



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

Sim2Real Project

Transfer Reinforcement Learning in Physical Systems

Summer 2022 (May - August)
Under supervision of Professor C. G. Lee

Mechanical & Industrial Engineering Department
University of Toronto

Final Progress Report

Matin Moezzi

Outline



- Simulation Team
- RL Team
- Physical System Team

Simulation Team - Final Report

Tools provided by

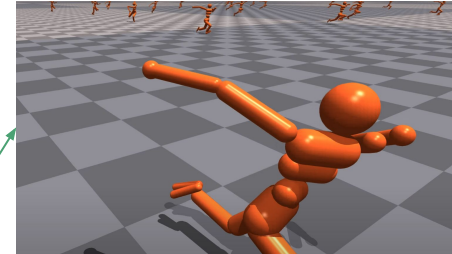


Omniverse Create



Isaac Sim

- Learned from tutorial video list provided by Cobionix
- Held tutorial sessions with Cobionix weekly at the start of this project
- Attend community sharing meeting and be active in Omniverse forum



Isaac Gym (previous ver.)



Isaac Sim (with Isaac Gym integrated)

Omniverse
Version
Update in
June

Simulation Team - Final Report

Omniverse Create Part

- Converted myCobot related STEP files (main body, camera flange and adaptive gripper) using SOLIDWORKS and Fusion 360 to compatible format to Omniverse.
- Assembled converted parts in Omniverse Create, created rigid body and colliders for each link, added joints between adjacent links and assigned driver onto each active and movable joint.
- Overcame the gearing issue of the gripper and created unique environments by every team member.
- Implemented simple control using GUI by add stiffness and damping onto each joint driver and set target position value or target velocity value.



Figure 1: Gripper's movement



Figure 2: Successful Implementation of Joints

Simulation Team - Final Report

Previous Isaac Gym Part

- Setup Linux environment for Isaac Sim and Isaac Gym and took example training environments as reference to build myCobot in random_policy circumstance.
- Rebuilt myCobot with camera flange and adaptive gripper in urdf format as old Isaac Gym did not support USD files yet.
- Revised usd import from example and created multiple instances for myCobot.
 - While urdf have limitation about their stricted tree structure of links, the gripper part need to be simplified.

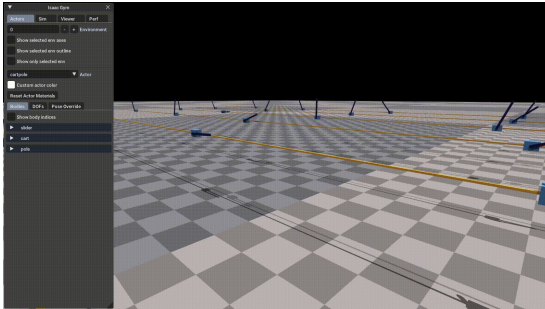


Fig1. Cartpole Example

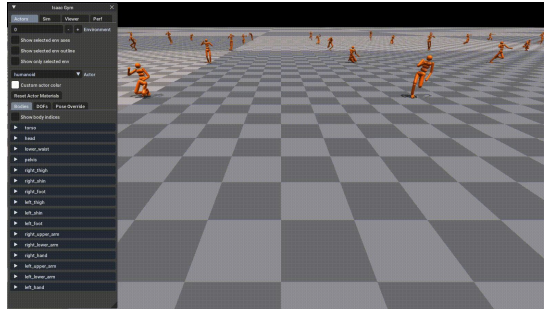


Fig2. Humanoid Example

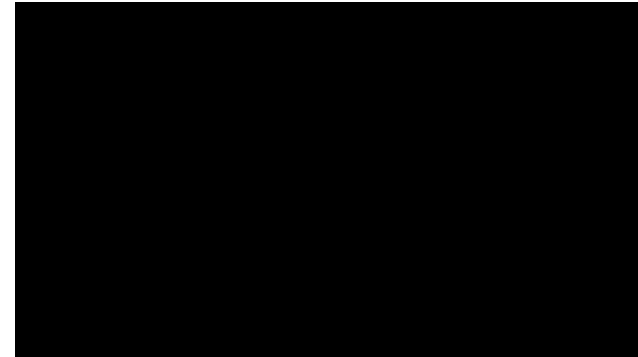


Fig3. Multiple myCobot Instances

Simulation Team - Final Report

Updated Isaac Sim Part - Modelling

- With Isaac Gym integrated into updated Isaac Sim, new docs were released and our team rebuilt myCobot model to fit new features.
 - Key structures with visual xform and collision xform, model became instanceable which will remain only one mesh and lead to less memory usage. (referred Franka Robot structure)
 - Add missing PhysXScene, Articulation root defining to rebuild robot to meet the instanceable model requirement, and modifies the physical property of obey the reality circumstance.
- Add walls to avoid camera interference among different environments
- Added camera after converting instanceable to keep joints active
- Simplified the gripper structure from complex gearing system into just two dependent prismatic joints that only move in one direction.

