Reinforcement Learning for Human Intention Anticipation in Collaborative Tasks

Pedro Miguel Loureiro Amaral

July 15, 2024

1 Introduction

Human-Robot Collaboration (HRC) is a research topic becoming increasingly important in modern industry, driven by the need to enhance productivity, efficiency, and safety in work environments. The combination of human skills and robotic capabilities provides significant potential to improve the execution of complex and repetitive tasks. However, effective synchronization of actions and seamless communication between partners are open challenges that need to be further addressed.

In collaborative scenarios, assistive robots are designed to work alongside humans in assembly processes or maintenance operations, providing timely support to enhance the overall efficiency of the task. Robots can assist the human worker by delivering a component, tool, or part, by holding a part while the operator works on it, or by performing autonomously a specific sub-task. In any case, the ability of an assistive robot to anticipate the upcoming needs of a human operator plays a pivotal role in supporting efficient teamwork. By anticipating human intentions, actions, and needs, robots can proactively assist or complement human tasks, providing timely support and improving overall efficiency.

In summary, the main objectives of this work are the following:

1. Optimize existing machine learning models to recognize the objects being grasped by the user by using the right-hand keypoints for real-time applications. Develop a ROS package that integrates the optimized models with the robot controller to provide the necessary information for the robot to anticipate the human partner's intentions.

2. Develop a reinforcement learning model to plan the robot's movements in real-time. This includes the development of a simulator that resembles the real-world scenario, the training of the model, and the integration of the model with the real robot controller.

3. Develop a robot controller that is able to control the robot's movements using velocities instead of positions to allow for smoother and more natural movements.

2 Experimental Setup

The work described in this dissertation is part of the AUGMANITY project[1] that aims to develop technologies to support people operating in industrial environments. The experimental setup comprises the integration of both hardware and software components in a prototype collaborative cell (LARCC) at the Laboratory for Automation and Robotics (LAR) located in the Department of Mechanical Engineering at the University of Aveiro, as illustrated in Figure 1. The LARCC is equipped with a UR10e collaborative robot and multimodal sensor devices, including three LiDAR Velodyne sensors and four Orbbec 3D cameras distributed throughout the work volume. The software architecture is built upon the ROS middleware [1], providing a robust framework for communication and coordination among the various components. In this context, this

[1] AUGMANITY website: `https://www.augmanity.pt`

section provides a description of the materials used during this study and the methodological approaches followed to face the key challenges.

Figure 1: Prototype collaborative cell LARCC

2.1 Robot Operating System (ROS)

ROS[2][2][3] is an open-source collection of tools and software libraries used to develop a robotics application and, in this work, it is used to establish communication throughout all of the infrastructure. ROS was chosen due to the hardware abstraction it offers given that it contains driver packages to deal with some hardware devices, allowing for easier communication with the robot and the cameras. Other relevant features include:

- **message broker**: every process in the project is a node in the ROS network and communicates with the other nodes mainly through topics (asynchronous publish/subscribe streaming of data) or services (synchronous RPC-style communication).
- **code reuse**: executables and packages are written to be as independent as possible, making the developer able to reuse them in another project.
- **rich ecosystem**: there are several open-source packages available to the developer that can be easily integrated.
- **scalability**: given that the nodes are so loosely coupled, it allows for node distribution.
- **language independence**: nodes can be written in any language since communication is established through well-defined objects.
- **data visualization**: there are tools to visualize the data and the functioning of the system in real-time, such as Rviz.

[2]ROS 1 documentation: `https://wiki.ros.org`
[3]ROS 2 documentation: `https://docs.ros.org/en/humble`

- **simulator support**: ROS has support for simulators with Gazebo being the most common.

For this system, the ROS 1 Noetic distribution was chosen over the more recent ROS 2 distributions so as to take advantage of work already done by other members of the AUGMANITY project at the University of Aveiro.

2.2 Perception System

In order to capture the necessary information from the environment, two Orbbec Astra Pro RGBD cameras were used. This camera model was developed by Orbbec Technologies and it is frequently used in computer vision and robotics [3]. Among the available cameras, it was chosen since it allowed to capture both color and depth images.

Figure 2: Orbbec Astra Pro [3]

In the experimental setup, one of the cameras is placed above the workspace facing downwards allowing the perception of position of the user's hand through the color and depth images. The second camera is above and slightly behind the robot to capture the user in front of the robot with the images from this camera being the ones used in the keypoint detection models. The communication with the cameras is established through ROS with the *usb_cam* package being used for the color image and the *ros_astra_camera* package being used for the depth image.

The camera calibration was done using the *camera_calibration*[4] package for the intrinsic parameters and the *atom*[5] package for the extrinsic parameters. The calibration process was done by capturing images of two charuco boards placed in the workspace.

2.3 Manipulator Arm Control

The collaborative robot available for this work is a UR10e model which was developed by Universal Robots. This model has six degrees of freedom with six rotational joints, allows for payloads up to 12.5 kg, and has a reach of 1300 mm being suitable for tasks such as machine tending, palletizing, and packaging[4]. In this work, the robot is equipped with a 2F-140 gripper developed by Robotiq, commonly used together with robot models from Universal Robots[5].

Both the robot and the gripper have ROS packages containing their drivers making their integration easier. The planning and execution of the arm movements are done through the MoveIt[6] framework, which is a widely-used open-

[4] *camera_calibration* package wiki page: `https://wiki.ros.org/camera_calibration`
[5] *atom* package documentation: `https://lardemua.github.io/atom_documentation/`
[6] MoveIt documentation: `https://ros-planning.github.io/moveit_tutorials`

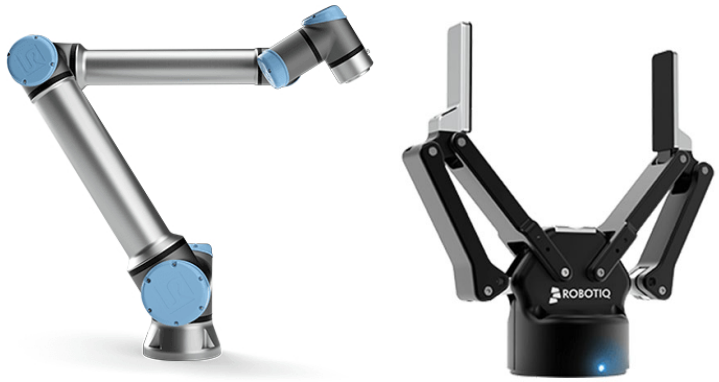


Figure 3: UR10e Collaborative Robot [6] and Robotiq 2F-140 Gripper [5]

source framework for robotics applications involving motion planning, manipulation, 3D perception, kinematics, control, navigation, and collision checking, with OMPL being chosen to handle the motion planning tasks.

