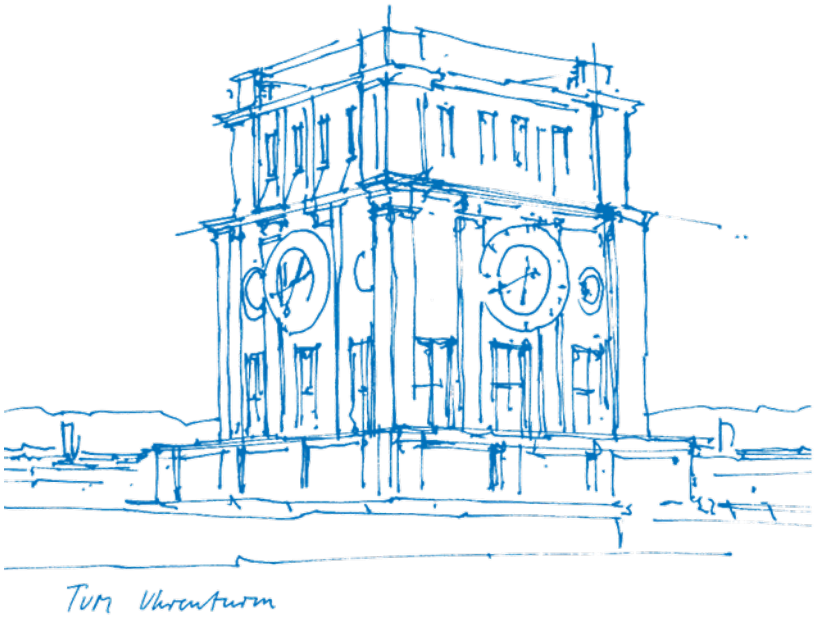
Complex Reflexes for Low-Level Robotic Systems

TUM Data Innovation Lab

Cristina Cipriani, Yang Liu, Andrea Linxen

Garching, 24. July 2018

Supervisor:	Professor Dr. Massimo Fornasier
Scientific Lead:	Dr. Martin Bischoff
Project Lead:	Dr. Ricardo Acevedo Cabra
Advisor:	Adrian Sieler



Introducing the team

Cristina Cipriani
Master Mathematics

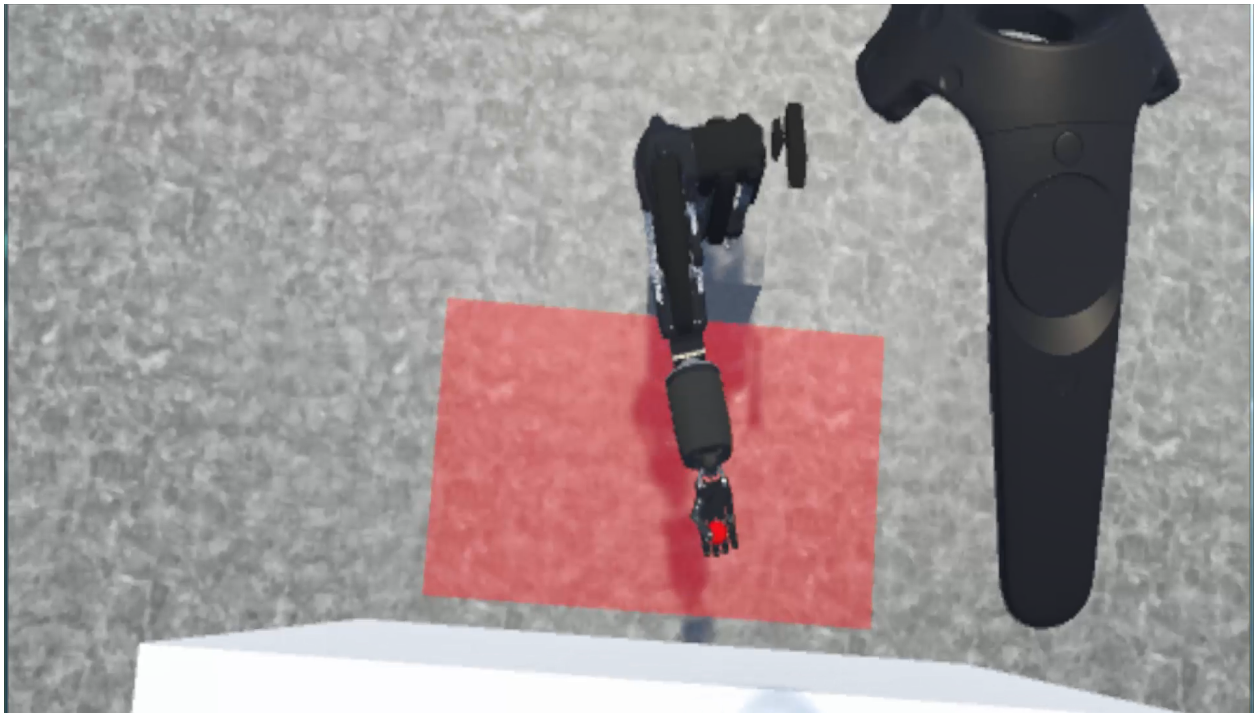
Andrea Linxen
Master Mathematics

Yang Liu
Master Informatics

Introducing the trained robot

Goal:

Communication framework between a robot and a simulation server which applies machine learning



Outline

- **Equipment and Software**
  - The ShadowArm Robot
  - Introduction to ROS and Gazebo
  - The Unity Environment
  - ML-Agents

- **Development**

  - ROS Communication and Gazebo Simulation
  - GrabBall Scene
  - ReachBall Scene

The ShadowArm robot

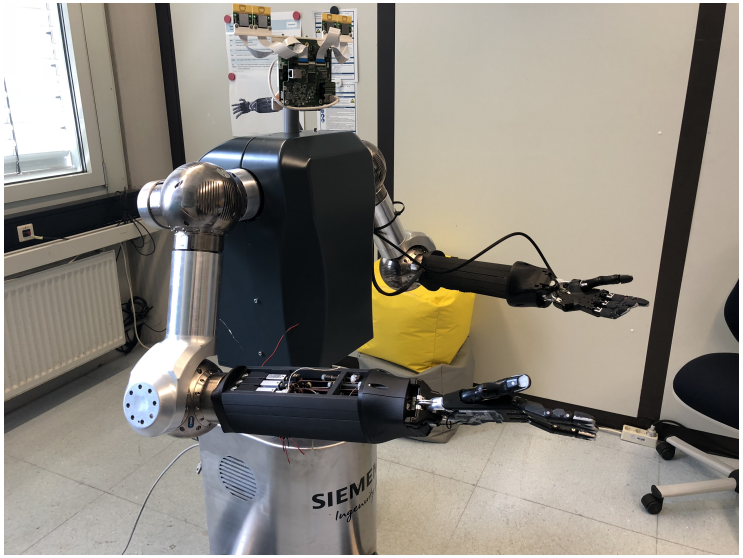
Combination of the Shadow Dexterous Hand and an extendable machine arm

