Andrea Arigliano · Davide Lugli · Filippo Marinelli

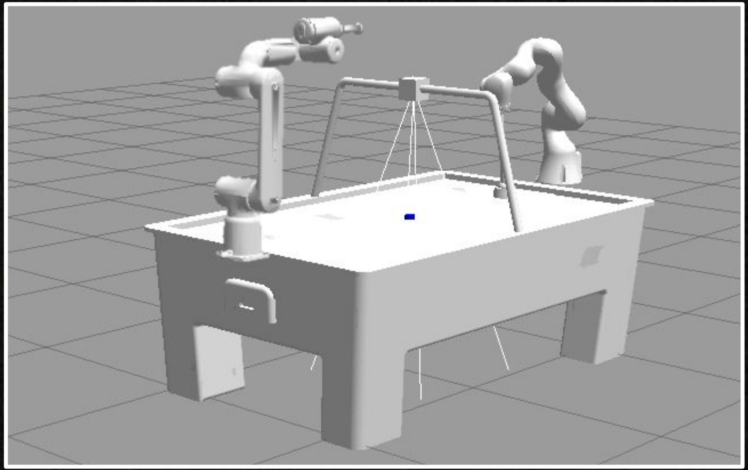
AiRL-Hockey

Teaching a robotic arm to play Air Hockey with Reinforcement Learning

Introduction

Goal: Teaching a robotic arm how to play a game of Air Hockey against an opponent without the need for human interaction, using Reinforcement Learning techniques.

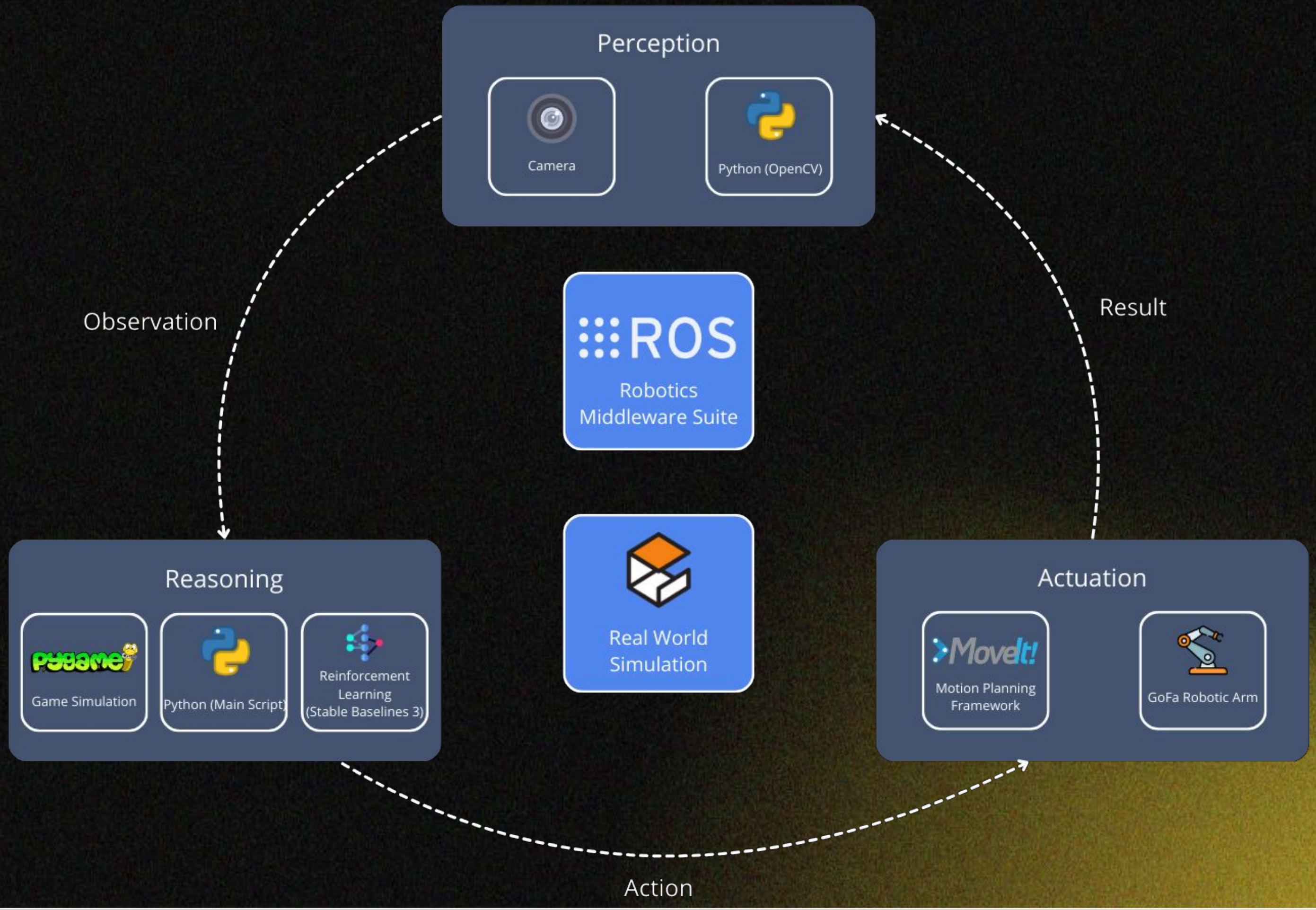
Motivation: This is a problem of interest for the academic community, as proven by the existence of a competition called the Air Hockey Challenge, where universities propose algorithms to compete against each other for prize money.