The configurations of the drivers and MoveIt was already done by other members of the AUGMANITY project and can be found on Github[7] along with a higher-level API that encloses that logic.

In the last part of this work, the *ur_rtde*[8] package was used to control the robot's movements using velocities instead of positions. This package allows to do everything that can be done with teach pendant connected to the robot, such as moving the robot and even turning on and off the free drive mode.

2.4 Computational Systems

The tasks involved in this work, such as training deep-learning models and analyzing images in real-time require high computational resources. To handle the real-time processing of images and robot control, the central computer present in the setup was used. To handle the deep-learning model training, the deep-learning research server from LAR was used, codenamed Deeplar:

- AMD RyzenTM Threadripper 2950X;
- Four NVIDIA GEFORCE® RTX 2080 Ti;
- 128GB DDR4 RAM.

The model training in Deeplar is executed using docker images, which allows multiple people to use the computer with each having their own isolated training environment with their own dependencies. The images used to design and train machine learning models in this work are based on the latest TensorFlow official image for GPUs.

TensorFlow is one of the most popular machine learning frameworks along with Pytorch. In this work, the former was chosen over the latter since the higher-level API allowed for faster development. The main features of Tensorflow[9] are:

[7]Github LarCC Repository: `https://github.com/lardemua/larcc_drivers`
[8]UR RTDE Documentation: `https://sdurobotics.gitlab.io/ur_rtde/index.html`
[9]Tensorflow documentation: `https://www.tensorflow.org/api_docs`

- **prepare data**: load data, data pre-processing and data augmentation;
- **build models**: design and train custom models with little code or use pre-trained ones (transfer learning);
- **deploy models**: helps using models in different platforms such as locally, in the cloud, in a browser, or in mobile;
- **implement MLOps**: run models in production, tracking their performance and identifying issues.

2.5 Keypoints Detection with MediaPipe

MediaPipe[10] consists of a set of libraries and tools to apply AI and Machine Learning techniques in other applications, particularly in pipelines for advanced real-time vision-based applications [7]. Although it contains many features, in this work the focus is on the Hand and the Pose Landmark Detection Models. The Hand Landmark Detection model [8] uses two sub-modules: a hand palm detection model and a hand landmark model. Each frame of an RGB input is fed into the palm detection model, which produces a bounding box based on the palm. The hand landmark model uses this bounding box and returns the keypoint localization of 21 landmarks, including the fingertips, the finger joints (knuckles), and the base of the palm (Figure 4a). The Pose Landmark Detection model also uses two sub-modules working in a similar way to return 33 landmarks over the entire body (Figure 4b).

(a) (b)

Figure 4: Mediapipe landmarker models [9]: (a) Hand Landmarker and (b) Pose Landmarker.

2.6 RL Simulator

Gymnasium and Mujoco

[10]MediaPipe documentation: `https://developers.google.com/mediapipe`

3 Learning-Based Recognition of Human-Grasped Objects

3.1 Proposed Approach

This work proposes a learning-based framework to enable an assistive robot to recognize the object grasped by the human operator. As illustrated in Figure 5, the proposed framework combines the strengths of MediaPipe in detecting hand landmarks in a RGB image with a deep multi-class classifier that predicts the object based on the configuration of the user's hand after grasping it. Accordingly, the developed object recognition system operates based on different principles, including the sensing device, the tracking method, and the machine learning approaches. From the point of view of the application in industrial settings, the proposed system has two strengths when compared to the use of data-gloves or electromagnetic motion capture systems. First, the simplicity of installation is associated with a much less complex and costly setup. Second, the non-intrusiveness of the required setup is a valuable factor in accelerating the acceptance of these technologies by humans in carrying out collaborative tasks (a process also referred to as "user adoption"). In contrast, vision-based hand tracking is affected by occlusions, changes in light conditions, and cluttered backgrounds. Furthermore, these problems are difficult to overcome with deep-learning techniques given the data dependency and generalization problems against hands, objects, and lighting conditions outside the training sets. Overall, this paper contributes to advances in understanding the opportunities and limitations of using this novel approach for the recognition of human-grasped objects.

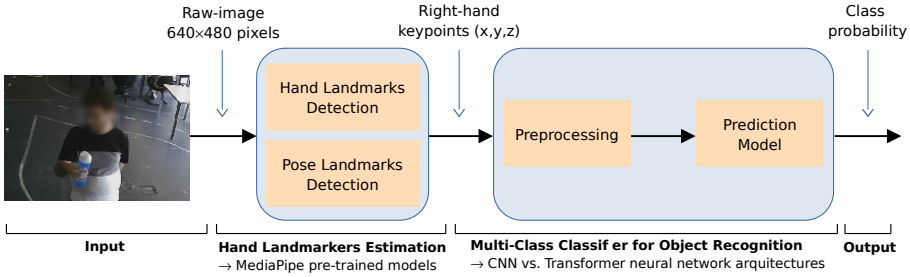


Figure 5: The proposed learning-based framework for object recognition based on the hand keypoints.

The output of the pre-trained model provides the (x, y, z) coordinates of landmarks for each detected hand. The (x, y) coordinates represent the horizontal and vertical positions of the landmark on the image plane, while the z-coordinate represents an estimate of the relative depth with respect to the wrist reference [10]. This work focuses on tracking the right hand by combining the Hand Landmark detection and the Pose Landmark Detection pre-trained models. This strategy proved to be useful to enhance the reliability of the process of extracting the coordinates of the right-hand keypoints from each frame.

The multi-class classifier for object recognition faces several challenges. First, there is limited information about the three-dimensional configuration of the

hand, namely if the hand configurations involve overlapping fingers or positions close to each other in the image plane. Consequently, the z-coordinate (relative depth) revealed to be a critical element for discriminating complex hand configurations. Second, the coordinates provided by MediaPipe can vary in scale and rotation depending on the hand's distance from the camera and the hand's orientation in the image, adding complexity to the task. For these reasons, a deep learning model able to learn complex features directly from the MediaPipe coordinates will be explored and evaluated with a view to its generalization ability in different scenarios and for various users.