Machine Arm

- Rotating trunk with extendable arm
- Full agility within the sphere

Shadow Dexterous Hand

- Similar to an average human hand
- Tendon-driven system with 24 joints
- Compatible with ROS



Outline

- **Equipment and Software**
  - The ShadowArm Robot
  - **Introduction to ROS and Gazebo**
  - The Unity Environment
  - ML-Agents

- **Development**

  - ROS Communication and Gazebo Simulation
  - GrabBall Scene
  - ReachBall Scene

ROS - Introduction

ROS (***Robot Operating System***) is one of the most famous and wildly used open source middle-ware in the world, designed to operate different robotic systems.

ROS provides a set of software libraries and tools that help you build robot applications, including:

- Hardware abstraction
- Low-level device control
- Implementation of commonly-used functionality
- Message-passing between processes
- Package management

ROS - Concepts

1. Node
2. Topic
3. ROSCore
4. Message

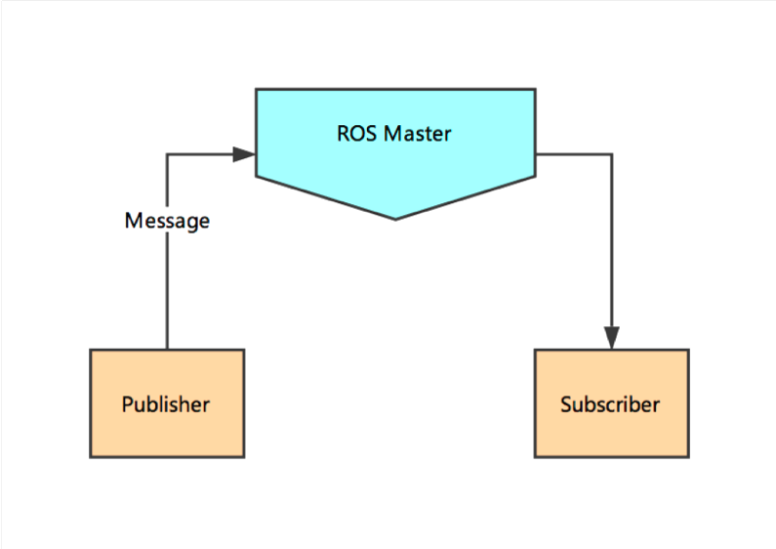
ROS - Node and Topic

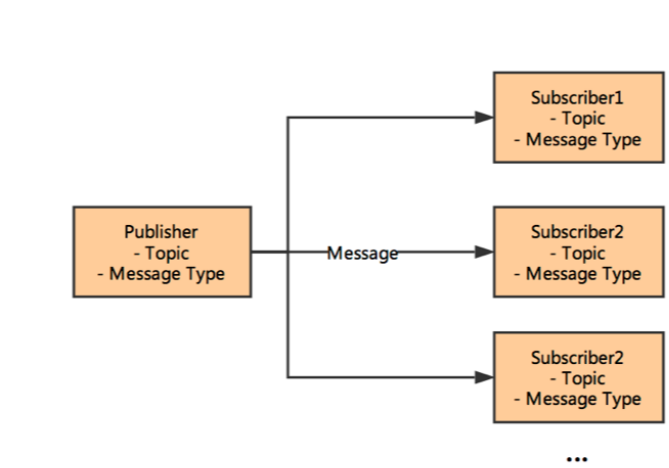
Node

A node is an entity that uses ROS to communicate with other nodes.

Topic

Nodes can publish messages to a topic as well as subscribe to a topic to receive messages.





ROS - Master

- ROS-Master manages all the nodes and their topics or services in the ROS Network.

- ROS-Master tracks the activities of all the nodes, and enable individual ROS nodes to locate one another.

- Once the locating is done, nodes can communicate peer-to-peer.

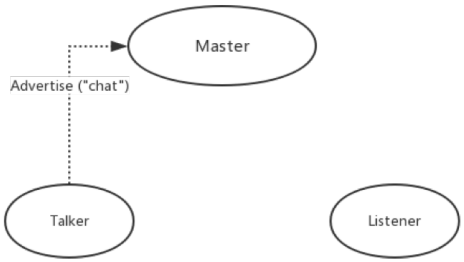


Figure: Talker-node advertises a new topic

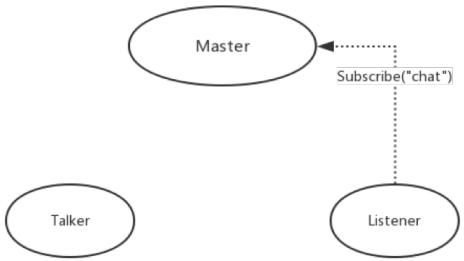


Figure: Listener-node subscribes a topic

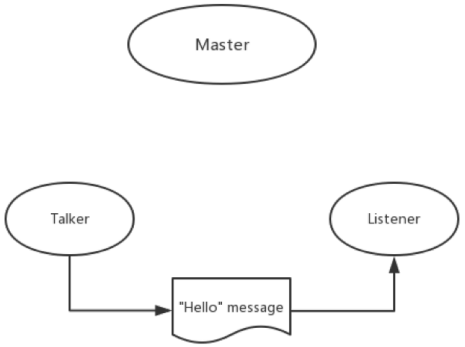


Figure: Talker publishes a new message on the topic ”chat” that Listener has subscribed

ROS - Message

- The communication between nodes is achieved by passing messages.

- The description of a message is written in a `.msg` file.