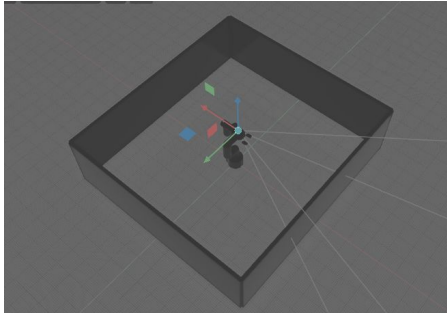


Fig1. myCobot with camera shown



Fig2. Gripper closed status

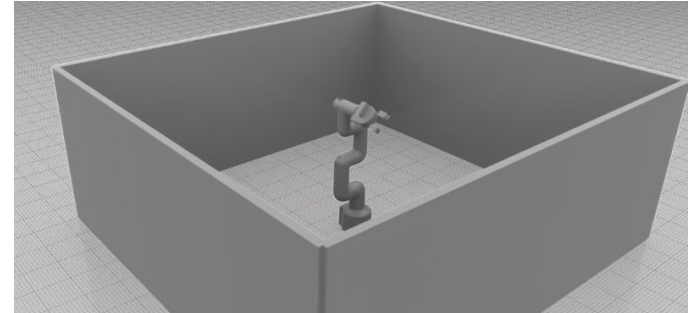


Fig3. Single myCobot Instanceable model

Simulation Team - Final Report

Updated Isaac Sim Part - Scripts



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

- Learned from examples like Cartpole and Shadow hands to be familiar with code structure and related functions to implement a simple RL training environment.
- Created new repository to track the code update and model version update.
- Reduced class inheritance complexity to figure out the basic classes that examples are using.
- Imported myCobot.usd to Isaac Sim with multiple instances successfully, together with camera flange and adaptive gripper.
- Currently limited the angles limit of each joint to keep its movement detectable but not exaggerated.