3.2 Data Representation

The four everyday objects selected for this study are all "graspable", i.e., more or less rigid. They include a cylindrical water bottle, a Rubik's cube, a plier, and a small and sharp screwdriver (Figure 6). Given the differences in shape, size, and/or weight, the goal is to discriminate these four objects based on the configuration adopted by the hand while interacting with them.

Figure 6: The objects used in the study include a water bottle, a Rubik's cube, a plier, and a screwdriver.

Dataset Acquisition

For this particular problem the dataset was manually collected, consisting of videos where one person would move and rotate a particular object (example frames in Figure 7). This acquisition involved the participation of three right-handed (male) volunteers aged between 23 and 26 years old. Participants were asked to naturally grab and hold an object placed on a table, followed by executing small movements of the hand in free space. These movements were performed while introducing random variations in the hand's orientation relative to the RGB camera to ensure diversity in the points of view from which the hand-object interaction is observed.

Figure 7: Dataset examples holding a bottle (left) and a phone (right).

Naturally, the successive frames could lead to similar grasping patterns from different views. To investigate intra-user variability and to ensure robust model training, users are instructed to perform multiple grasping trials of the selected object across four distinct acquisition sessions. Bearing this in mind, the data acquisition system was designed to facilitate the fast generation of training datasets, accommodating the inclusion of new users and/or additional acquisition sessions. On the one hand, the system is integrated into the workflow of the proposed object recognition framework. On the other hand, it is particularly well-suited for implementation in industrial settings where end-users may not possess extensive expertise in machine learning or computer vision. The instructions provided to users during the data acquisition sessions were intentionally straightforward, ensuring that non-experts could readily participate in the process.

Videos over four sessions per user were recorded at 10 frames per second. For each object and each user, four data acquisition sessions were carried out, which gave rise to the dataset used in the study. Therefore, the dataset consists of a total of 11 054 samples, distributed practically equally across the three participants (around 3600 samples per participant) and the four objects (between 2618 and 2849 samples per object). The exact number of samples of the entire dataset per class and per user is shown in Table 1.

Table 1: Number of samples in the dataset per class and user

Dataset	Bottle	Cube	Phone	Screwdriver	Total
User1	828	928	950	957	3663
User2	886	926	939	946	3697
User3	904	907	937	946	3694
Total	2618	2761	2826	2849	11 054

Preprocessing

After having a dataset, the data had to be processed to have a fitting structure to be used in the model training. The images from the videos were processed using the Mediapipe hands model resulting in 21 points for each hand detected (Figure 8).

Figure 8: Points detected on the pictures in Figure 7 by Mediapipe Hands Model.

The points corresponding to the right hand are then subject to further transformations and normalization. First, the original coordinates of the keypoints (raw data), which are already normalized within the range of 0 to 1 are converted into coordinates relative to a reference. Specifically, for each keypoint $P = (x, y, z)$, the coordinates of the reference point are subtracted $P_{ref} = (x_{ref}, y_{ref}, z_{ref})$ from them to obtain relative coordinates $P_{rel} = (x_{rel}, y_{rel}, z_{rel})$. In this study, the reference is defined as the centroid C of the set of hand keypoints. This transformation into relative coordinates is particularly useful because the absolute position of the hands in the image may vary from frame to frame due to different distances from the camera or hand orientations. Instead, relative coordinates are translation invariant and they reduce the influence of any rotations that might be present in the raw data. Therefore, the network will focus on the spatial relationships between keypoints, rather than their absolute positions, making it less sensitive to hand orientations and scale variations.

After obtaining the relative coordinates with respect to the reference point, scaling is applied to each dimension independently by dividing by an appropriate constant to ensure that the hand's representation spans the entire range, as follows:

$$scaleFactor = \frac{0.5}{\max(\{|x_i|, |y_i|, |z_i|\} : i = 1, \cdots, n)} , \tag{1}$$

where $\{x_i, y_i, z_i\}$ denote relative coordinates. This feature scaling revealed to be a valuable pre-processing step to help make the data more consistent, helping the model to learn the relevant patterns without being influenced by variations in hand position, hand size, or scale. Further, it helps to maximize the separation among keypoints, helping the model to discriminate the output class. Finally, a uniform adjustment is made by adding 0.5 to each coordinate, centering the points between 0 and 1 on the scale. It is important to note that throughout the point processing, the order of the points is never changed, and therefore the models can take advantage of this structure. Figure 9 shows examples of the normalized keypoints representation expressed according to the previous steps, that is:

$$P_{norm} = (P - C) \times scaleFactor + 0.5 . \tag{2}$$

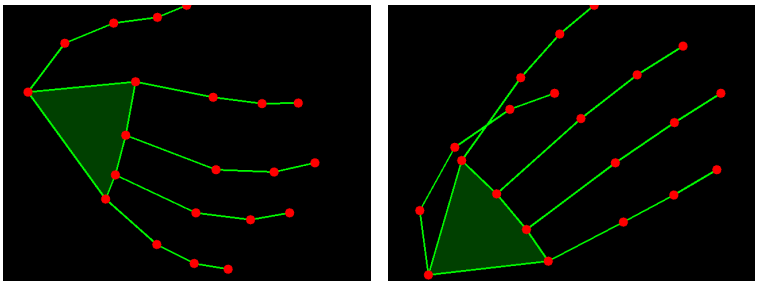


Figure 9: Points from the pictures in Figure 8 after normalization

3.3 Models

For this work, the architectures tested were the CNN and the Transformer given their ability at detecting local relations, which makes them an advantageous solution for this problem given that each sample provided to the model is made of the 21 3D points that always follow the same structure representing the right hand. For further details... (cite article)

CNN Classifier

The developed CNN comprises two convolutional layers each with 64 feature maps and ReLU activation functions. The first layer uses a kernel size of 3×3 pixels performing a 1D convolution on the 3×21 data with a stride of 1 pixel. The flattened output from the final layer is connected to a dense layer with 128 neurons, followed by another dense layer with the number of neurons equal to the number of classes. The output layer consists of the final connect layer with softmax activation. The softmax function takes a vector of real-valued scores (often called logits) and transforms them into a probability distribution over multiple classes. For the classification task with 4 classes, the output layer has 4 neurons, each representing the probability of the input belonging to a particular class. To prevent overfitting, dropout layers are incorporated after each fully connected layer. The final model can be seen in Figure 10 and it has 156 644 trainable parameters. It is made of two convolutional layers followed by three dense layers, with the third being the output layer. Between the convolutional and the dense layers and between both dense layers, there is also a dropout layer to help with overfitting.