Odometry.msg

std_msgs/Header **header**
string **child_frame_id**
geometry_msgs/PoseWithCovariance **pose**

- `geometry_msgs/Point position`
- `geometry_msgs/Quaternion orientation`

geometry_msgs/TwistWithCovariance **twist**

- `geometry_msgs/Vector3 linear`
- `geometry_msgs/Vector3 angular`

Joint_State.msg

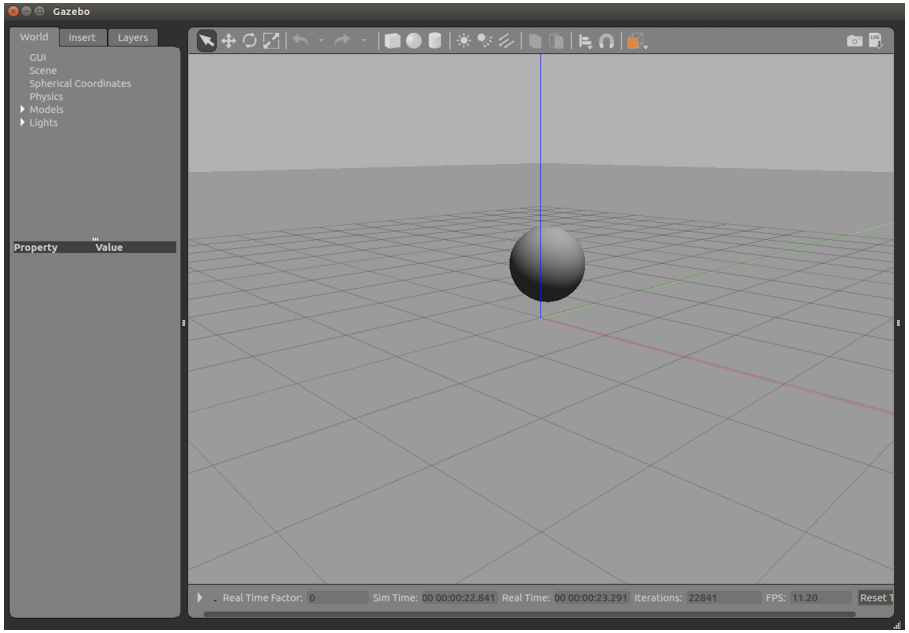
std_msgs/Header **header**
string[] **name**
float64[] **position**
float64[] **velocity**
float64[] **effort**

Gazebo Introduction

- A well-designed simulation tool that has its own physics engine
- Optimal to work hand in hand with ROS

- Application area:
  - Test algorithms
  - Design robots
  - Perform regression testing
  - Simulate indoor/outdoor environment

- Important components in our project:
  - URDF - To model an object or to assemble a robot with multiple joints
  - Plugins - To control the behavior of an object in Gazebo
  - Launch file - To start a Gazebo world with all the related objects/sensors in it.



Outline

- **Equipment and Software**
  - The ShadowArm Robot
  - Introduction to ROS and Gazebo
  - **The Unity Environment**
  - ML-Agents

- **Development**

  - ROS Communication and Gazebo Simulation
  - GrabBall Scene
  - ReachBall Scene

The Unity Environment

Game engine capable of simulating a physical training environment

Components of the Physics Engine

Rigidbodies

Regulate the influence of external forces

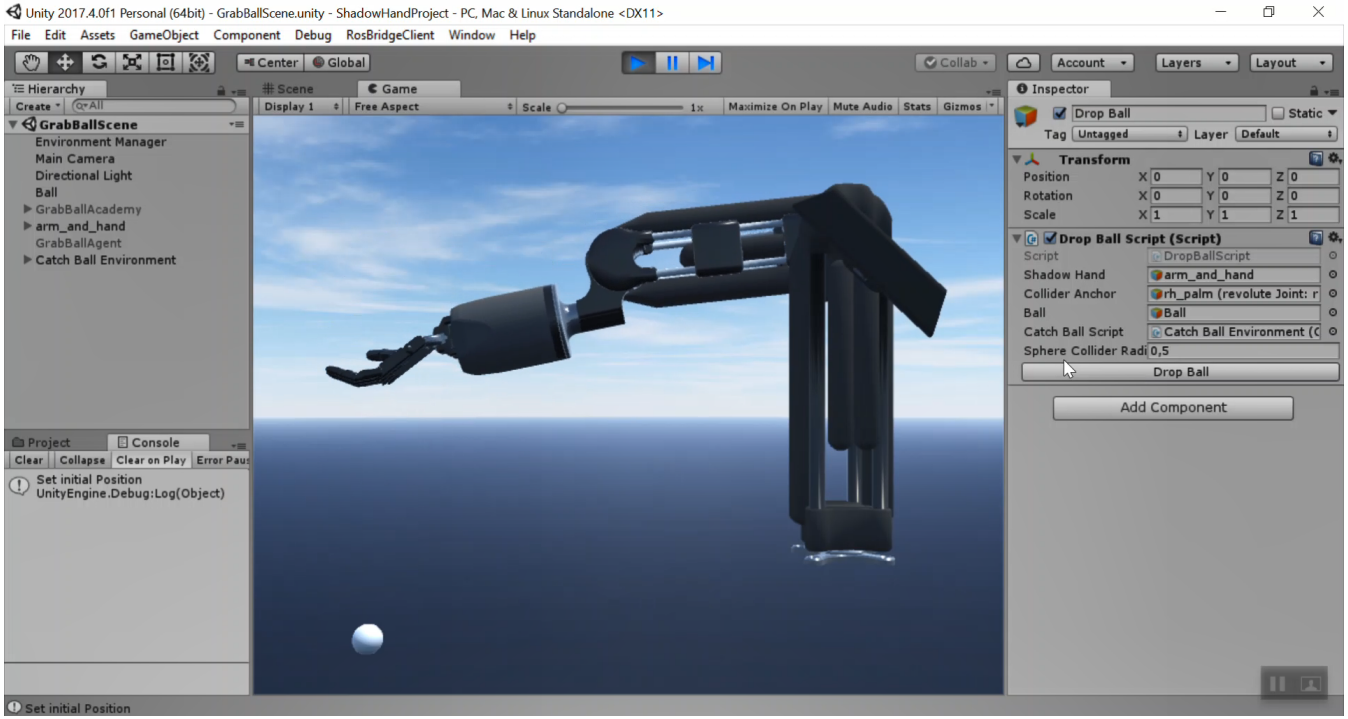
Layer-Based Collision detection

Detecting collisions depending on layers

Joints

Influence the position and movement of connected Rigidbodies

The Unity training environment



Outline

- **Equipment and Software**
  - The ShadowArm Robot
  - Introduction to ROS and Gazebo
  - The Unity Environment
  - **ML-Agents**