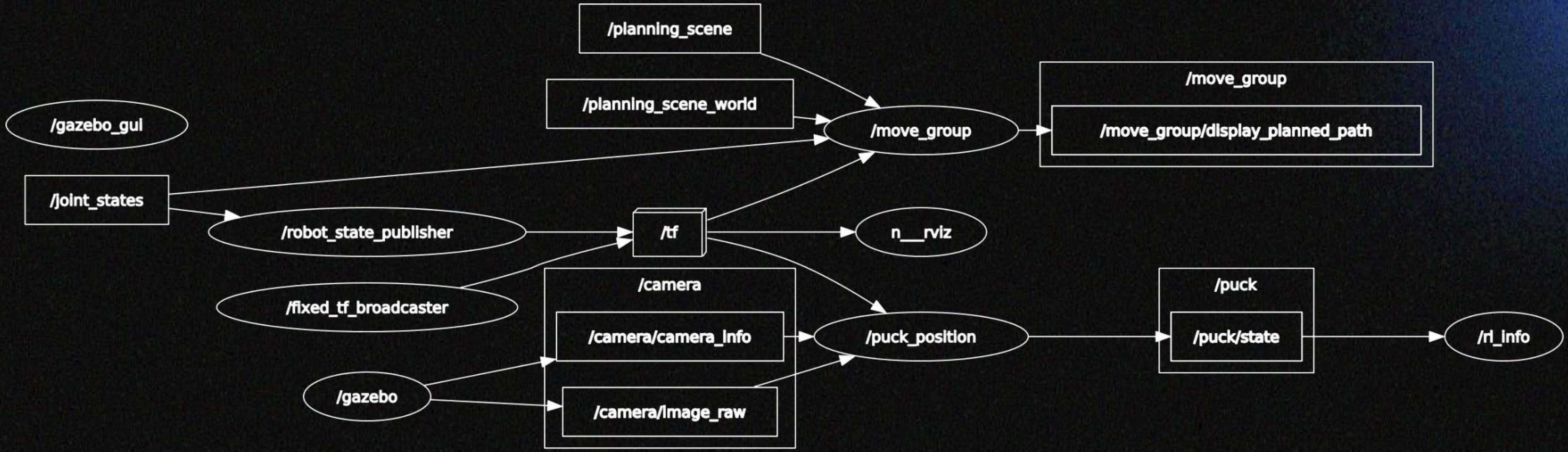
Architecture

The project was entirely developed and ran on a virtual machine running Ubuntu 20.04 and makes use of the following tools and frameworks:

Robot Models	GoFa CRB 15000, KUKA LBR iiwa 14 R820
Robotics Middleware Suite	ROS 1
Simulation Suite	Gazebo
Motion Planning Framework	MoveIt
Other Tools	Python (OpenCV, PyGame)
Reinforcement Learning Model	Stable Baselines 3


Perception
Camera
Python (OpenCV)
Observation
Result
ROS
Robotics
Middleware Suite
Reasoning
pygame
Game Simulation
Python (Main Script)
Reinforcement
Learning
(Stable Baselines 3)
Real World
Simulation
Actuation
MoveIt!
Motion Planning
Framework
GoFa Robotic Arm
Action