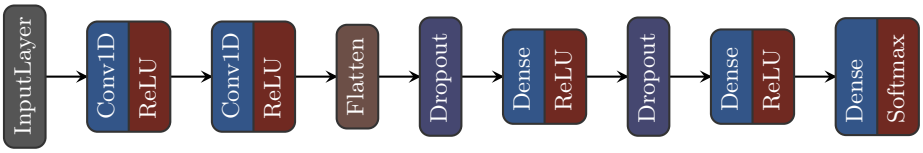


Figure 10: CNN model architecture

Transformer Neural Network Classifier

The developed Transformer model is made of two Transformer encoder stacks (Figure 11) comprised of the following layers: multi-head self-attention, layer

normalization, and feedforward neural networks. Within each encoder, multi-head self-attention is applied to capture dependencies among the keypoints, where four attention heads are used for enhanced feature extraction. Following self-attention, two position-wise feedforward neural networks are employed to process the attended features and capture complex patterns. Layer normalization is applied after each sub-layer to stabilize the activations and facilitate training convergence. The resulting architecture can be seen in Figure 12 and it has 16 384 trainable parameters.

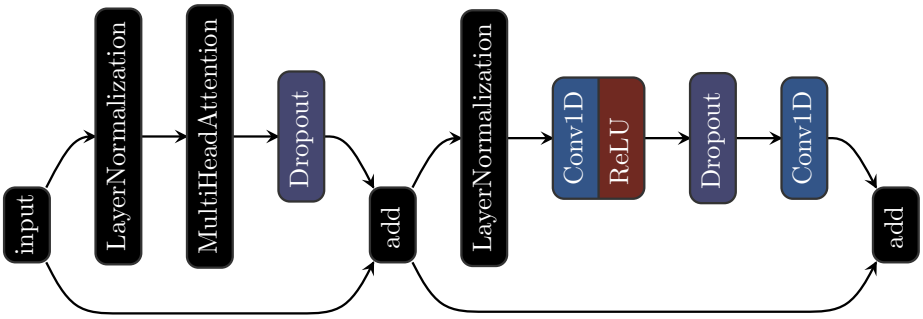


Figure 11: Transformer encoder block

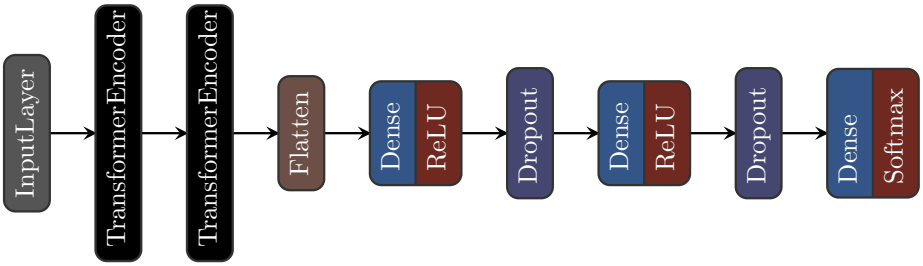


Figure 12: Transformer model architecture

3.4 Model Comparison

With both models trained, Table 2 shows the performance of all the machine learning models involved in this classification task, including those belonging to MediaPipe.

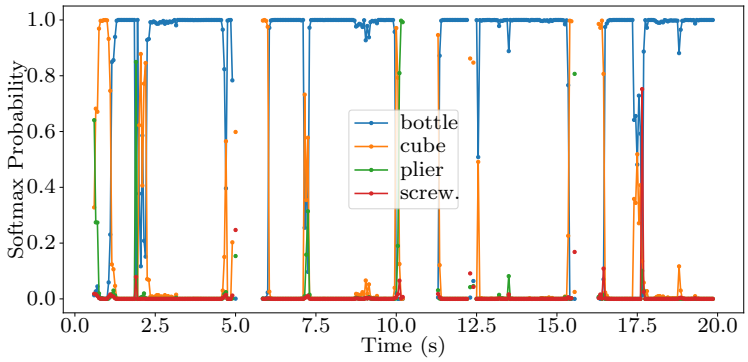
Table 2: Model Comparison

Model	Performance	Prediction Time (ms)
Hand Landmarker	10.09 MNAE*	43.5
Pose Landmarker Lite	87.0 PDJ*	21.8
Pose Landmarker Full	91.8 PDJ*	28.7
Pose Landmarker Heavy	94.2 PDJ*	83.1
CNN Object Classifier	90.5% Acc*	14.1
Transformer Object Classifier	86.7% Acc*	16.0

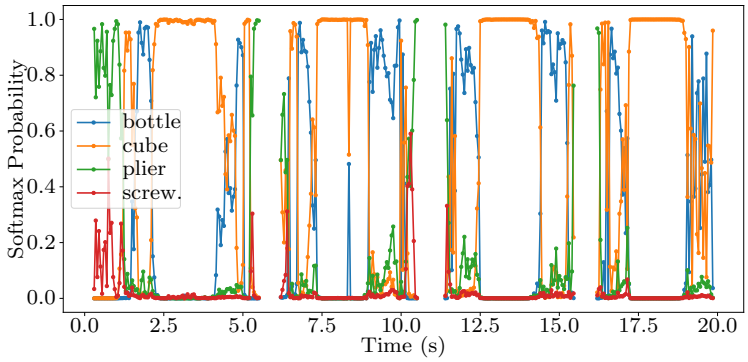
*MNAE: Mean of Normalized Absolute Error
*PDJ: average Percentage of Detected Joints
*Acc: model Accuracy

For the Hand Landmarker, there is only a single option available so it was the one used. Then, for the Pose Landmarker there are three options available but given that it has relatively less relevance and it runs in parallel with the Hand Landmarker, the Full version was used since it has the highest performance while keeping a lower prediction time. Finally, for the object classifier, the CNN presented the highest accuracy and the lowest prediction time so it was selected for further optimizations in real-time.