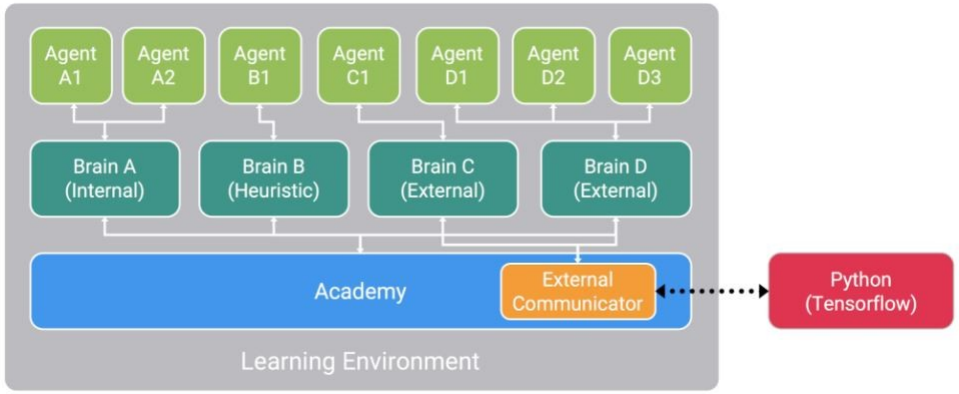
- **Development**

  - ROS Communication and Gazebo Simulation
  - GrabBall Scene
  - ReachBall Scene

ML-Agents

Unity Machine Learning Agents (ML-Agents) is an open-source Unity plugin that enables games and simulations to serve as environments for training intelligent agents.

- **Agent:**
  actor that can observe its environment and decide on the best course of action.
- **Brain:**
  it encapsulates the decision making process. Four types are allowed: external, internal, player and heuristic.
- **Academy:**
  it orchestrates all the Agent and Brain objects in a Unity scene.



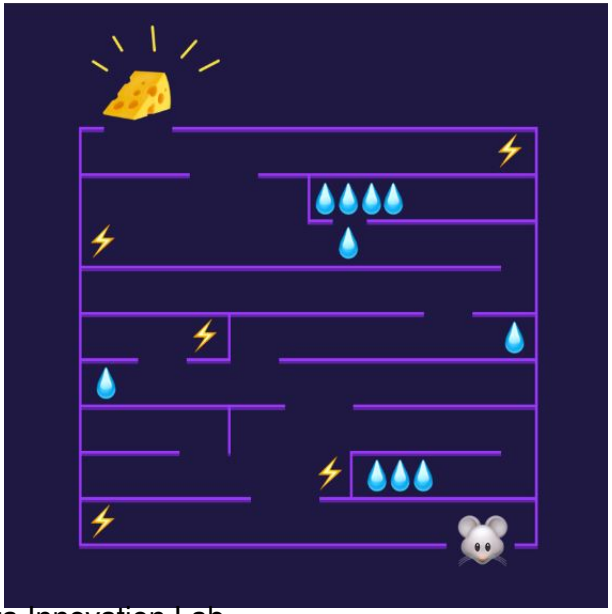
Reinforcement Learning

Learning what to do in order to maximize a numerical reward signal.

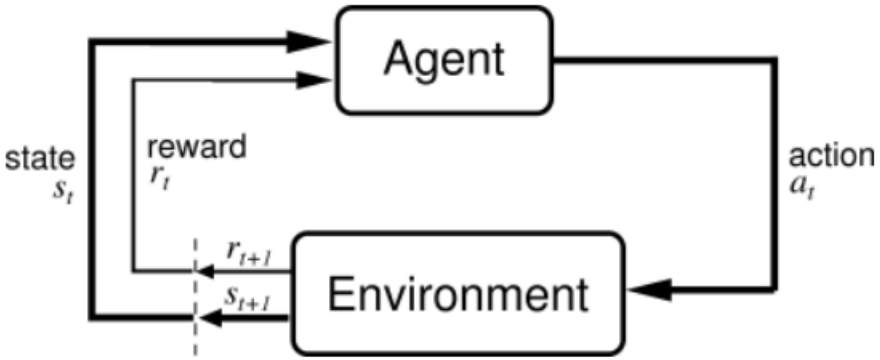
The learner is not told which actions to take, but it must discover itself which actions yield the most reward by performing them.
$\longrightarrow$ ***"trial-and-error search"***

Actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards.
$\longrightarrow$ ***"delayed reward"***

- **Cheese:** +1000 points
- **Water drops:** +10 points
- **Flash:** -100 points



Reinforcement Learning



$t = 0, 1, 2, ..$
State s_t
Action a_t
Reward r_t

$\longrightarrow$ ***policy:***

$$\pi_t(a|s) = Pr(a_t = a|s_t = s)$$

mapping from states to probabilities of selecting each possible action

Values Functions

State-value function: expected return when starting in s and following π thereafter:

$$V^{\pi}(s) = E_{\pi}(R_t | s_t = s) \tag{1}$$

Action-value function: expected return starting in s, taking action a, and then following policy π:

$$Q^{\pi}(s, a) = E_{\pi}(R_t | s_t = s, a_t = a) \tag{2}$$

Values Functions

State-value function: expected return when starting in s and following π thereafter:

$$V^{\pi}(s) = E_{\pi}(R_t | s_t = s) \tag{3}$$

Action-value function: expected return starting in s, taking action a, and then following policy π:

$$Q^{\pi}(s, a) = E_{\pi}(R_t | s_t = s, a_t = a) \tag{4}$$

$\longrightarrow$ ***Optimal values:***

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \tag{5}$$

$$Q^{*}(s, a) = \max_{\pi} Q^{\pi}(s, a) \tag{6}$$

Values Functions

State-value function: expected return when starting in s and following π thereafter:

$$V^{\pi}(s) = E_{\pi}(R_t | s_t = s) \tag{7}$$

Action-value function: expected return starting in s, taking action a, and then following policy π:

$$Q^{\pi}(s, a) = E_{\pi}(R_t | s_t = s, a_t = a) \tag{8}$$

$\longrightarrow$ ***Optimal values:***

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \tag{9}$$

$$Q^{*}(s, a) = \max_{\pi} Q^{\pi}(s, a) \tag{10}$$

$\longrightarrow$ This is never actually done!