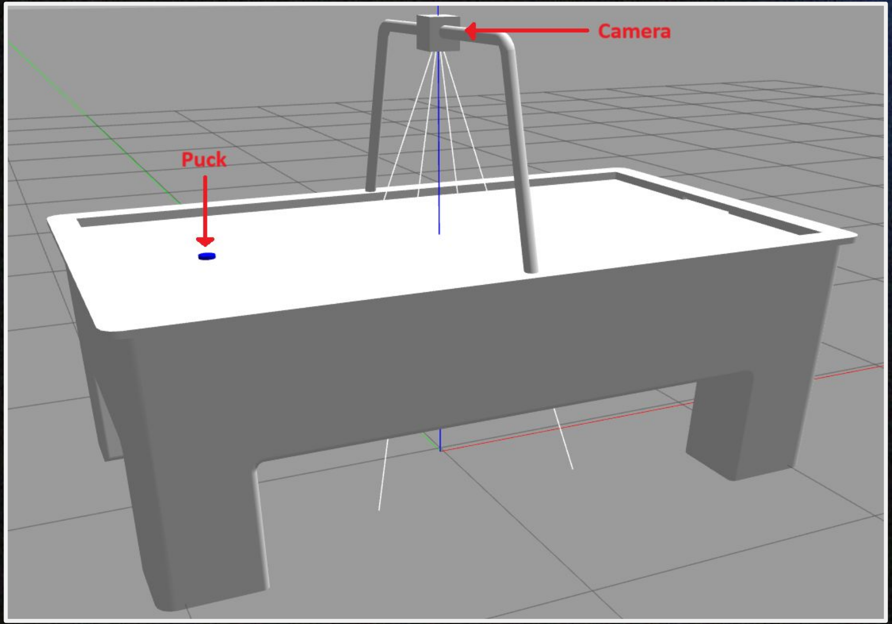
The core of the system: ROS



Simulation Environment

We modeled the world used for our simulation in **Gazebo**, by adding three simple elements:

- Air Hockey Table, with a very low friction coefficient and collision-enabled walls.
- Camera, with an increased Field of Vision in order to capture the whole game field.
- Hockey Puck, the disc that the robot needs to hit in order to play and score against its opponent.



Camera

01 Technical Specifications

- Horizontal FOV of 129°
- 800x800 Image Resolution
- Clipping Planes from 0.1 to 100 m
- Update Rates of 30 Hz
- ROS Topics:
  - /image_raw
  - /camera_info

02 Image Retrieval

A Python script is used to continuously retrieve images from the ROS topic.

03 Object Detection

A Python script is used to detect the puck in the image and extract its coordinates w.r.t. the world.
For this purpose, the OpenCV Python library has been used.