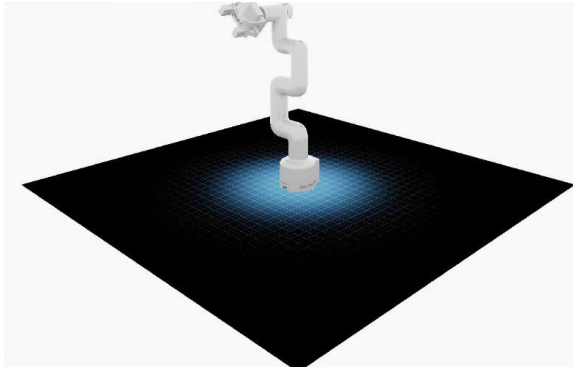


Fig1. myCobot model with gripper imported singly

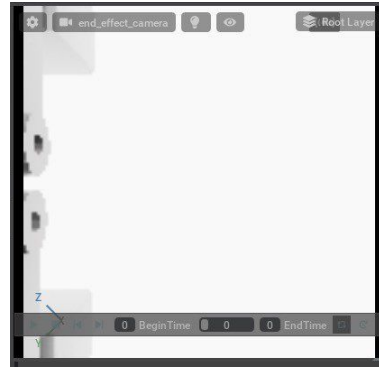


Fig2. RGB camera image

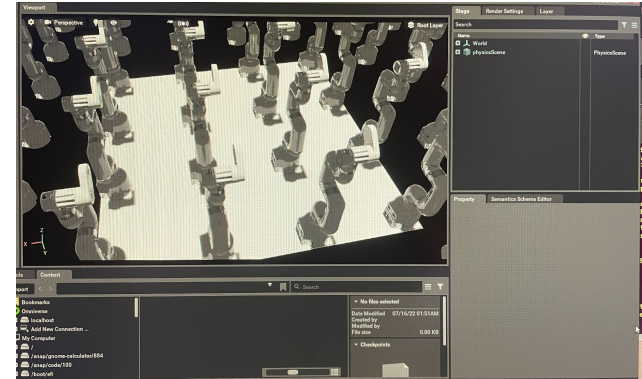


Fig3. 64 myCobot Instances

Simulation Team - Final Report

Updated Isaac Sim Part - Scripts



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

- Added resizing and translation function to ensure models stays above ground and have proper size.
- Added red cube through scripts and could be revised to random position after each round of training.
- Activated the camera in each instances which can be viewed through created viewpoints.
- Successfully implemented the random policy algorithm with myCobot model to test whether all joints, gripper and camera are working correctly.
- Finished get observation function (next state), in which we can get the RGB camera matrix, joints position, joint velocity and etc.
- All interfaces ready for RL.

```
def get_observations(self) -> dict:
    self.env.world.render()
    camera_matrix = []
    # wait_for_sensor_data is recommended when capturing multiple sensors, in this case we can s
    for i in range(self.num_envs):
        gt = self.sd_helper.get_groundtruth(
            ["rgb"], self.viewport_window[i], verify_sensor_init=False, wait_for_sensor_data=0
        )["rgb"][0, :, :3]
        camera_matrix.append(gt)
    dof_pos = self.mycobots.get_joint_positions()
    dof_vol = self.mycobots.get_joint_velocities()
```

Fig1. Available variables in next state

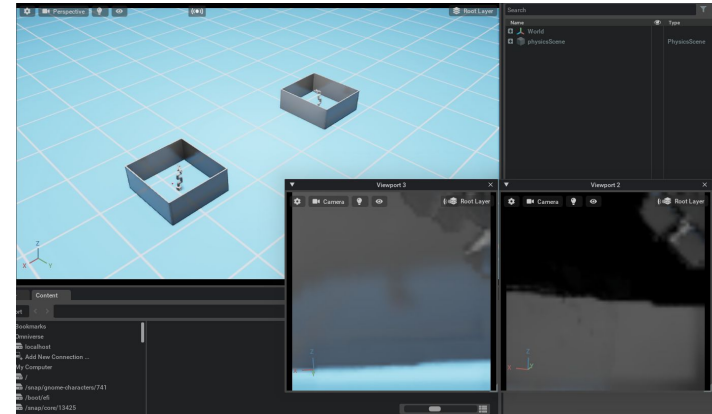


Fig2. Rendered random policy of myCobot

RL team



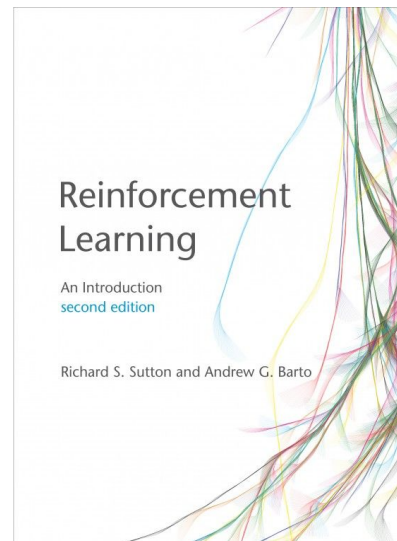
Prepared by: Tingge Zhang & Shawn Zhai

Textbook Reading: *Reinforcement Learning* by Richard S. Sutton and Andrew G. Barto

- Chapter 1: Intro, Chapter 3: Finite MDP, Chapter 6: Temporal-Difference Learning, and Chapter 13: Policy Gradient Methods

Online Research:

- Model-free & model-based RL algorithms
- Lecture on Deep reinforcement learning for robotic control by Human Brain Project
- Algorithms:
 - Deep Deterministic Policy Gradient (DDPG)
 - Trust Region Policy Optimization (TRPO)
 - Proximity Policy Optimization (PPO)
 - Asynchronous Advantage Actor-Critic (A2C)
 - Soft Actor-Critic (SAC)
 - Epsilon Greedy
 - Decaying Epsilon Greedy



RL Team



Prepared by: Tingge Zhang & Shawn Zhai

The team studied Nvidia github repository along with other related repositories, and read corresponding documentations

- Omniverse Isaac Gym Reinforcement Learning Environments for Isaac Sim
<https://github.com/NVIDIA-Omniverse/OmnisaacGymEnvs>
 - Issac Sim Setup
 - Provided Examples (Cartpole, Shadow Hand, etc.)
- RL Games Repository https://github.com/Denys88/rl_games
- Omniverse Python Documentation
<https://docs.omniverse.nvidia.com/py/isaacsim/source/extensions/omni.isaac.gym/docs/index.html>
- How to set up our own tasks
<https://github.com/NVIDIA-Omniverse/OmnisaacGymEnvs/blob/main/docs/framework.md>
- RL Training Examples
https://github.com/NVIDIA-Omniverse/OmnisaacGymEnvs/blob/main/docs/rl_examples.md

