Applying Reinforcement Learning to Robotic Dart Throwing

12/10/2025

Zaaim Rahman, Maunika Achanta,
Nivedh Koya, Wafee Choudhury

Overview

Introduction [01]

Architecture [02]

Simulation Setup & Training [03]

Robot Control [04]

Demo [05]

Results and Evaluations [06]

Conclusion and Future Work [07]

Introduction

- *Abstract: Our project centers around applying reinforcement learning (RL) to a dart-throwing application to test a robot arm's accuracy in throwing projectiles.*
- Reinforcement Learning needed for delayed-reward dart throwing, where early arm motions give no immediate feedback
- PyBullet simulation with SO-101 arm using two-stage training: dense rewards for throw mechanics, sparse rewards for target accuracy
- Integration of 6D pose estimation and a learned velocity/torque controller for real-world deployment

Architecture

Reinforcement Learning Environment

- Mimic the real dynamics of the robot
- Approximate dart behaviour

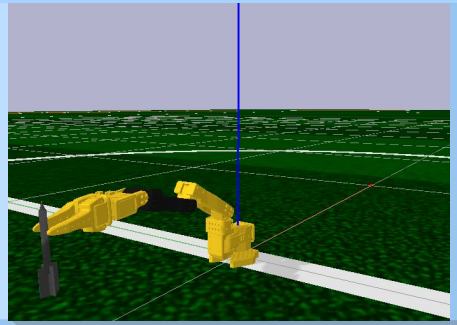
Training Algorithm to Train Policy

- Train the robot in the simulated environment
- Store trained models for testing

Deploying Policy to SO101-Arm

- Transfer policy for testing on real SO101-Arm

Components



PyBullet

A physics simulation library using the Bullet 2.X Engine with Python bindings.

Stablebaselines3 + Gymnasium

Standardized implementations of reinforcement learning environments to apply to custom environments

LeRobot API

A wrapper around the FeeTech STS3215 Motor specification table for motor control

Simulation Environment

Utilized PyBullet and URDF Files

Key activities

Develop simulation environment to allow the robot arm to learn to throw

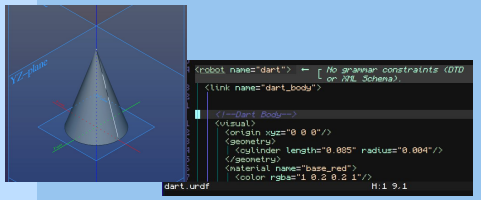
Challenges

- Limitations of aerodynamic simulation of dart fins
- Motor speed limitation
- Joint inconsistencies

URDF Files

SO101-Arm URDF pulled from GitHub repo

Create URDF file for board and dart



Simulation Debugging

Develop controller to control both simulated arm and real arm

Initial State

Develop classes for RobotAgent and Environments to reset to initial state

Collision detection for rewards

Training Setup

Observations

Joint Positions
+
Joint Velocities

The trajectory after release does not change the reward at all
⇒
The state of releasing the dart is associated with the reward

Actions

Joint Positions
+
Boolean Release

Reward Functions

Initial Reward
01

Rewards were given based on the position, orientation, linear velocity, and angular velocity of the dart.

Shoulder Panning
02

The model would always tend to the left side when throwing which caused it to lose the dart.

Velocity Loop-Hole
03

The model realized it could oscillate along the Z axis to accumulate linear velocity reward, solved by adding a discount factor over time

Release Angle
04

The release angle was optimized to 45 degrees in the simulation but did not translate well in real life. Solved by adding reward for desired angle.

Time Delta Limit
05

The real robot's motors could not keep up with the time delta decided by the physics simulation. Solved by creating buffertime between simulation steps.