Camera: Pipeline

OpenCV

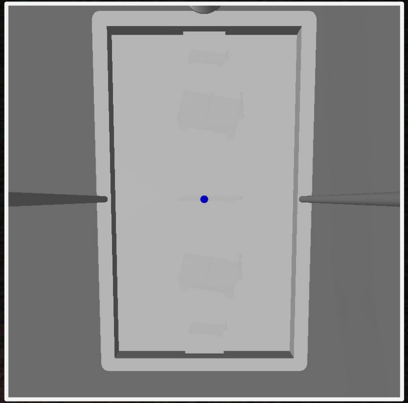
Buffering Differentiation

Ros Topic

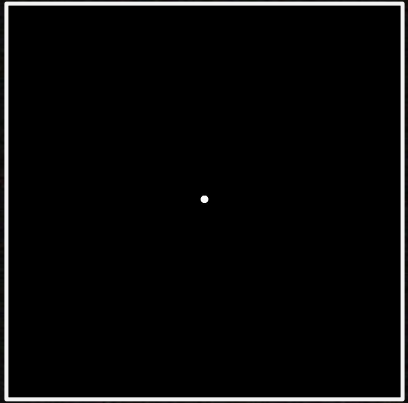
Camera to World

Camera: Puck Detection

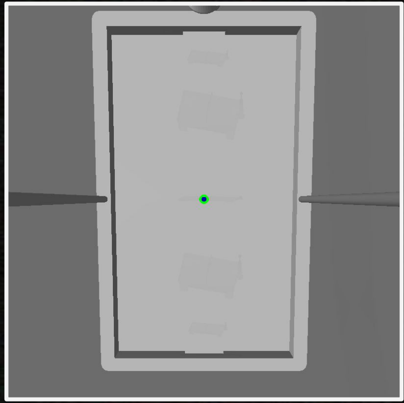
Raw Image



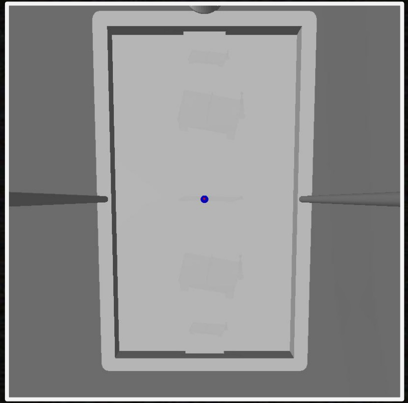
Mask



Puck Contour



Puck Center

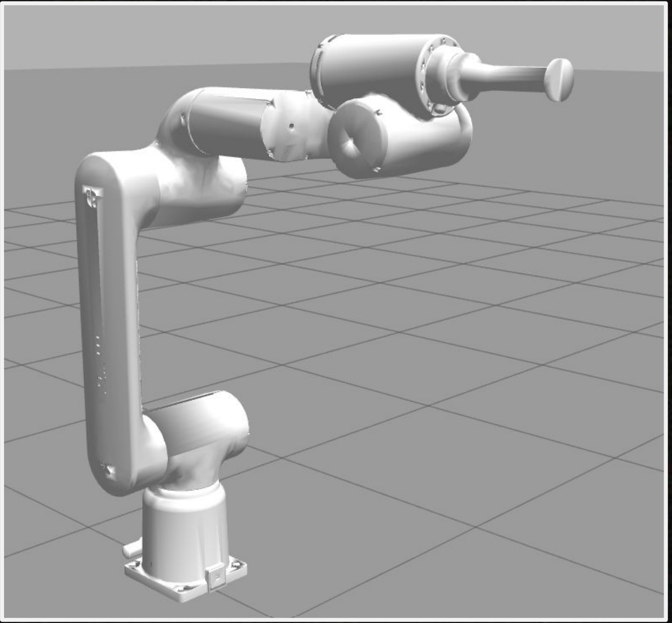


Robot: GoFa CRB 15000

- 6 Degrees of Freedom (6 Revolute Joints);
- Maximum Reach 0.95m;
- Maximum Payload 5kg.

This robot is an anthropomorphic arm, thus being suitable for this execution task as it emulates very closely the way an human arm would perform the movement.

The end-effector is a fixed joint link that resembles the mallet used in an Air Hockey match to hit the puck.



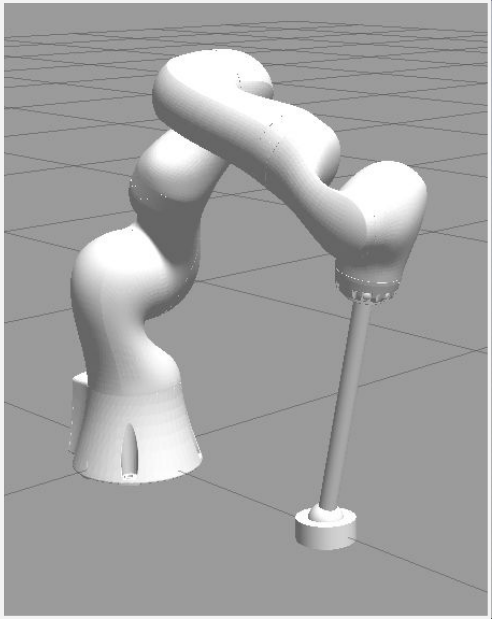
Robot: KUKA LBR iiwa 14 R820

- 7 Degrees of Freedom (7 Revolute Joints);
- Maximum Reach 0.82m;
- Maximum Payload 7-14kg.

This robot model was chosen because it is the official robot of the Air Hockey Challenge '23 from the University of Darmstadt.

The end effector was modeled to resemble the original one by attaching a fixed pole to the last link of the robot, and at its other end the mullet linked to the pole with a spherical joint.

The spherical joint was constructed by overlapping three revolute joints on the three different axes respectively.



