, Statistics. This table shows mean and standard deviation of the success ratio of an untrained-for situation to the baseline.

C. Evaluation Metrics

In order to assess interception performance we consider evaluation metrics related to the frequency of success and the three different kinds of failures. *Success: Stick Hit Projectile* means the stick successfully intercepted the projectile before a failure case could occur. *Failure: Projectile Hit Robot* means the projectile touched the robot arm before the projectile was intercepted by the stick. *Failure: Stick Hit Robot* means the stick touched the robot arm before the projectile was intercepted by the stick. *Failure: Projectile Hit Ground* means the projectile touched the ground before the projectile was intercepted by the stick.

D. Hardware and Software Used

The training evaluation was run on a machine with four Intel Xeon Gold 6148 Processors with 20 cores each (2.40 GHz) with 512 GB of RAM running Ubuntu 18.04.3 LTS. Post-training evaluations were run on a laptop with an AMD Ryzen 7 Pro 3700U Processor (2.30 GHz, 4 Cores, 8 Threads, 4 MB Cache) and 8 GB DDR4 2400MHz RAM running Ubuntu 20.04.2 LTS. RL networks were implemented using RLlib 1.0.0 [29]. The robot-projectile system was simulated in PyBullet 3.0.6 [32].

V. Evaluation Results

A. Training Evaluation

The frequency of outcomes as training progresses can be seen in the shaded regions of Figure 2, and the final outcome rates and steps taken to reach a desired success rate of 85% can be found in Table II.

1) Joints versus Links, the 1 Frame Case: For this case, Links outperforms Joints in terms of both learning rate and performance at the end of training (Figure 2 a, b). To reach the 85% threshold, Joints, 1 Frame policies took 358.85 million time steps while Links, 1 Frame policies took only 198.55 million time steps. That is, the Joints policies took 185.8% more time steps to reach the threshold success rate than the Links policies. At the end of training, Joints, 1 Frame policies had a mean success rate of 88.8%, while the Links, 1 Frame policies had mean success rate 92.0%, a difference of 3.2%. Thus, the choice of features used to represent the robot in an interception learning task can influence the rate of learning and the ultimate success of the policy. One possibility is that the learning task requires closing the distance between stick and projectile while keeping the distance of the projectile and stick from the robot body above thresholds. The positions of pieces of the robot body and stick are directly available from to links based policies, while joints based policies must spend additional time and tuned parameters learning a mapping from joints to positions first before calculating an embedded representation of distances.

The failure rates post-training are listed in Table II. The occurrence of failures decrease or stay the same for Links relative to Joints, however, projectile-robot collision events account for 56% of failures for Joints but only 49% for Links (1 Frame). Again, links as features directly represent positions, allowing for easier accounting of distances between the projectile and robot. This could explain the reduction of failures seen here.

2) Joints versus Links, 1 Frame versus 2 Frames: In the case of 2 Frames as input features, the policy is provided velocity information. Moving to 2 Frames helps Links, but is detrimental to Joints. Using 2 Frames increases the time steps to reach an 85% success rate threshold for both robot observation types. Joints, 2 Frames (Figure 2 d) takes 165.2% the time steps taken by Joints, 1 Frame (Figure 2 a), while Links, 2 Frames (Figure 2 e) takes 135.7% the time steps taken by Links, 1 Frame (Figure 2 b). The mean success rate at the end of training decreased for Joints (by 2.6%) but increased for Links (by 4.9%), relative to their one Frame counterparts. It is surprising that Joints, 2 Frames has a lower success rate than Joints, 1 Frame. Likely Joints, 2 Frames has not completely converged in the time step limit. The increase in final success rate for Links type in contrast likely means that the access to velocity information is what allows the Links, 2 Frames setup to find a policy that outperforms the position-only Links, 1 Frame setup.

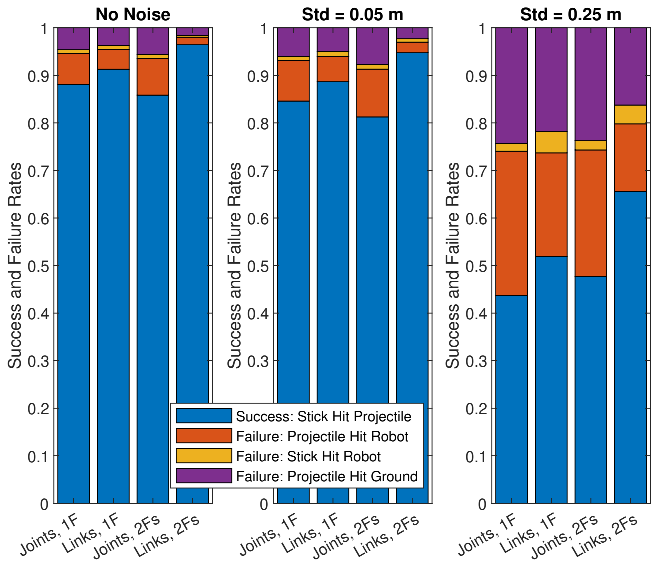


Fig. 4: Noisy Projectile Observations Evaluation. Stacked bar plots show mean success and failure rates under the effect of different amounts of projectile observation noise. Here '1F' is short for '1 Frame' and 2Fs is short for '2 Frames'.