The ENV file defines the RL cycle

The class inherits from a base class in the isaac sim library

```
from omni.isaac.gym.vec_env import VecEnvBase
```

```
import torch
```

```
import numpy as np
```

```
# VecEnv Wrapper for RL training
```

```
class VecEnvRLGames(VecEnvBase):
```

```
    def _process_data(self):
```

```
        self._obs = torch.clamp(self._obs, -self._task.clip_obs, self._task.clip_obs).to(self._task.rl_device).clone()
```

```
        self._rew = self._rew.to(self._task.rl_device).clone()
```

```
        self._states = torch.clamp(self._states, -self._task.clip_obs, self._task.clip_obs).to(self._task.rl_device).clone()
```

```
        self._resets = self._resets.to(self._task.rl_device).clone()
```

```
        self._extras = self._extras.copy()
```

```
    def set_task(
```

```
        self, task, backend="numpy", sim_params=None, init_sim=True
```

```
) -> None:
```

```
    super().set_task(task, backend, sim_params, init_sim)
```

```
    self.num_states = self._task.num_states
```

```
    self.state_space = self._task.state_space
```

Initialize Data

Set up task

RL Team

Prepared by: Tingge Zhang & Shawn Zhai



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

Transit to next state

```
def step(self, actions):
    actions = torch.clamp(actions, -self._task.clip_actions, self._task.clip_actions).to(self._task.device).clone()
    self._task.pre_physics_step(actions)
```

Render the frames

```
for _ in range(self._task.control_frequency_inv):
    self._world.step(render=self._render)
    self.sim_frame_count += 1
```

Compute and update parameters

```
self._obs, self._rew, self._resets, self._extras = self._task.post_physics_step()
self._states = self._task.get_states()
self._process_data()

obs_dict = {"obs": self._obs, "states": self._states}

return obs_dict, self._rew, self._resets, self._extras
```

Reset the task and recompute the parameters

```
def reset(self):
    """ Resets the task and applies default zero actions to recompute observations and states. """
    self._task.reset()
    actions = torch.zeros((self.num_envs, self._task.num_actions), device=self._task.device)
    obs_dict, _, _, _ = self.step(actions)

    return obs_dict
```

RL Team

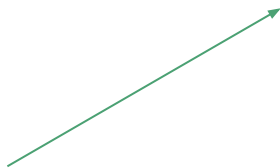
Prepared by: Tingge Zhang & Shawn Zhai



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

```
30 from omniisaacgymenvs.utils.hydra_cfg.hydra_utils import *
31 from omniisaacgymenvs.utils.hydra_cfg.reformat import omegaconf_to_dict, print_dict
32 from omniisaacgymenvs.utils.rl_games.rl_games_utils import RLGPUAlgoObserver, RLGPUEnv
33 from omniisaacgymenvs.utils.task_util import initialize_task
34 from omniisaacgymenvs.utils.config_utils.path_utils import retrieve_checkpoint_path
35 from omniisaacgymenvs.envs.vec_env_rl_games import VecEnvRLGames
36
37 import hydra
38 from omegaconf import DictConfig
39
40 from rl_games.common import env_configurations, vecenv
41 from rl_games.torch_runner import Runner
42
43 import os
44 import torch
45
```

Initialize the rl
games trainer



Run the Trainer



```
46 class RLGTrainer():
47     def __init__(self, cfg, cfg_dict):
48         self.cfg = cfg
49         self.cfg_dict = cfg_dict
50
51     def launch_rl_gym_hydra(self, env):
52         # 'create_rlgpu_env' is environment construction function which is passed to RL Games and called internally.
53         # We use the helper function here to specify the environment config.
54         self.cfg_dict["task"]["test"] = self.cfg.test
55
56         # register the rl-games adapter to use inside the runner
57         vecenv.register('RLGPU',
58                       lambda config_name, num_actors, **kwargs: RLGPUEnv(config_name, num_actors, **kwargs))
59         env_configurations.register('rlgpu', {
60             'vecenv_type': 'RLGPU',
```

```
58         lambda config_name, num_actors, **kwargs: RLGPUEnv(config_name, num_actors, **kwargs))
59         env_configurations.register('rlgpu', {
60             'vecenv_type': 'RLGPU',
61             'env_creator': lambda **kwargs: env
62         })
63
64         self.rl_gym_config_dict = omegaconf_to_dict(self.cfg.train)
65
66     def run(self):
67         # create runner and set the settings
68         runner = Runner(RLGPUAlgoObserver())
69         runner.load(self.rl_gym_config_dict)
70         runner.reset()
71
72         # dump config dict
73         experiment_dir = os.path.join('runs', self.cfg.train.params.config.name)
74         os.makedirs(experiment_dir, exist_ok=True)
75         with open(os.path.join(experiment_dir, 'config.yaml'), 'w') as f:
76             f.write(omegaconf.to_yaml(self.cfg))
77
78         runner.run({
79             'train': not self.cfg.test,
80             'play': self.cfg.test,
81             'checkpoint': self.cfg.checkpoint,
82             'sigma': None
83         })
84
```