Values Functions

State-value function: expected return when starting in s and following π thereafter:

$$V^{\pi}(s) = E_{\pi}(R_t | s_t = s) \tag{11}$$

Action-value function: expected return starting in s, taking action a, and then following policy π:

$$Q^{\pi}(s, a) = E_{\pi}(R_t | s_t = s, a_t = a) \tag{12}$$

$\longrightarrow$ ***Optimal values:***

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \tag{13}$$

$$Q^{*}(s, a) = \max_{\pi} Q^{\pi}(s, a) \tag{14}$$

$\longrightarrow$ This is never actually done!
$\longrightarrow$ Model-free methods

Outline

- **Equipment and Software**
  - The ShadowArm Robot
  - Introduction to ROS and Gazebo
  - The Unity Environment
  - ML-Agents
- **Development**
  - **ROS Communication and Gazebo Simulation**
  - GrabBall Scene
  - ReachBall Scene

Gazebo Setup - Requirements

- Setup the simulation scene as close as possible to the real-world condition
  - Gravity
  - Initial status
- An object (ball) which does a parabolic movement
  - Random initial position, speed, direction
  - Falls eventually within the reachable area of the robot hand
- The *Shadow Robot* should start with random pose
- Gazebo and Unity are able to communicate via ROS

Gazebo Setup - Ball Model

ball.urdf

```
<?xml version="1.0"?>
<robot name="ball">
     <pose>0 0 5 0 0 0</pose>

     <link name="ball link">
        <inertial >
          <mass value="1"/>
          <inertia ixx="0" ixy="0" ixz="0"
                   iyy="0" iyz="0" izz="0" />
        </inertial>
        <collision name="collision">
          <geometry>
            <sphere radius="0.05"/>
          </geometry>
        </collision>

        <visual name="visual">
           <geometry>
              <sphere radius="0.05"/>
           </geometry>
        </visual>
     </link>

     <gazebo>
        <plugin name="ball push"
                filename="libballpush.so"/>
     </gazebo>
</robot>
```

Explanation

⟶ **<pose>** Initial position and initial orientation

⟶ **<link>** The main content of this ball object

⟶ The inertial properties of the link, include mass and rotational inertia

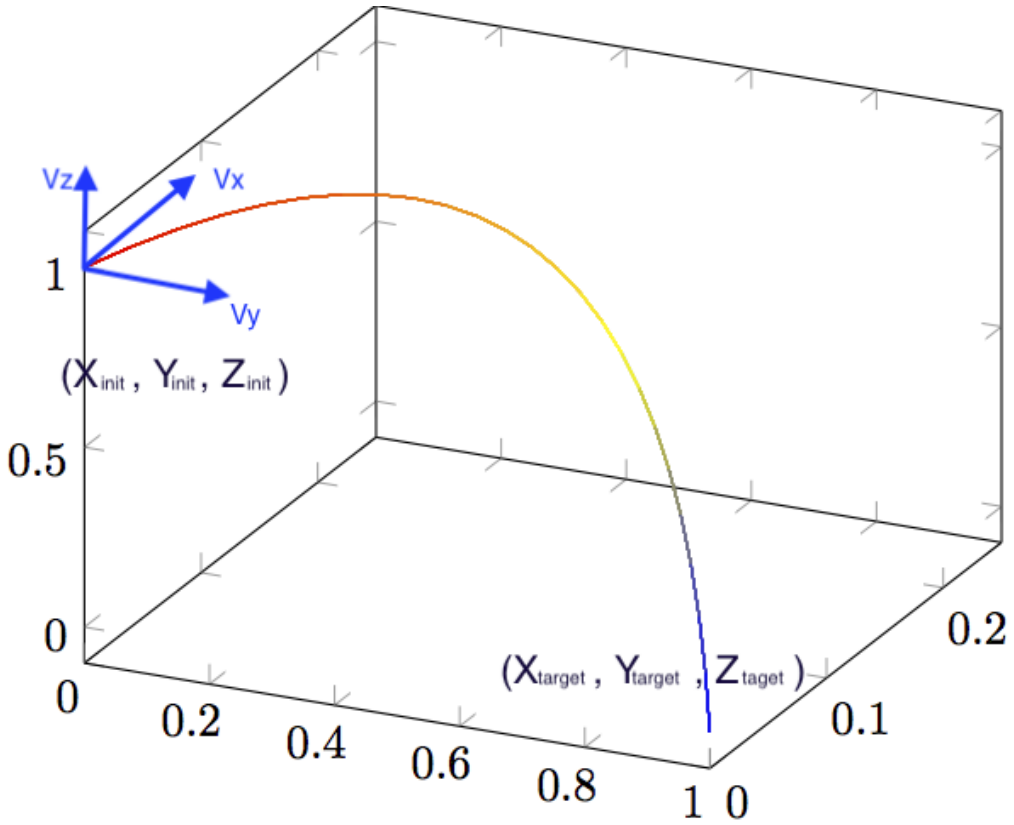
⟶ **<collision>** defines the boundary of the interaction with other objects in the simulation.

⟶ **<visual>** Shape of the object (box, cylinder, etc.) for visualization purposes

⟶ **<gazebo>** The plugin we use to control the movement

Gazebo Setup - Ball Movement

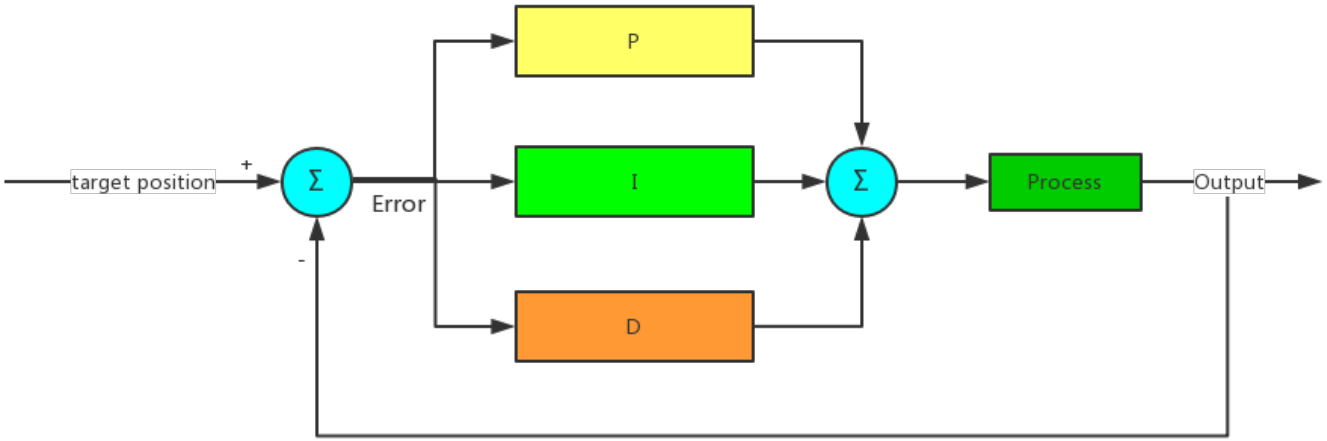
In order to let the ball falls within the reachable area of the robot hand, we generate the final position where the ball should fall into, then calculate the initial position of the ball backwards.