Success Rate Ratio: Noisy Observation / Baseline		
	Std $= 0.05$ m	Std $= 0.25$ m
Joints, 1F	$96.06\% \pm 1.92\%$	$49.70\% \pm 6.37\%$
Links, 1F	$97.09\% \pm 0.78\%$	$56.82\% \pm 3.71\%$
Both, 1F	$98.41\% \pm 0.39\%$	$70.89\% \pm 2.26\%$
Joints, 2Fs	$94.64\% \pm 2.05\%$	$55.48\% \pm 5.91\%$
Links, 2Fs	$98.25\% \pm 0.51\%$	$67.96\% \pm 3.02\%$
Both, 2Fs	$98.57\% \pm 0.36\%$	$77.10\% \pm 3.02\%$

TABLE IV: Noisy Projectile Observations Evaluation, Statistics. This table shows mean and standard deviation of the success ratio of an untrained-for situation to the baseline.

When adding another frame of observation (2 Frames), the failure rates worsen for Joints and improve with Links except for stick-robot collision events using Joints. Projectile-robot collisions are the most affected, with a 1.5% increase for Joints and a 2.8% decrease for Links. This suggests the implied velocity only obtained through Link observations is useful to the policy.

3) Joints, Links, or Both: Here we evaluate the Both observation type. Both, 1 Frame observations outperformed Links, 1 Frame observations. However, Both, 2 Frames performed about equally with Links, 2 Frames (Figure 2 e, f). In terms of success rates, summarised in Table II, the Both observation type with 1 Frame performs better (94.6%) compared to Joints, 1 Frame (88.8%) and Links, 2 Frames (92.0%). For 2 Frames, Links and Both perform about as well (an average success rate difference of 0.1%), however Links, 2 Frames take longer on average to reach an 85% success rate than Both, 2 Frames (269.5 million steps versus 244.25 million steps). Thus, there are advantages to giving both joint angles and link positions as features to a policy.

Combining Joints and Links into one observation type (Both) yields an improvement overall in final success rates, but only for 1 Frame observations (a 2.6% difference in failure rates). For 2 frames, there is only a 0.1% difference in failure rates between Links and Both, with Both failing 0.3% more to robot-projectile collisions but 0.1% less to stick-robot and ground-projectile collisions. The penalty for each failure are weighted equally, so it may be that policies based off Links and Both find different approaches to attain similar success rates when using 2 frames.

B. Post-Training Evaluation

To assess the degree to which trained policies generalize, we evaluate the previously trained policies under noise and dynamics that were not present during training: varying the projectiles' horizontal speed, observed position, and trajectory dynamics. The change in success rate relative to the original policies' performance (without changes) when subject to these variations is recorded to measure robustness. We collect success and failure rates over the course of one hundred thousand time steps for each trained policy, and present the ratios of success in novel situations to baseline.

1) Projectile Horizontal Speed Variation: In Figure 3, we show the success and failure rates for the baseline and two varied horizontal projectile speeds with values in Table III. Links, 2 Frames and Both, 2 Frames retain better performance than the other observation types when variations in speed are introduced. Relative to baseline, policies trained on Links maintain more of their original performance than those trained on Joints. For example, for the $H_{Speed} \in [1, 5]$ m/s case, the Links, 1 Frame policies outperform the Joints, 1 Frame policies by 12.88%. When we compare the relative success rates of 1 Frame and 2 Frames policies, using multiple frames relative to a single frame results in worsened retention of the original performance for Joints but an improved retention of the original performance for Links. Concretely, for $H_{Speed} \in [1, 5]$ m/s, success retention for Joints, 2 Frames is decreased -6.25% relative to Joints, 1 Frame, while Links, 2 Frames increases success retention relative to Links, 1 Frame by 5.4%. Again, the Links features can be directly related to the features of the obstacle in both position and velocity, and it is important to know the direction and speed the obstacle is moving for interception. Therefore, the benefit unique to the Links type networks when given velocity information could be the result of learning both the inverse kinematics of the robot and dynamics for the obstacle. Note that the Both observation types are not significantly different from Links here.