Robot: Movement

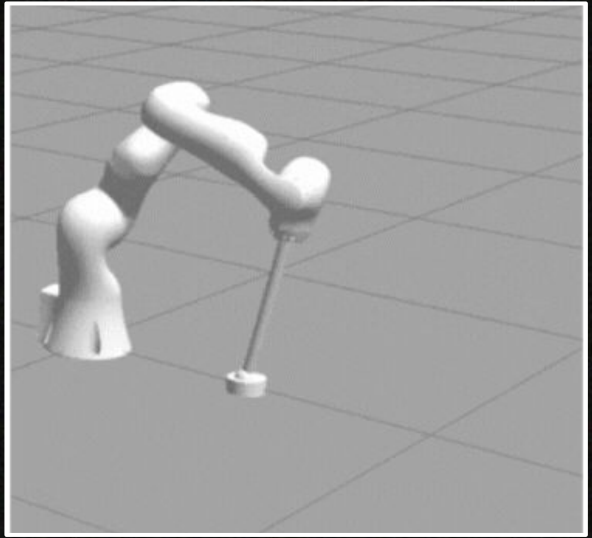
Problem: Fast, reactive and fluid robot movement needed to counteract the fast movement of the puck on the field.

Solution: Low-level joint velocity control implemented with a velocity-actuated joint trajectory controller.

The computation of the joint velocity coefficients is done via Inverse Kinematics using the pseudo-inverse Jacobian:

$$\dot{q} = J^{-1}(q)\dot{x}$$

In order to fully experiment with the movement of the robots, a custom **TELEOP** module was implemented, which moves the robot with WASD keys.



Reinforcement Learning

01 Technical Specifications

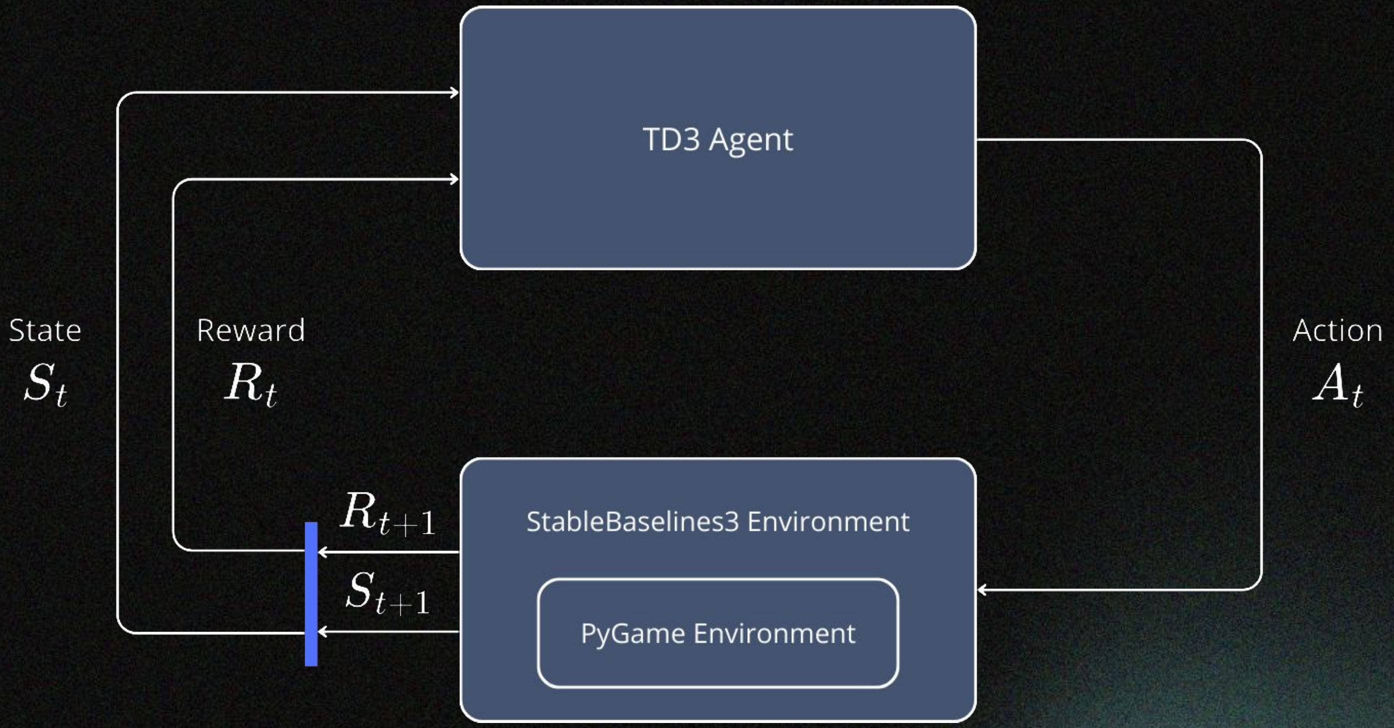
- **PyGame** for the training environment
- **Stable Baselines 3** as the RL framework
- **Tensorflow** as the backend
- **Tensorboard** to monitor the training process