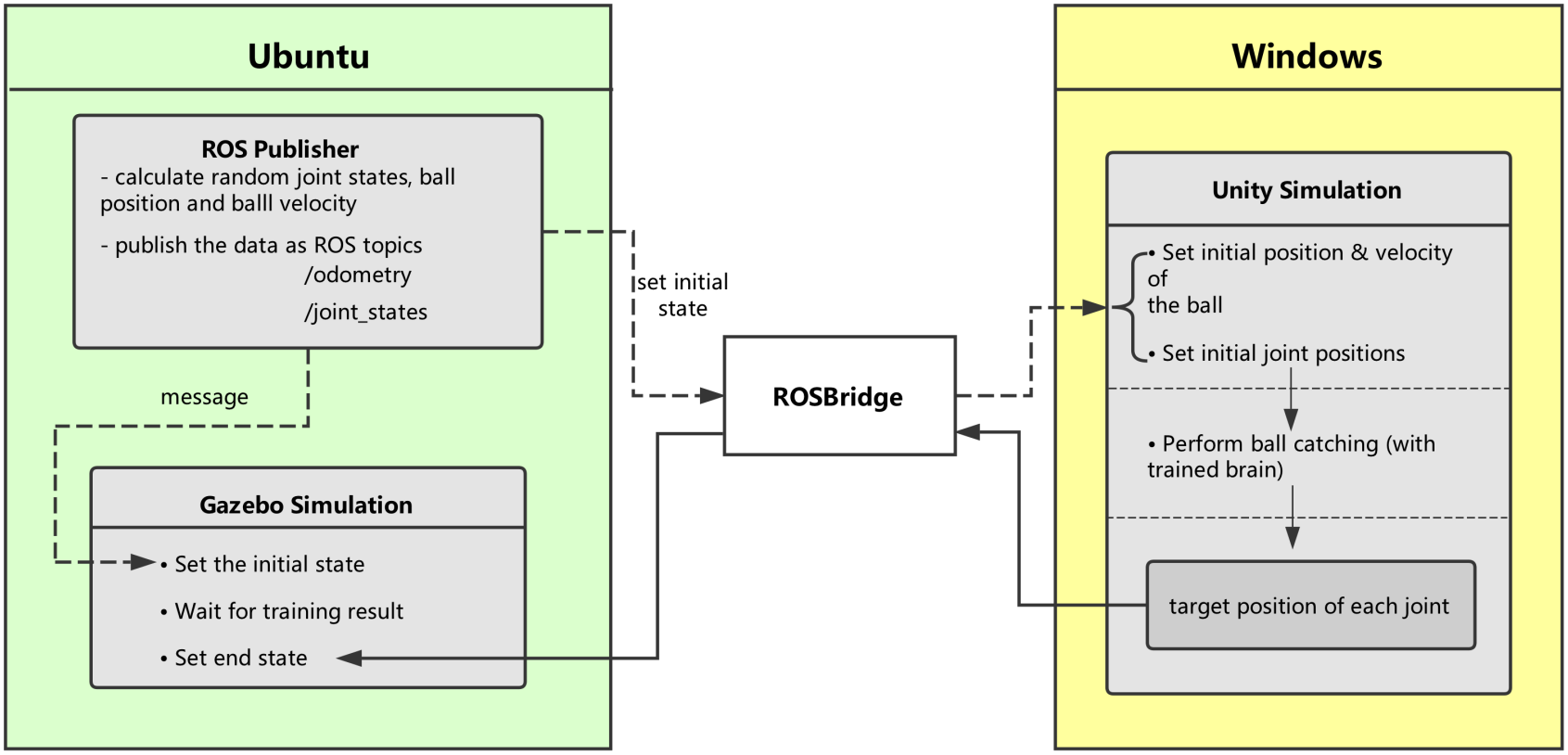
Gazebo Setup - Joint control of Shadow Robot

- Two approaches to control the joints of the Shadow Robot

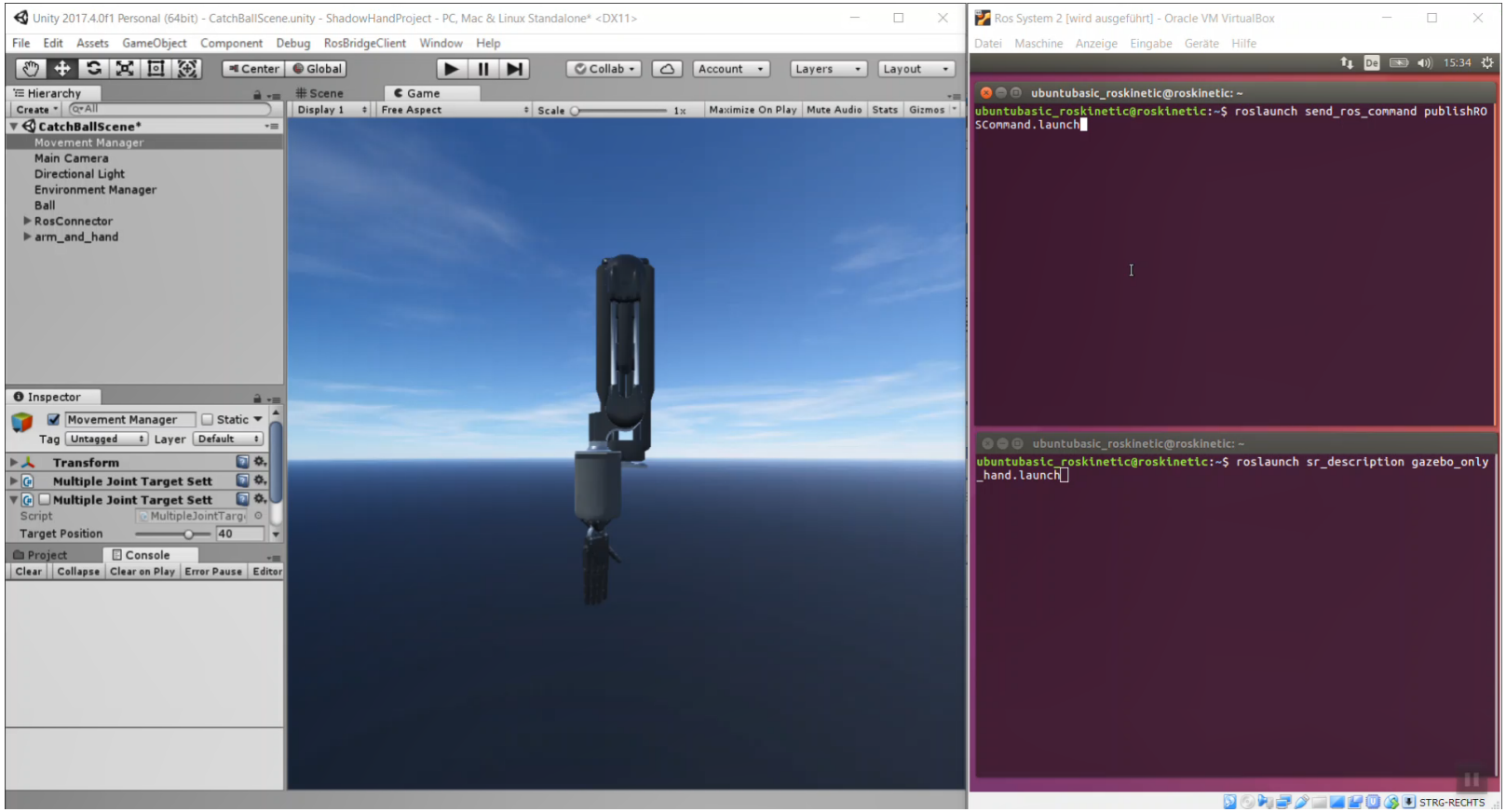
1. Use a native **ROS service** in Gazebo to directly change the joint positions
   + Easy to use, no effort to be made to write an additional plugin for that.
   - Extra step by opening another Terminal in Ubuntu, extra waiting time
2. Use **PID-controller** to apply force to each joint accordingly
   + Reusable when we got target joint positions from Unity
   + Physics simulation, calculated forces are applicable for real-world robot.