Policies based off Joints have the most significant increases in failures, especially when using 2 frames of observation. Compared to baseline, trials of [1,5] m/s for Joints (2 Frames) have a 30% increase in projectile-ground collisions and a 9% increase in projectile-robot collisions. With Links (2 Frames) these are only 15% and 4%, respectively. Links policies likely learn more generally to close projectile-stick distance while maintaining safe projectile-robot distance, while policies that use joints may prioritize actions for specific situations, which do not generalize as well.

2) Noisy Projectile Observations: It is important for simulation trained policies to be robust to noise, so we consider the effect of adding Gaussian noise with zero mean to each component of the projectile's position observation each time step, see Figure 4 and Table IV. For large noise added to projectile position observations, Both outperforms the other observation types. Compared to baseline, there is only a small effect of adding noise with 0.05 m standard deviation. For 0.25 m standard deviation, however, there is overall low success retention compared to baseline. Policies trained on Links still maintain a higher success rate than the Joints policies, with Joints, 1 Frame's success retention on average 7.12% worse than Links, 1 Frame and Joints, 2 Frames' success retention on average 11.14% worse than Links, 2 Frames. There is better performance of Both observation types than the associated Links type for large noise in projectile position observation. We suspect position uncertainty is reduced using multiple observation types.

For failure cases, all observation types see marked increases in projectile-robot and projectile-ground collision failures under 0.25 m projectile noise, however, these increases are less for links (2 Frames). For links, projectile-robot collisions increase 15% and projectile-ground collisions increase 12%, as opposed to 18% and 19% for Joints. For 2 frames of observation, policies trained on Links seem to be more resilient.

3) Disturbed Projectile Trajectories: So far, we have only considered parabolic projectile trajectories, but not every object in a real environment would have easily predictable dynamics. To see how well the different observation types handle a violation of this property, we add Gaussian perturbations to the components of projectile linear velocity every time step with zero mean. Note, this change means that the policies, which receive at most two frames of state, can not predict exactly where the projectile will be in the future. The success rates relative to baseline are shown in Figure 5 and listed in Table V. With large trajectory disturbances, Links, 2 Frames and Both, 2 Frames outperform the other observation types. With 0.05 m/s perturbations there is not much effect. With 0.25 m/s perturbations, however, there is a drastic increase in the projectile hitting the ground, however the increase is greater for Joints types (by 36.2% for 1 Frame and by 39.52% for 2 Frames) than for Links types (by 20.53% for 1 Frame and by only 14.48% for 2 Frames). Policies trained on Links still maintain a higher success rate than the Joint policies, with Joints, 1 Frame's success ratio on average 19.7% worse than Links, 1 Frame and Joints, 2 Frames' success ratio on average 31.58% worse than Links, 2 Frames. The policies trained on Links observations still outperform the Joints policies when presented with uncertain dynamics and benefit from velocity information (2 Frames) whereas the Joints policies did not. Policies trained on multiple frame observations using link features produce policies that respond immediately to each individual state. Even though the velocity is varied at each time step, the resulting individual states were still familiar to the policy, so the retained success rate is to be expected. The performance

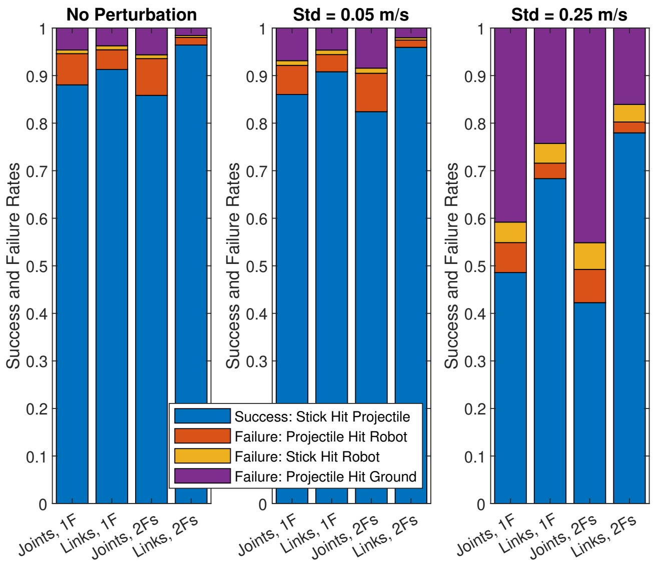


Fig. 5: Disturbed Projectile Trajectory Evaluation. Stacked bar plots show mean success and failure rates under the effect of different disturbances to velocity of the projectile. Here '1F' is short for '1 Frame' and 2Fs is short for '2 Frames'.