02 RL Algorithm

TD3 - Twin Delayed Deep Deterministic Policy Gradient

- **Observation Space** → 8-dimensional continuous bounded Box
  - Position and Velocities of Puck and Mallet
- **Action Space** → 2-dimensional continuous bounded Box
  - Desired Mallet Velocity of the Mallet
- **2-Phase Reward Model:**
  - Attack Phase → Approaching the puck to score,
  - Defense Phase → Defending the goal staying in a safe-zone.

Reinforcement Learning: Pipeline

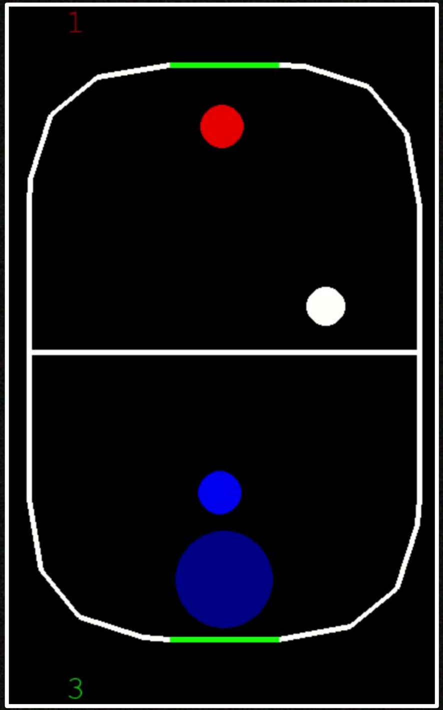


Reinforcement Learning: Simulation

Real-World game simulation using **PyGame** environment, with every physics-related parameter tunable w.r.t. desired simulation behaviour (friction, mass, speed, bounce).

The simulation renders the table, puck, and two mallet competing in the game. The sizes of the table and every other component is tunable in the constants section, achieving a dynamic testbed for experiments.

The game implements a simple heuristic AI to play with the training robot agent, which always tries to hit the puck and defend when unreachable.