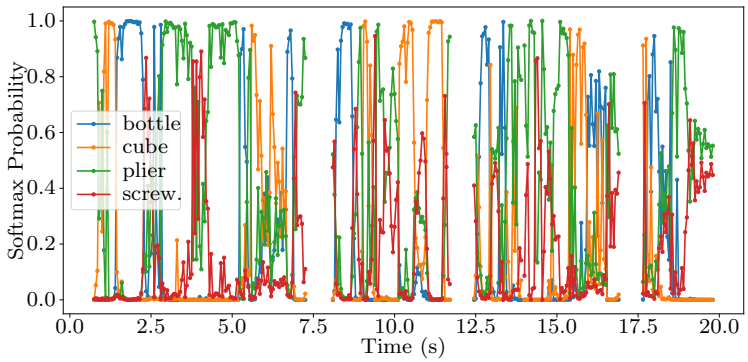
With all models of the pipeline selected, a test was made in real-time for each object to check the stability and reliability of the model prediction in a certain frame, resulting in the graphics in Figure 13.



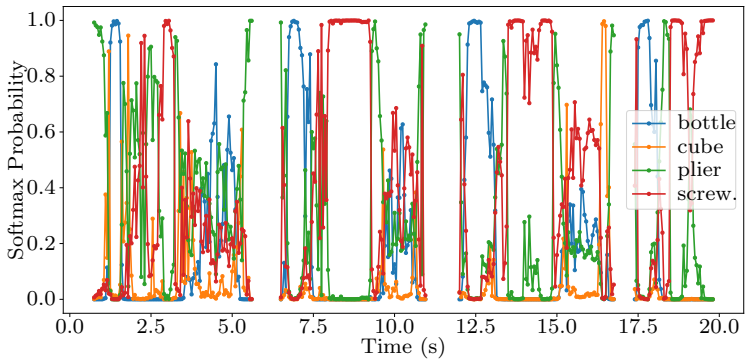
(a)



(b)



(c)

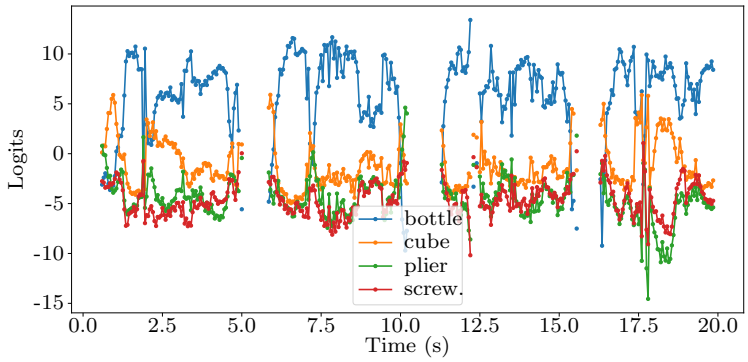


(d)

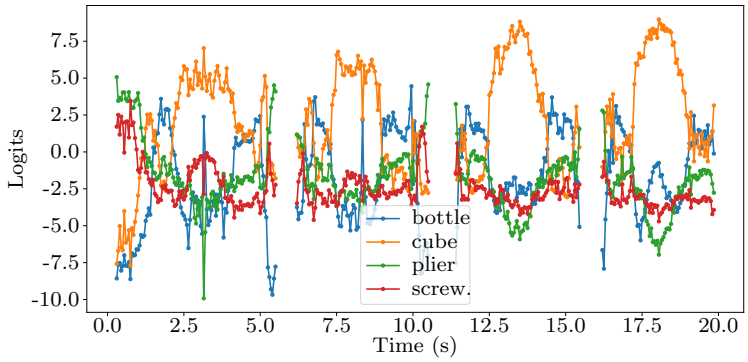
Figure 13

These results show that there are a significant number of frames where the model outputs a softmax probability equal or very close to 1.0 about the wrong label, especially in the video where the user is holding a plier. Furthermore, these frames are not isolated, sometimes the error happens over multiple consecutive frames making it even harder to establish a rule about when a prediction is valid.

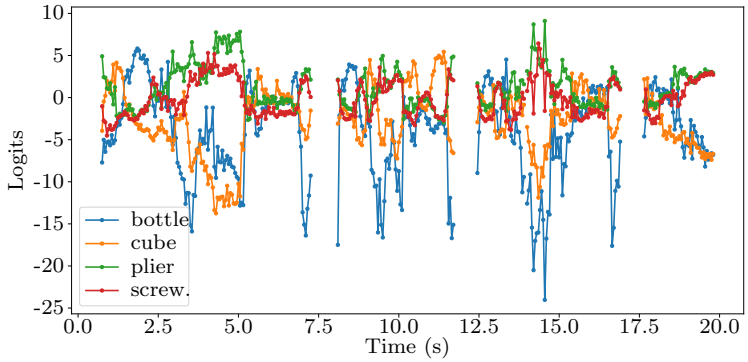
With the unreliability of the softmax probability, the values of the neural network before the softmax operation names logits were tested to ascertain if they would be able to provide additional information, resulting in the graphics in Figure 14.



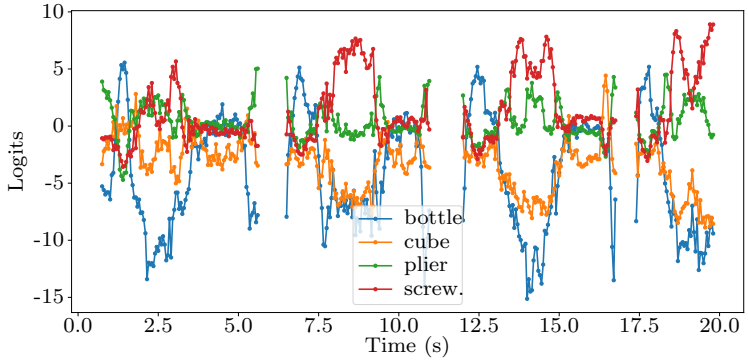
(a)



(b)



(c)



(d)

Figure 14

The results show that across all four classes, when a logit value is higher than six, the model is always giving a correct prediction. This information can be used to create a rule that will only consider the prediction valid if the highest logit value is higher than six. As an extra layer of security, the prediction will only be considered valid if the model gives the same prediction for two consecutive frames.

4 Path Planning using Reinforcement Learning

This section covers the development of a reinforcement learning model to plan the path of a robot manipulator in a collaborative cell. The model was trained and tested in a simulated environment and then a small integration with the real collaborative cell was done to test the model in a real environment. All the code developed for this research is available in the following repository: `https://github.com/lardemua/rl-robot-control`.