RL Team

Prepared by: Tingge Zhang & Shawn Zhai

RLTask Class from Nvidia OmnisaacGymEnvs Repository



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

```
# initialize data spaces (defaults to gym.Box)
if not hasattr(self, "action_space"):
    self.action_space = spaces.Box(np.ones(self.num_actions) * -1.0, np.ones(self.num_actions) * 1.0)
if not hasattr(self, "observation_space"):
    self.observation_space = spaces.Box(np.ones(self.num_observations) * -np.Inf, np.ones(self.num_observations) * np.Inf)
if not hasattr(self, "state_space"):
    self.state_space = spaces.Box(np.ones(self.num_states) * -np.Inf, np.ones(self.num_states) * np.Inf)
```

Classes and methods
from the OpenAI Gym
Github Repository

```
from abc import abstractmethod
import numpy as np
import torch
from gym import spaces
from omni.isaac.core.tasks import BaseTask
from omni.isaac.core.utils.types import ArticulationAction
from omni.isaac.core.utils.prims import define_prim
from omni.isaac.cloner import GridCloner
from omniisaacgymenvs.tasks.utils.usd_utils import create_distant_light
import omni.kit
```

RL Team

Prepared by: Tingge Zhang & Shawn Zhai



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

Space class and Box class

- * They clearly define how to interact with environments, i.e. they specify what actions need to look like and what observations will look like
- * They allow us to work with highly structured data (e.g. in the form of elements of `:class:`Dict`` spaces) and painlessly transform them into flat arrays that can be used in learning code
- * They provide a method to sample random elements. This is especially useful for exploration and debugging.

- The Space class defines the general structure for action, state and observation spaces
- The Box class inherits from the Space class, it takes in 2 numpy arrays (1 for lower bound and 1 for upper bound) that defines the data range of spaces
- It has a sample method that generate a random sample inside the box

RL Team

Prepared by: Tingge Zhang & Shawn Zhai



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

```
self.action_space = spaces.Box(  
    np.array([-1, 0, 0]).astype(np.float32),  
    np.array([+1, +1, +1]).astype(np.float32),  
) # steer, gas, brake
```

The first array `np.array([-1,0,0])` are the lowest accepted values, and the second `np.array([+1,+1,+1])` are the highest accepted values. In this case (using the comment) we see that we have 3 available actions:

1. **Steering:** Real valued in `[-1, 1]`
2. **Gas:** Real valued in `[0, 1]`
3. **Brake:** Real valued in `[0, 1]`

RL Team

Prepared by: Tingge Zhang & Shawn Zhai



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

- Literature Review: Playing Atari Breakout Game with Deep Reinforcement Learning by Mnih, Kavukcuoglu, Silver, Graves, Antonoglou, Wierstra, Riedmiller
 - This article presented a Deep Q-learning model that finds a optimal control policy for playing the Breakout Game.
 - The model can be modified and used in our model to analyze frames captured by the robot's camera in order to develop a policy for picking and placing tasks.



RL Team

Prepared by: Tingge Zhang & Shawn Zhai

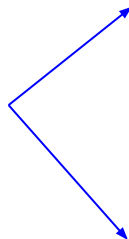


Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

Import OpenAI baselines github repo
and keras library from tensorflow



Set up parameters and environments



```
from baselines.common.atari_wrappers import make_atari, wrap_deepmind
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Configuration parameters for the whole setup
seed = 42
gamma = 0.99 # Discount factor for past rewards
epsilon = 1.0 # Epsilon greedy parameter
epsilon_min = 0.1 # Minimum epsilon greedy parameter
epsilon_max = 1.0 # Maximum epsilon greedy parameter
epsilon_interval = (
    epsilon_max - epsilon_min
) # Rate at which to reduce chance of random action being taken
batch_size = 32 # Size of batch taken from replay buffer
max_steps_per_episode = 10000

# Use the Baseline Atari environment because of Deepmind helper functions
env = make_atari("BreakoutNoFrameskip-v4")
# Warp the frames, grey scale, stake four frame and scale to smaller ratio
env = wrap_deepmind(env, frame_stack=True, scale=True)
env.seed(seed)
```