Gazebo Setup - ROS Communication



Communication between Ros and Unity



Outline

- **Equipment and Software**
  - The ShadowArm Robot
  - Introduction to ROS and Gazebo
  - The Unity Environment
  - ML-Agents

- **Development**
  - ROS Communication and Gazebo Simulation
  - **GrabBall Scene**
  - ReachBall Scene

Creating the GrabBall Agent

Goal

Grabbing and securing a ball with the fingers of the Shadow Dexterous Hand

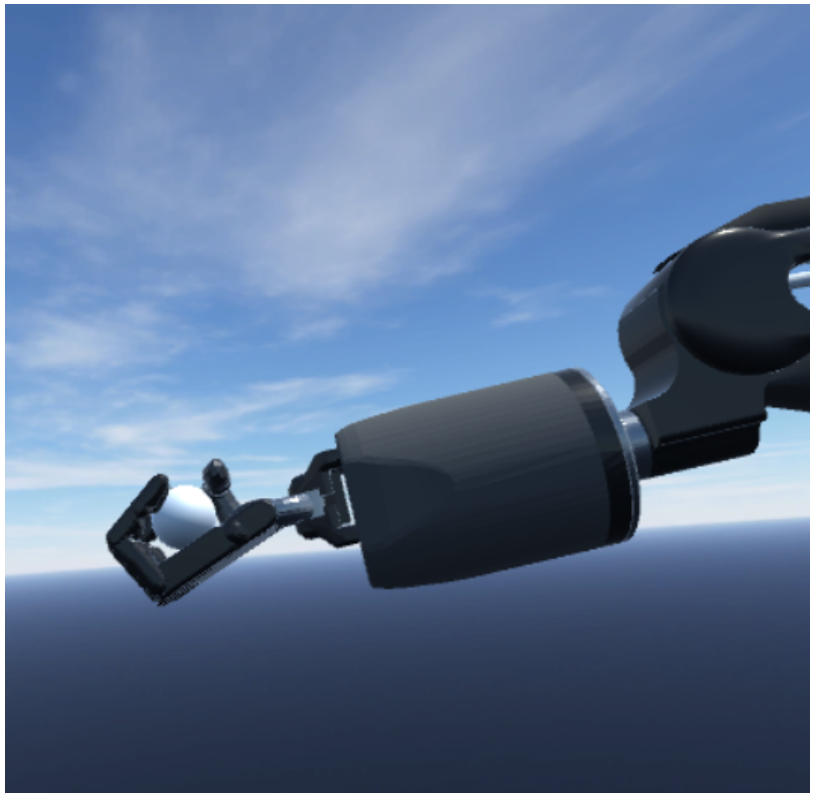
Simplifications

- **Modified Surface**
  Increasing friction and reducing elasticity
- **Layer-based collision detection**
  Preventing internal collisions
- **Drop ball in straight line**
  Neglecting horizontal forces
- **Decreasing the degrees of freedom**
  Fingers curl into the palm depending on a variable force
- **Tilting the palm downwards**
  Make ball roll towards the fingers

Training the GrabBall Agent

Design of the GrabBall Agent

- **Start**
  Create a trigger-sphere around ShadowHand
- **Collect Observations**
  Recording status of ball and palm
- **Agent Action**
  - Called at each time step
  - Decides on value of force
  - Activates Agent's reset if ball is dropped
  - Assigns Rewards and Penalties
- **Agent Reset**
  Move ball and Agent into initial position



Outline

- **Equipment and Software**
  - The ShadowArm Robot
  - Introduction to ROS and Gazebo
  - The Unity Environment
  - ML-Agents

- **Development**

  - ROS Communication and Gazebo Simulation
  - GrabBall Scene
  - **ReachBall Scene**

ReachBall Scene

Goal: reach the exact position of the Ball

Simulation setup:

- the **Hand and the Fingers** remain fixed
- four joints of the model are actively controlled (**Trunk, Shoulder, Elbow and Wrist**)

Learning Environment

- **Which observations should we collect?**
  - velocity and the relative position of the Ball and Palm
  - position of all the relevant joints (Shoulder, Elbow, Wrist)
  - velocity of all the relevant joints (Shoulder, Elbow, Wrist)
  - Trunk's angular velocity

Learning Environment

- **Which observations should we collect?**
  - velocity and the relative position of the Ball and Palm
  - position of all the relevant joints (Shoulder, Elbow, Wrist)
  - velocity of all the relevant joints (Shoulder, Elbow, Wrist)
  - Trunk's angular velocity
- **Which actions can be taken?**
  - **Trunk's Rotation:** it may rotate 180 degree around itself
  - **Arm's Extension:** the movements of Shoulder, Elbow and Palm are constrained in order to move together to a specified target position.