Success Rate Ratio: Noisy Projectile Velocity / Baseline		
	Std = 0.05 m/s	Std = 0.25 m/s
Joints, 1F	$97.70\% \pm 1.15\%$	$55.13\% \pm 5.62\%$
Links, 1F	$99.45\% \pm 0.46\%$	$74.83\% \pm 2.49\%$
Both, 1F	$98.88\% \pm 0.43\%$	$73.89\% \pm 2.58\%$
Joints, 2Fs	$96.02\% \pm 1.21\%$	$49.22\% \pm 2.42\%$
Links, 2Fs	$99.49\% \pm 0.75\%$	$80.80\% \pm 3.68\%$
Both, 2Fs	$99.34\% \pm 0.84\%$	$79.68\% \pm 2.93\%$

TABLE V: Disturbed Projectile Trajectory Evaluation, Statistics. This table shows mean and standard deviation of the success ratio of an untrained-for situation to the baseline.

of the Both observation types are in line with the associated Links type for a given degree of trajectory disturbance and number of frames.

For 0.25 m/s perturbations, projectile-ground and stick-robot collision failures are increased compared to no perturbations, while projectile-robot collision failures are not as affected. The worst increases are for Joints, with a 40% increase in projectile-ground collisions compared to only a 14% increase for Links (2 Frames). This again suggests that policies trained on Links may be learning to minimize the distance between the stick and the projectile, while the those trained on Joints are instead memorizing mappings from observations to behaviors, which we would expect to be less resilient to situations that did not occur in training.

VI. Conclusions

Learning how to intercept is a fundamental component of many robotic tasks. In our experiments we find that joint angle observations, a standard choice in robotics, do not perform well compared to link position observations in either learning to intercept or generalizing to untrained-for projectile noise and dynamics. This is perhaps surprising, because it is well known that link positions can be derived from joint angles by learning the forward kinematics. The poor performance of the Joints observation type is perhaps because joint angle feature policies do need to encode computations similar to forward kinematics in order to be able to calculate an embedded representation of distances, whereas link position feature policies don't have to learn forward kinematics and can allocate those saved neurons otherwise. We also note that Links observations are able to better employ temporal information (two time steps of observations) than Joints, maybe because Joints have to learn a form of forward kinematics, whereas Links do not.

Having combined link and joint observations helps in some situations, but not others. Future work should address, for instance, how much Links information must be added to Joints for it to perform better: e.g., is stick position enough to improve performance or is more positional information needed. Future work should also examine if these trends extend to robots with fewer or more DOFs, or different DOF types, and demonstrate interception on real robots. It is also important to determine if these results extend to other interception problems, such as catching, batting or sword fighting. Feature set selection greatly impacts RL performance, and future effort should seek to find more general relations across a variety of interception tasks.

References

[1] A. Namiki and M. Ishikawa, "Robotic catching using a direct mapping from visual information to motor command," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 2, 2003, pp. 2400–2405.

[2] G. Bätz, A. Yaqub, Haiyan Wu, K. Kühnlenz, D. Wollherr, and M. Buss, "Dynamic manipulation: Nonprehensile ball catching," in *Mediterranean Conf. Control Autom. (MED)*, 2010, pp. 365–370.

[3] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, and J. Peters, "Trajectory planning for optimal robot catching in real-time," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2011, pp. 3719–3726.

[4] J. Malzahn, A. S. Phung, and T. Bertram, "A multi-link-flexible robot arm catching thrown balls," in *German Conf. Robot. (ROBOTIK)*, 2012, pp. 1–6.

[5] S. Kim, A. Shukla, and A. Billard, "Catching objects in flight," *IEEE Trans. Robot.*, vol. 30, no. 5, pp. 1049–1065, 2014.

[6] S. Salehian, M. Khoramshahi, and A. Billard, "A dynamical system approach for softly catching a flying object: Theory and experiment," *IEEE Trans. Robot.*, vol. 32, no. 2, pp. 462–471, 2016.

[7] D. Carneiro, F. Silva, and P. Georgieva, "The role of early anticipations for human-robot ball catching," in *IEEE Int. Conf. Auton. Robot Sys. Competitions (ICARSC)*, 2018, pp. 10–16.

[8] R. Andersson, "Understanding and applying a robot ping-pong player's expert controller," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. IEEE Computer Society, 1989, pp. 1284–1289.

[9] F. Miyazaki, M. Takeuchi, M. Matsushima, T. Kusano, and T. Hashimoto, "Realization of the table tennis task based on virtual targets," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 4, 2002, pp. 3844–3849.