RL Team

Prepared by: Tingge Zhang & Shawn Zhai




Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

Create layers that convolve filters and rectify input images



Create a model and a target model for Deep Q Learning



```
num_actions = 4

def create_q_model():
    # Network defined by the Deepmind paper
    inputs = layers.Input(shape=(84, 84, 4,))

    # Convolutions on the frames on the screen
    layer1 = layers.Conv2D(32, 8, strides=4, activation="relu")(inputs)
    layer2 = layers.Conv2D(64, 4, strides=2, activation="relu")(layer1)
    layer3 = layers.Conv2D(64, 3, strides=1, activation="relu")(layer2)

    layer4 = layers.Flatten()(layer3)

    layer5 = layers.Dense(512, activation="relu")(layer4)
    action = layers.Dense(num_actions, activation="linear")(layer5)

    return keras.Model(inputs=inputs, outputs=action)

# The first model makes the predictions for Q-values which are used to
# make a action.
model = create_q_model()
# Build a target model for the prediction of future rewards.
# The weights of a target model get updated every 10000 steps thus when the
# loss between the Q-values is calculated the target Q-value is stable.
model_target = create_q_model()
```

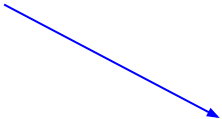
RL Team

Prepared by: Tingge Zhang & Shawn Zhai

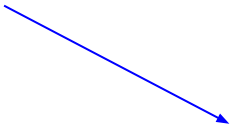


Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

Select actions using decaying
epsilon greedy algorithm and
compute next state and reward



Append the observations to the
replay buffer



```
# Use epsilon-greedy for exploration
if frame_count < epsilon_random_frames or epsilon > np.random.rand(1)[0]:
    # Take random action
    action = np.random.choice(num_actions)
else:
    # Predict action Q-values
    # From environment state
    state_tensor = tf.convert_to_tensor(state)
    state_tensor = tf.expand_dims(state_tensor, 0)
    action_probs = model(state_tensor, training=False)
    # Take best action
    action = tf.argmax(action_probs[0]).numpy()

# Decay probability of taking random action
epsilon -= epsilon_interval / epsilon_greedy_frames
epsilon = max(epsilon, epsilon_min)

# Apply the sampled action in our environment
state_next, reward, done, _ = env.step(action)
state_next = np.array(state_next)

episode_reward += reward

# Save actions and states in replay buffer
action_history.append(action)
state_history.append(state)
state_next_history.append(state_next)
done_history.append(done)
rewards_history.append(reward)
state = state_next
```

RL Team

Prepared by: Tingge Zhang & Shawn Zhai



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

Update the target model every 4 frames for future Q value prediction

```
# Update every fourth frame and once batch size is over 32
if frame_count % update_after_actions == 0 and len(done_history) > batch_size:

    # Get indices of samples for replay buffers
    indices = np.random.choice(range(len(done_history)), size=batch_size)

    # Using list comprehension to sample from replay buffer
    state_sample = np.array([state_history[i] for i in indices])
    state_next_sample = np.array([state_next_history[i] for i in indices])
    rewards_sample = [rewards_history[i] for i in indices]
    action_sample = [action_history[i] for i in indices]
    done_sample = tf.convert_to_tensor(
        [float(done_history[i]) for i in indices]
    )

    # Build the updated Q-values for the sampled future states
    # Use the target model for stability
    future_rewards = model_target.predict(state_next_sample)
    # Q value = reward + discount factor * expected future reward
    updated_q_values = rewards_sample + gamma * tf.reduce_max(
        future_rewards, axis=1
    )
```



Literature Review on papers of Robot arm pick & drop tasks in the robot community:

- Algorithm: Deep Q-learning is commonly used for discrete action space, Deep Deterministic Policy Gradient (DDPG) and Trust Region Policy Optimization (TRPO) are used for continuous action space
- State Space:
 - For discrete algorithms, they turn raw RGB-D images into 3D point cloud or height maps as the state input
 - For continuous algorithms, they use position and angular velocity of the joints on the robot arm along with the x, y, z position of the object as the state input

RL Team

Prepared by: Shawn Zhai & Tingge Zhang



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

- Action Space:
 - Actions are categorized as “pushing” and “grasping”
 - The pushing action is performed by moving the gripper horizontally for a certain distance, directions are divided into 16 intervals ($360^\circ / 16 = 22.5^\circ$)
 - For continuous action space, it is typically defined by the changes in position and velocity of the robot’s actuators (actuated joints).
 - To detect if a object is successfully grasped, some papers use sensors and visual indicators, and some other paper make the gripper perform the closing action again after grasping, if the gripper is still in the opening state, that means the object is being held by the gripper

RL Team

Prepared by: Shawn Zhai & Tingge Zhang



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

- Reward Function:
 - #1: Use distance between the gripper and the object
 - 3D Distance is approximated using 2D position captured by the camera
 - Penalize the distance from the object to the gripper
 - #2: Assign a small positive reward for pushing actions if it makes it easier for grasping in the future
 - If the difference between the current and previous D heightmap or 3D point cloud is greater than the customized threshold, the action is considered having a positive effect on the following grasping work