$\longrightarrow$ *huge reduction of the degrees of freedom*

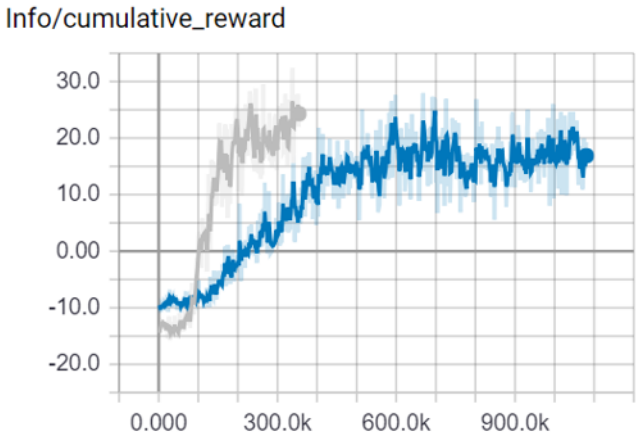
$\longrightarrow$ *other approaches?*

Our Approaches to define Actions

- **Increment of the arm extension:** the same joints mentioned above are moved through small increments of their positions and not by a fixed target position
- **Increment of the position of each joint:** each joint is now independent from the others, hence the movement is still the result of small increments of their positions, but it is not constrained anymore

$\longrightarrow$ *4 degrees of freedom instead of 2*

- *blue line:*
  it corresponds to the agent trained with 4 degrees of freedom
- *grey line:*
  corresponds to the agent trained with 2 degrees of freedom

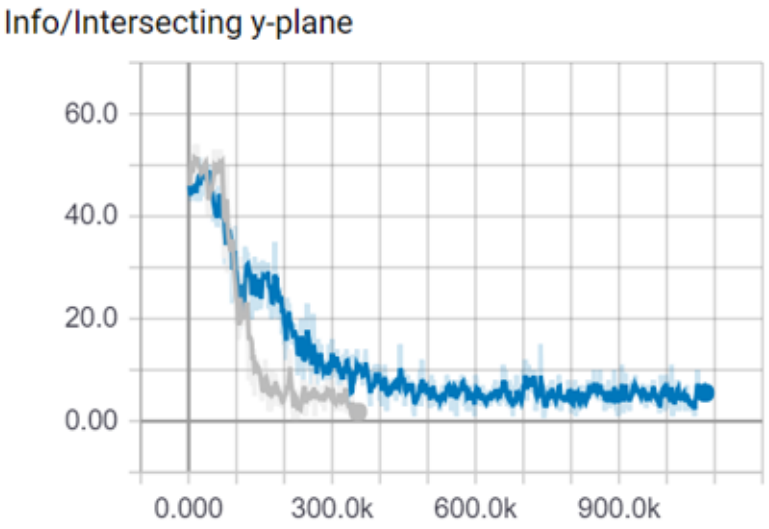


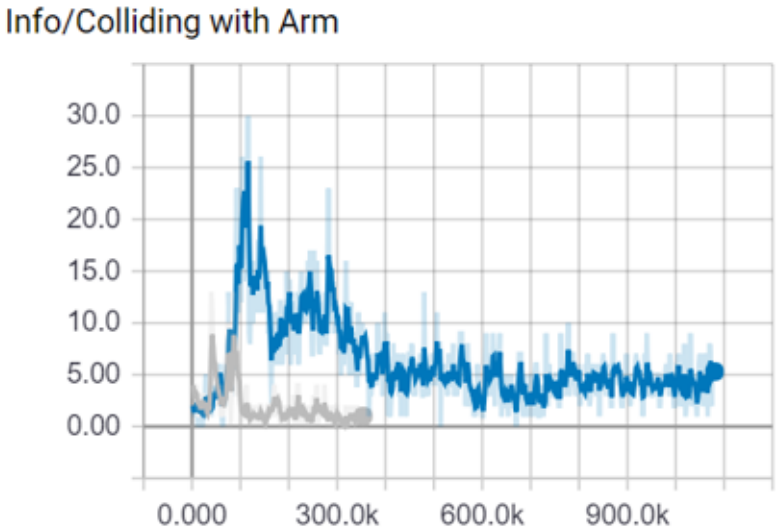
Learning Environment

- **How do we define the Reward?**

Learning Environment

- **How do we define the Reward?**
  - *negative reward* whenever the ball falls (i.e. does not land stable on the Palm)
  - *negative reward* when the Ball collides with anything else than the Palm.



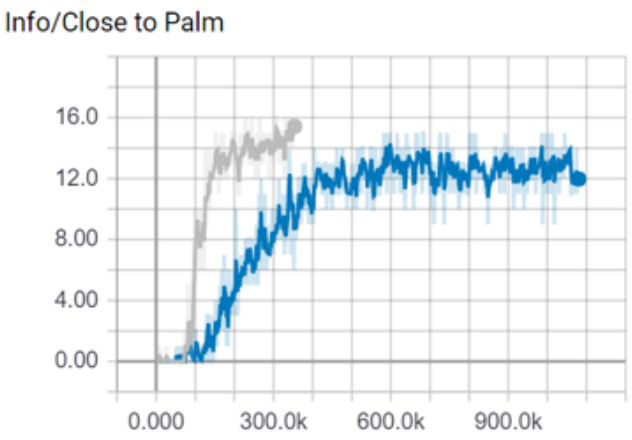


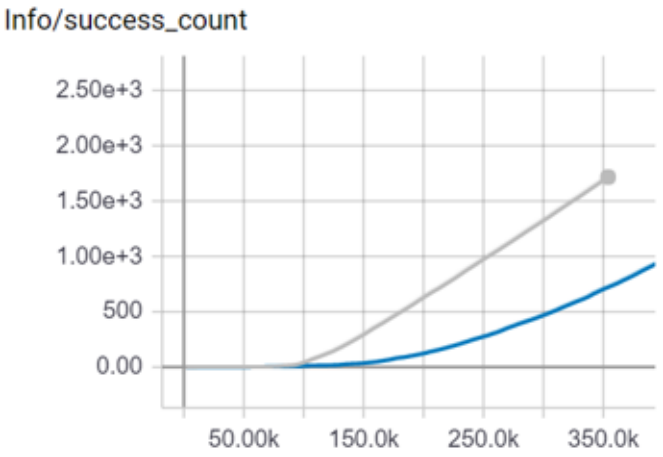
Learning Environment

- **How do we define the Reward?**
  - *negative reward* whenever the ball falls (i.e. does not land stable on the Palm)
  - *negative reward* when the Ball collides with anything else than the Palm.
  - reward proportional to the *negative distance between Ball and Palm* while the Ball is falling

Learning Environment

- **How do we define the Reward?**
  - *negative reward* whenever the ball falls (i.e. does not land stable on the Palm)
  - *negative reward* when the Ball collides with anything else than the Palm.
  - reward proportional to the *negative distance between Ball and Palm* while the Ball is falling
  - *positive reward* whenever the ball is close enough to the Palm (deterministic radius)
  - *positive reward* when the Ball collides with the palm, but only if it's the first time it happens





Our Agents

Agent with 2 degrees of freedom:

Agent with 4 degrees of freedom:

Conclusion