4.1 Simulated Environment

The environment used for model training was based on the Fetch Reach environment made available by Gymnasium Robotics[11]. In this environment, the task was to make the manipulator move the end-effector to a random 3D position above a table in front of the robot, as show in Figure 15. The robot in this simulation has 7-DoF and is controlled by small displacements of the gripper in cartesian coordinates, with Mujoco being responsible for the inverse kinematics computation. This environment also forced the end-effector to always be facing downwards which is not always the case in a real collaborative environment.

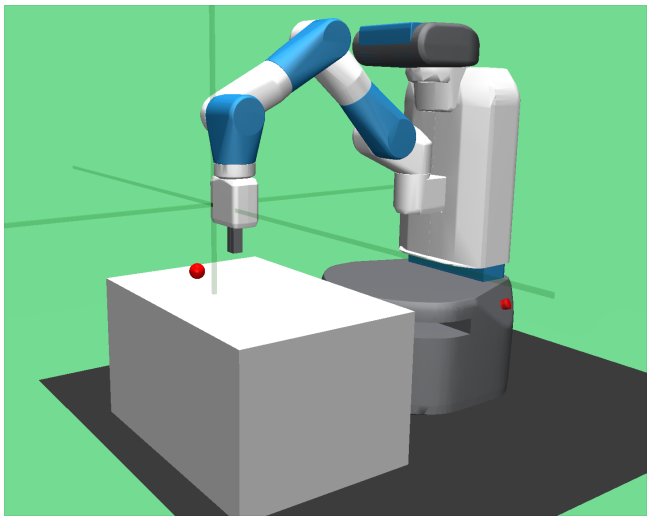


Figure 15: Fetch Reach Environment

The Fetch Reach environment served as a starting point for this research but

[11]Fetch Reach Environment from Gymnasium Robotics: `https://robotics.farama.org/envs/fetch/reach/`

ended up being mostly rewritten to not only resemble the available collaborative cell but also allow for direct joint control instead of the previous cartesian displacements. The final environment, now named Larcc, can be seen in Figure 16 which now includes a UR10e robot model with a Robotiq 2F-140 gripper just like in the real collaborative cell. The UR10e model was obtained from Menagerie, a collection of models for Mujoco curated by Google DeepMind[12] while the gripper model was obtained from the robosuite framework by ARISE Initiative[13]. The models used were edited so they could be used together, and the final environment was deemed acceptable given that the difference between the end-effector position in ROS and in the simulator differ by less than 1mm for the same joint positions. In this environment, the goal is to move the end-effector to a target position and orientation which are both represented by 3D axes in the simulator. As the actions affect the joints directly, a model trained is this environment ends up replacing the inverse or differential kinematics.

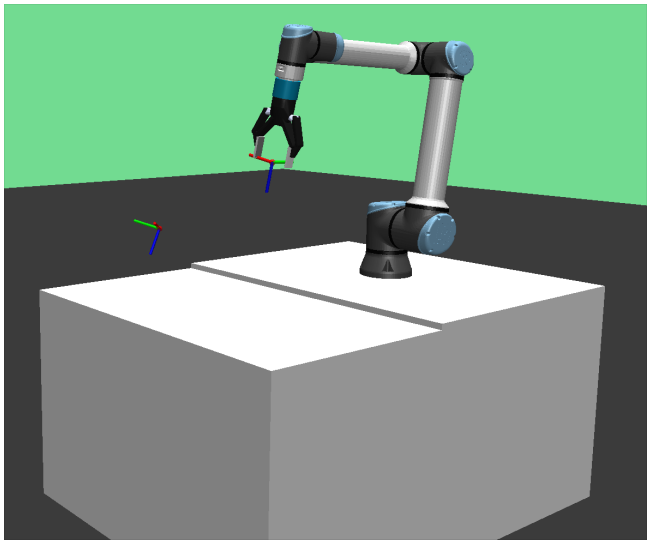


Figure 16: Larcc Environment

Starting State

To set up a new episode during training, the robot state and the goal position and orientation in the environment are reset.

For the goal, the position is sampled from a uniform distribution corresponding to the volume 10 cm to 60 cm above the table in front of the robot. The orientation is sampled from a uniform distribution in RPY format and then converted to quaternions. The orientation is then validated to check if it is facing mostly forward and downwards, which is the orientation that the gripper should have to pick up an object. If the orientation is not valid, a new one is sampled until a valid one is found.

[12]UR10e Model from Menagerie: `https://github.com/google-deepmind/mujoco_menagerie/tree/main/universal_robots_ur10e`

[13]Robotiq 2F-140 Model from robosuite: `https://github.com/ARISE-Initiative/robosuite/blob/master/robosuite/models/assets/grippers/robotiq_gripper_140.xml`

For the starting state of the robot there are two possibilities: it can start in the fixed position shown in Figure 16 or in a random position. The random position is obtained by sampling the joint positions from a uniform distribution in the $[-\pi, \pi]$ interval, which is the same interval used for the joint limits in the UR10e robot. However, given that a fully random starting position can make the task too difficult, the end-effector position and orientation from the random starting position is validated with the same constraints as the goal position and a new starting position is sampled until a valid one is found.

Observation Space

The observation space in this environment follows the structure commonly used Gymnasium environments being a dictionary with 3 keys:

- *observation*: consists of the positions of the 6 joints; given that they were limited to the $[-\pi, \pi]$ interval, they were normalized by a division by π;
- *achieved_goal*: current position and orientation of the end-effector; the position was normalized by subtracting the position of the robot base link in the environment and then dividing by the arm's range, and the orientation was kept because it is defined with quaternions which are already in the $[-1, 1]$ interval;
- *desired_goal*: position and orientation of the goal; normalized in the same way as the achived_goal above.

Action Space

The action space represents the displacements in all joints for a single timestep. These displacements are also normalized which means that the final action applied to the environment is the result of the action output given by a model which is in the $[-1, 1]$ range multiplied by the maximum displacements of the joints. The first two joints associated with the shoulder can move at most 0.8 radians per timestep while the other four joints associated with the elbow and the wrist can move at most 1.2 radians per timestep, representing the different maximum velocities on the UR10e robot joints (https://www.universal-robots.com/products/ur10-robot/).