[10] L. Acosta, J. J. Rodrigo, J. A. Mendez, G. N. Marichal, and M. Sigut, "Ping-pong player prototype," *IEEE Robot. Automat. Mag.*, vol. 10, no. 4, pp. 44–52, 2003.

[11] T. Senoo, A. Namiki, and M. Ishikawa, "High-speed batting using a multi-jointed manipulator," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 2, 2004, pp. 1191–1196.

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

[13] Z. Wang, C. H. Lampert, K. Mülling, B. Schölkopf, and J. Peters, "Learning anticipation policies for robot table tennis," in *Proc. IEEE Int. Conf. on Intel. Robot. Sys. (IROS)*, 2011, pp. 332–337.

[14] C. Liu, Y. Hayakawa, and A. Nakashima, "Racket control and its experiments for robot playing table tennis," in *Int. Conf. Robot. Biomimetics (ROBIO)*, 2012, pp. 241–246.

[15] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *Int. J. Robot. Res.*, vol. 32, no. 3, pp. 263–279, 2013.

[16] Y. Huang, B. Schölkopf, and J. Peters, "Learning optimal striking points for a ping-pong playing robot," in *Proc. IEEE Int. Conf. on Intel. Robot. Sys. (IROS)*, 2015, pp. 4587–4592.

[17] Y. Huang, D. Büchler, O. Koç, B. Schölkopf, and J. Peters, "Jointly learning trajectory generation and hitting point prediction in robot table tennis," in *IEEE-RAS Int. Conf. Humanoid Robot. (Humanoids)*, 2016, pp. 650–655.

[18] Y. Zhu, Y. Zhao, L. Jin, J. Wu, and R. Xiong, "Towards high level skill learning: Learn to return table tennis ball using Monte-Carlo based policy gradient method," in *IEEE Int. Conf. Real-time Comput. Robot. (RCAR)*, 2018, pp. 34–41.

[19] R. Mahjourian, N. Jaitly, N. Lazic, S. Levine, and R. Miikkulainen, "Hierarchical policy design for sample-efficient learning of robot table tennis through self-play," *Computing Research Repository (CoRR) in arXiv*, 2018. [Online]. Available: http://arxiv.org/abs/1811.12927

[20] W. Gao, L. Graesser, K. Choromanski, X. Song, N. Lazic, P. Sanketi, V. Sindhwani, and N. Jaitly, "Robotic table tennis with model-free reinforcement learning," in *Proc. IEEE Int. Conf. on Intel. Robot. Sys. (IROS)*, 2020, pp. 5556–5563.

[21] T. Kunz, P. Kingston, M. Stilman, and M. Egerstedt, "Dynamic chess: Strategic planning for robot motion," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2011, pp. 3796–3803.

[22] T. Kröger, K. Oslund, T. Jenkins, D. Torczynski, N. Hippenmeyer, R. B. Rusu, and O. Khatib, *JediBot – Experiments in Human-Robot Sword-Fighting*. Heidelberg: Springer International Publishing, 2013, pp. 155–166.

[23] A. Namiki and F. Takahashi, "Motion generation for a sword-fighting robot based on quick detection of opposite player's initial motions," *J. Robot. Mechatronics*, vol. 27, pp. 543–551, 10 2015.

[24] Y. Wang, E. Li, F. Wang, and B. Xu, "A virtual character learns to defend himself in sword fighting based on Q-network," in *IEEE Int. Conf. Tools Artif. Intel. (ICTAI)*, 2016, pp. 291–298.

[25] J. Dehesa, A. Vidler, C. Lutteroth, and J. Padget, "Touché: Data-driven interactive sword fighting in virtual reality," in *Proc. Conf. Human Factors Comp. Sys. (CHI)*, 2020, pp. 1–14.

[26] I. J. Da Silva, C. O. Vilao, A. H. Costa, and R. A. Bianchi, "Towards robotic cognition using deep neural network applied in a goalkeeper robot," in *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*. IEEE, 2017, pp. 1–6.

[27] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition," *Computing Research Repository (CoRR) in arXiv*, 2017. [Online]. Available: http://arxiv.org/abs/1710.03748

[28] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[29] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "RLlib: Abstractions for distributed reinforcement learning," in *Proc. of Int. Conf. on Machine Learning*, vol. 80, 2018, pp. 3053–3062.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *Computing Research Repository (CoRR) in arXiv*, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[31] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proc. of Int. Conf. on Learning Representations*, Y. Bengio and Y. LeCun, Eds., 2016.

[32] E. Coumans and Y. Bai, "PyBullet, a Python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.