Robot Control

FeeTech STS3215 Spec

FeeTech specification for motor memory table

Can be used to manipulate motor properties

Memory Table API in LeRobot

self.bus.sync_read()
self.bus.sync_write()

Internal functions used to read/write using UART adapter packets with motors

Current Joint Velocities

“Present_Velocities”
At Byte 58

This is the key to memory table mapping for receiving joint velocities

Joint Maximum Velocity

“Goal_Velocity”
At Byte 46

This is the key to memory mapping for setting the goal velocity of joints

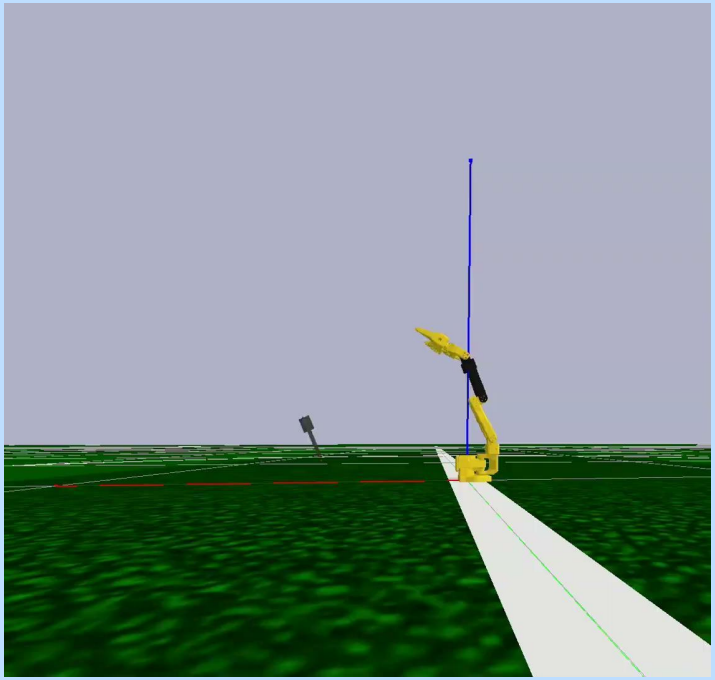
- Write delay required
- Test for moving speed

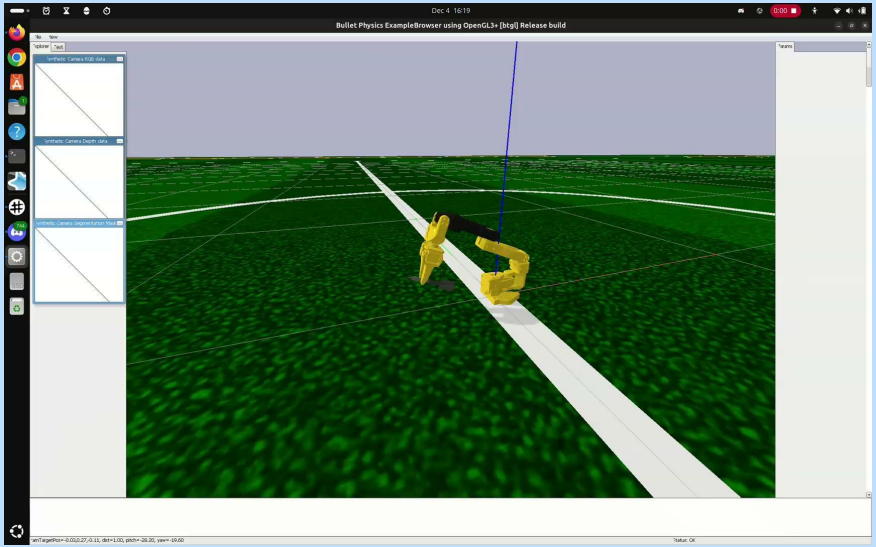
SO101FollowerII

Wrapper class to extend SO101Follower Class

Allow to easily transfer Policy

Demo




Dec 4 16:19
Bullet Physics ExampleBrowser using OpenGL3+ [btgl] Release build

Results

Our robot was able to learn the throwing motion and achieved throws of around one meter, but the results were not consistently reproducible. The gap between simulation and the real robot limited our ability to continue with more advanced training, and precise release timing remained difficult to master.

1. Successfully trained the robot to perform the full throwing motion
2. Achieved ~1-meter throws, but with inconsistent repeatability
3. Simulation - real differences and poor release timing prevented further training

Conclusion and Future Work

- Main limitation was the sim-to-real gap — inaccurate dart aerodynamics, joint inconsistencies, and motor constraints blocked stable policy transfer.
- Precise release timing remained the biggest challenge, preventing the robot from progressing to accurate target-based training.
- Future work includes improving simulation fidelity and building a more reliable release-timing mechanism for consistent throws.
- Incorporating 6D pose estimation and refined reward shaping will enable accurate, real-world, target-driven performance.

Thank you