Rewards

The used environment has dense rewards so as to give the agent frequent and consistent feedback about its actions. This means that in every timestep the reward will increase if the the end-effector is closer to the goal and decrease otherwise, helping the model learn which actions lead to a successful episode. The reward in each timestep is obtained from the following rewards:

- *position_reward*:

$$\begin{cases} 1 - distance(goal\_pos, end - effector\_pos) & \text{if } distance(...) < 2, \\ -1 & \text{otherwise} \end{cases};$$

- *orientation_reward*:

$$max\left(\begin{cases} innerproduct(goal\_quaternion, end\text{-}effector\_quaternion), \\ innerproduct(-goal\_quaternion, end\text{-}effector\_quaternion) \end{cases}\right);$$

- *bonus_reward*:

$$\begin{cases} 1 & \text{if } position\_reward > 0.98 \text{ and } orientation\_reward > 0.98, \\ 0 & \text{otherwise} \end{cases}.$$

These rewards affect the final timestep reward with different weights, but the sum of the weights is always equal to 1 keeping the final reward in the $[-1, 1]$ range.

4.2 Results

The tests in the described environment were executed using the Soft Actor-Critic (SAC) algorithm. SAC is an off-policy model-free reinforcement learning algorithm known for its ability to achieve high performance while maintaining stability and sample efficiency. It is composed of three key components: an actor network, a critic network, and entropy regularization. The actor network is responsible for learning the policy and outputting the actions to be taken by the agent. The critic network is responsible for evaluating the actions taken by the agent. The entropy regularization is used to encourage exploration of the environment by introducing randomness or uncertainty in the agent's actions.

Fixed Initial Position Results

A SAC model was trained on the developed environment with a fixed initial robot state and with a 0.5 weight on the position reward and a 0.25 weight on the orientation and on the bonus reward. The model was trained with early stopping configured to evaluate the model every 500 episodes and stop training if the evaluation average reward does not increase for 20 evaluations. Figure 17 and Figure 18 show the evolution of the actor and critic loss during training.

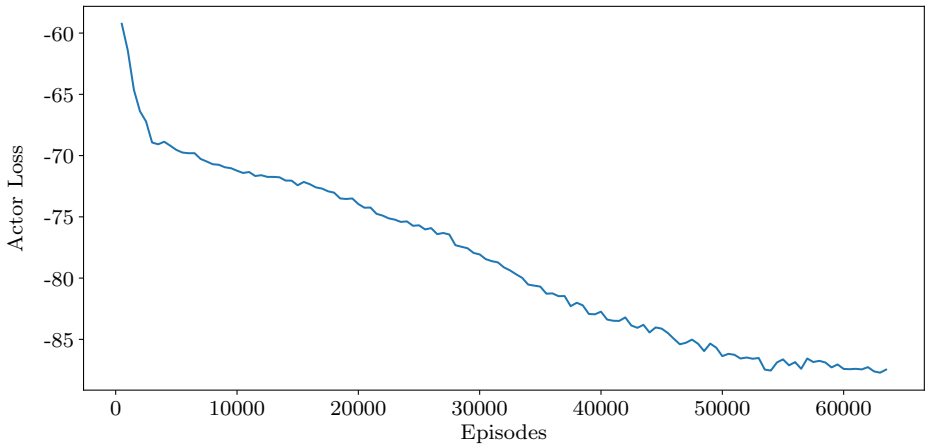


Figure 17: SAC Actor Loss during Training

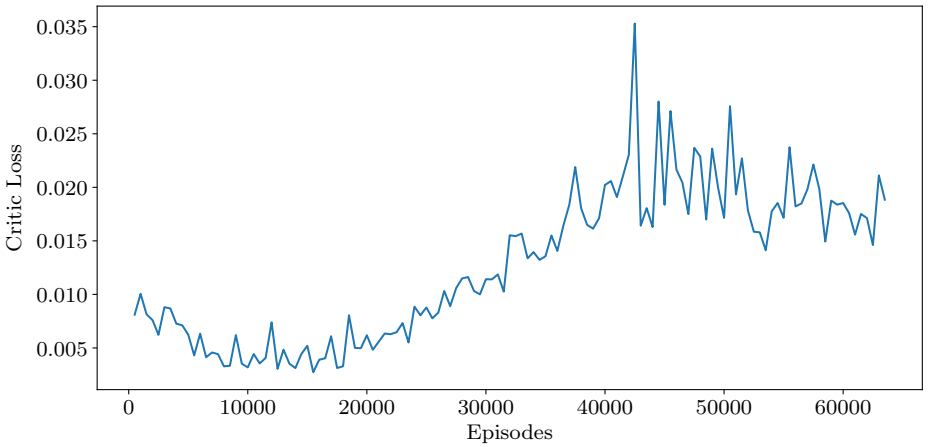


Figure 18: SAC Critic Loss during Training

Figure 19 shows the evolution of the entropy coefficient during training. A higher entropy coefficient indicates increased exploration of the environment by the model. Considering the reward components in Figure 20, the entropy coefficient initially descreases as the model learns to maximize the position and orientation reward but then increases again as the model tries to maximize the bonus reward.