RL Team

Prepared by: Shawn Zhai & Tingge Zhang



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

- #3: Apply small negative reward to all steps prior to termination to encourage a faster grasping action.
- #4: Use piecewise reward function to avoid sparse reward which leads to slow or non-convergence in model training.
 - Introduces the idea of “pixel change”
 - Divide the pixel change percentage into several intervals And assign reward accordingly
 - The paper does not define “pixel change” and its advantages, here’s our inference:
if the frame has 100 pixels and 60 pixels in the previous frame appears again in the current frame, then the pixel change will be 40%. It is used as a way to prevent sparse reward and also likely to promote exploration.

$$R = \begin{cases} 1 & \text{grasp successfully} \\ -1 & \text{grasp failed} \\ 0.3 & \text{pixel change } \tau (10\% \sim 24\%) \\ 0.5 & \text{pixel change } \tau (24\% \sim 40\%) \\ 0.7 & \text{pixel change } \tau (40\% \sim 100\%) \\ -0.1 & \text{otherwise} \end{cases}$$

RL Team

Prepared by: Shawn Zhai & Tingge Zhang

- A single RGB-D camera set at a fixed location in the workspace is commonly used for capturing the states
- Setting cameras from dual viewpoint (set cameras both within and outside the grasping area) is helpful for ensuring no information is missing and the complete formation of object is captured



- Affine Transformation can be used to determine 3D Distance using 2D position captured by the camera

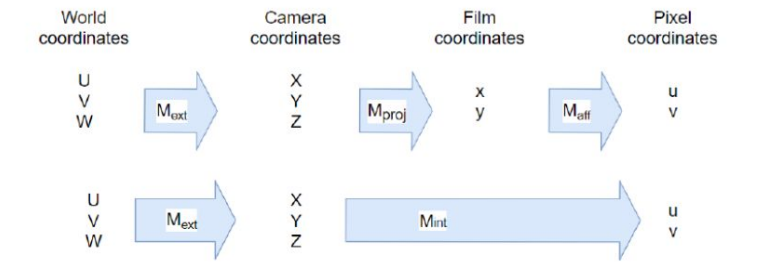
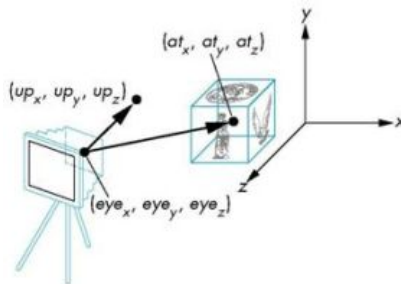


Figure 5. Forward projection.

Physics system Team - Final Report

Presented by: Erry Guan



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

- Documented the myCobot 280 Pi related material
 - About mycobot API, install software, links, resources, ect
- Controlled myCobot in different methods
 - Used Blockly to proceed joint control and gripper status control
 - Used Roboflow to make joint function examination
 - Used Python API - pymycobot to perform high level scripts control
- Established remote connection and virtual desktop to myCobot using VNC server



Fig1. Roboflow interface

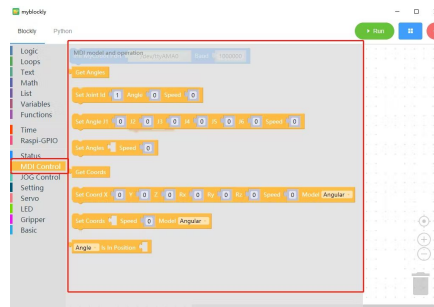


Fig2. myblockly interface

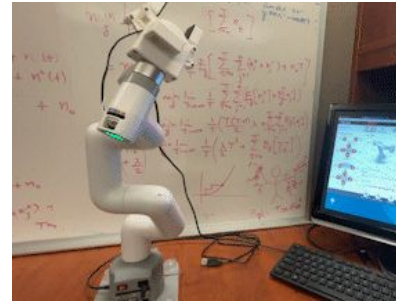


Fig3. Roboflow control

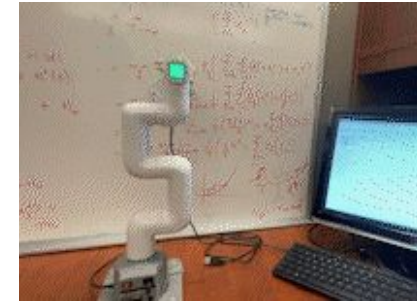


Fig4. myblockly control

Physics system Team - Final Report

Presented by: Erry Guan



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

- Upgrade myCobot Raspberry Pi performance by reinstalling the OS to activate not running core.
 - Camera's frame per second got improved significantly
- Installed camera flange and adaptive gripper onto the main body
- Tried different object detection to detect dice
 - Implemented the "EfficientNet" and "YOLO" models using Tensorflow Lite, which can get phone class and key class in pre-trained model instead of dice
 - Based on opencv-python, applied HSV on greyscale analysis to isolate dice from background