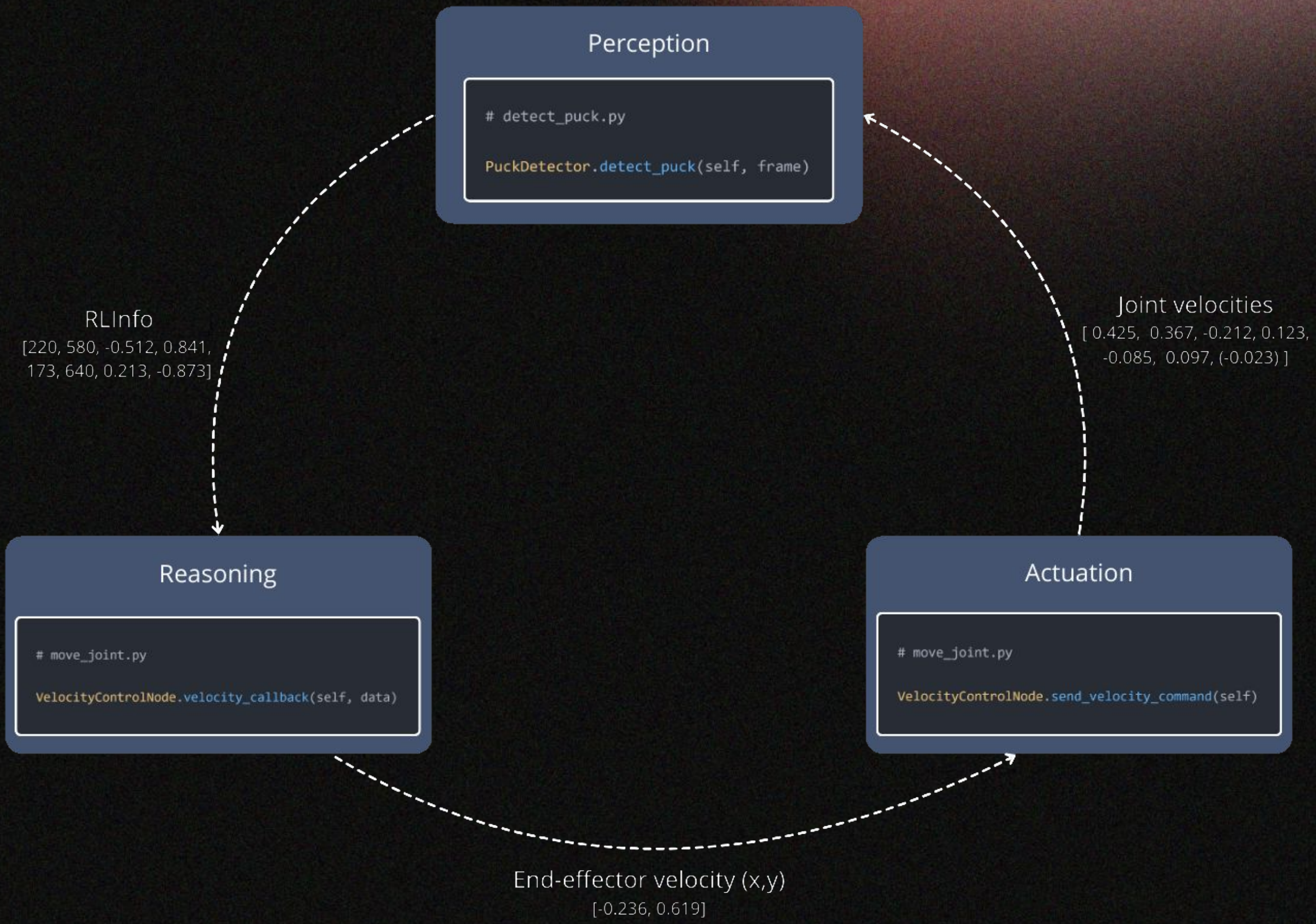
Framework

Simulation

- **Gazebo** → Environment
- **ODE (Open Dynamics Engine)** → Physics
- Low **friction** parameter for the table
- High **bounce** parameter for the puck
- Robot **fixed** at the table's end

Frames and transforms

- **Coordinates** expressed w.r.t. *world frame*
- **Transforms** published on dedicated topic
  - camera_link ⟷ world
  - base_link ⟷ world
- **Conversions** between reference frames
  - Puck detection**:**
    camera_link → world
  - Robot movement**:**
    base_link → world → obs


Perception
# detect_puck.py
PuckDetector.detect_puck(self, frame)
RLInfo
[220, 580, -0.512, 0.841,
173, 640, 0.213, -0.873]
Joint velocities
[0.425, 0.367, -0.212, 0.123,
-0.085, 0.097, (-0.023)]
Reasoning
# move_joint.py
VelocityControlNode.velocity_callback(self, data)
Actuation
# move_joint.py
VelocityControlNode.send_velocity_command(self)
End-effector velocity (x,y)
[-0.236, 0.619]

Limitations

- **Heavy simulation** → The physics components of the simulation like friction and restitution coefficients are strongly impacting the performances of the simulation;
- **Large training times** → The training times of the RL model spans around 6-8 hours without GPU acceleration, thus being time-intensive;
- **More training needed** → The usual training steps for complex RL environments go up to hundreds of millions of steps;
- **Unstable learning** → Careful tuning of the parameters is needed since the case-study is a reward sparse environment.

Future Works

- **Lighter simulation environment** → Porting the simulation to newer engines could benefit the experimenting process, like MuJoCo which is used in the Air Hockey Challenge 23';

- **GPU Parallelized training** → To counter the high training times of the model;

- **Model-Based Algorithm** → To counter the instability in the training process due to the sparsity of the reward model, a model-based RL algorithm could be implemented:
    - In this way, learning from previously gathered experiences could strongly benefit the learning process stabilizing it to more meaningful strategies,
    - And could also be beneficial to introduce multiple levels of rewards which are calculated separately and not accumulated, for e.g.
        - Reward for Defense Phase
        - Reward for Attack Phase
        - Reward for Preparation Phase

Thanks!

Do you have any questions?

Andrea Arigliano
270188@studenti.unimore.it

Davide Lugli
269692@studenti.unimore.it

Filippo Marinelli
274790@studenti.unimore.it