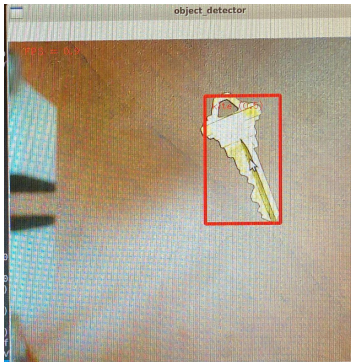


Fig1. Key class detection

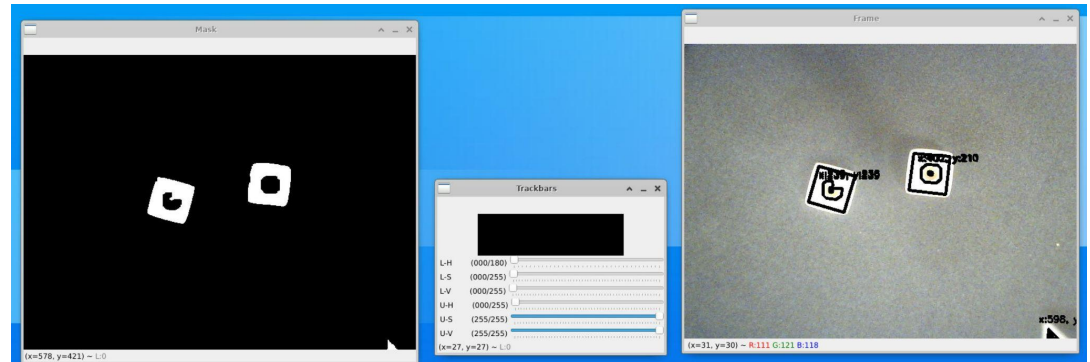


Fig2. dice detection in opencv-python

Physics system Team - Final Report

Presented by: Erry Guan



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

Pick and Place Task

Object detection

Opencv-python findContours method

Transformation matrix

Transformed the object coordinates from camera frame to robot end-effector local frame to world coordinates which was defined at the base of myCobot

Joint Control

Input end-effector destination and control drivers to corresponding angles

Gripper Control

Close gripper to grab the dice

Back to Origin

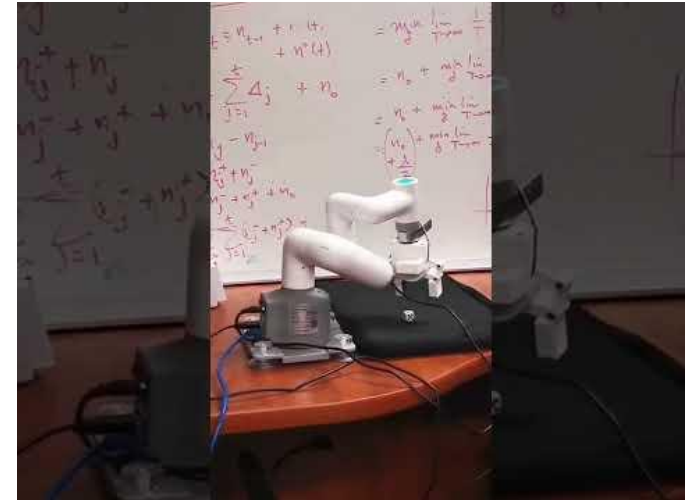
Move to Destination

Gripper Open

Back to Origin

Pick part

Place part



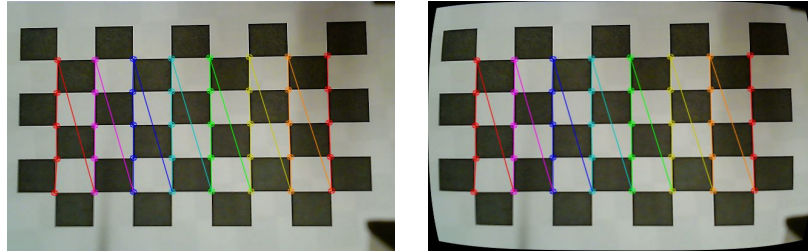
Physics system Team - Final Report

Presented by: Erry Guan



Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

- Determined the action space based on available joint range
- Camera calibration and distortion using chessboard calibration
- RL training scripts API have been completed for physics system



Currently limitation of Pick and Place:

- Long waiting time to satisfy motor calibration
- Object detection angle limitation for monocular camera, need more research to figure out how to implement this improvement



Thank you for your attention