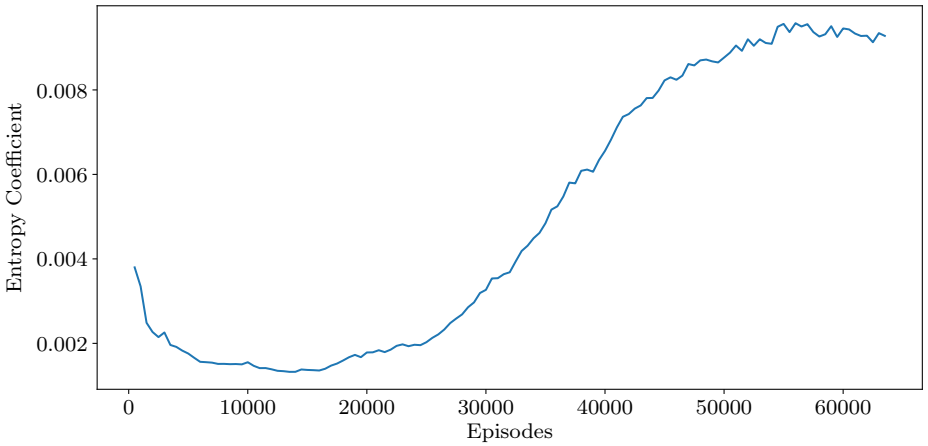


Figure 19: SAC Entropy Coefficient during Training

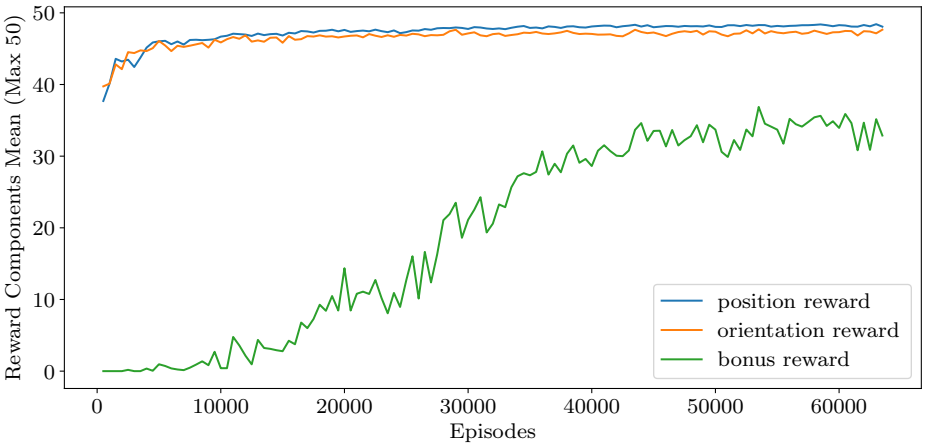


Figure 20: SAC Episode Reward Components Mean (Max 50) during Training

Figure 21 and Figure 22 show the evolution of the episode reward mean and the success rate during training. The success rate is calculated as the percentage of episodes where the episode is successful which also corresponds to the episodes where there is bonus reward. The success rate is a good indicator of how well the model is learning to reach the goal. Considering that the maximum reward in a single episode is 50, the model was able to reach a reward close to the maximum. Additionally, the fact that the validation reward is higher than the training reward is expected given that the in the trainig episodes the model attempts to explore the environment according to its entropy while in the validation episodes the model takes the best actions according to its learned policy.

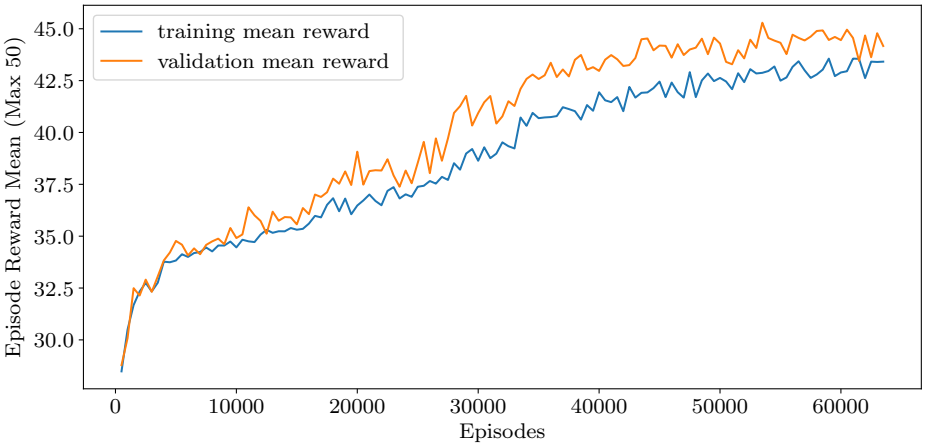


Figure 21: SAC Episode Reward Mean (Max 50) during Training

Figure 22: SAC Success Rate during Training

Reward Components Testing

As said before, the graphs above correspond to the model trained with a 0.5 weight on the position reward and a 0.25 weight on the orientation and on the bonus reward. Further testing was done with the model to check which would be the best weights to balance the position and orientation reward components. Table 3 shows the results for all tested weights, when the highest validation mean episode reward was recorded. The results show that the best weights are a 0.5 weight on the position reward and a 0.25 weight on the orientation reward, which is the same as the initial model. The other tested weights resulted in not only lower rewards but also on longer trainings.

Table 3: SAC Results with Different Reward Component Weights

Weights	Training Episodes	Training Episode Reward Mean	Validation Episode Reward Mean
position: 0.5 orientation: 0.25	53500	42.9	45.3
position: 0.44 orientation: 0.31	83500	32.0	33.2
position: 0.375 orientation: 0.375	98000	32.9	34.3
position: 0.31 orientation: 0.44	77500	31.7	32.7
position: 0.25 orientation: 0.5	81000	32.3	32.9

Random Initial Position Results

With the best weights defined, an attempt was made to train the model with a random but valid starting position. The results were significantly worse than

using a fixed starting position. Figure 23 and Figure 22 shows the evolution of the episode reward mean and the success rate. The model was not able to reach a reward close to the maximum and the success rate peaked at 0.5 around the 170000 training episodes and then decreased until 0 as the reward slowly decreased.

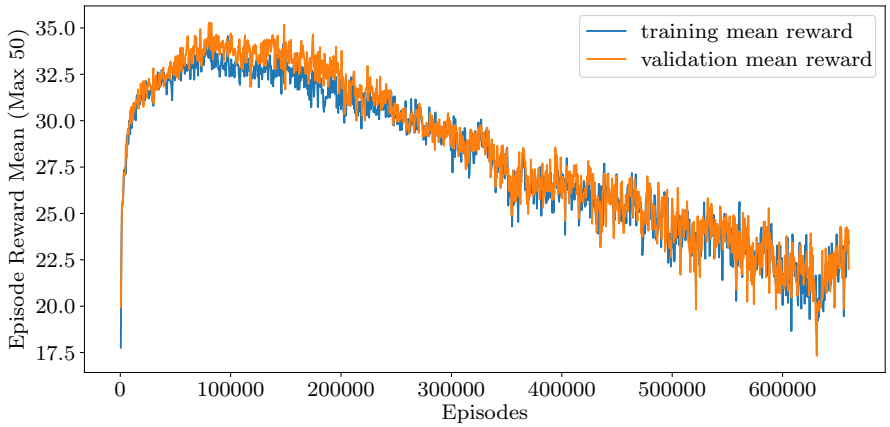


Figure 23: SAC Success Rate during Training (Random Starting Position)

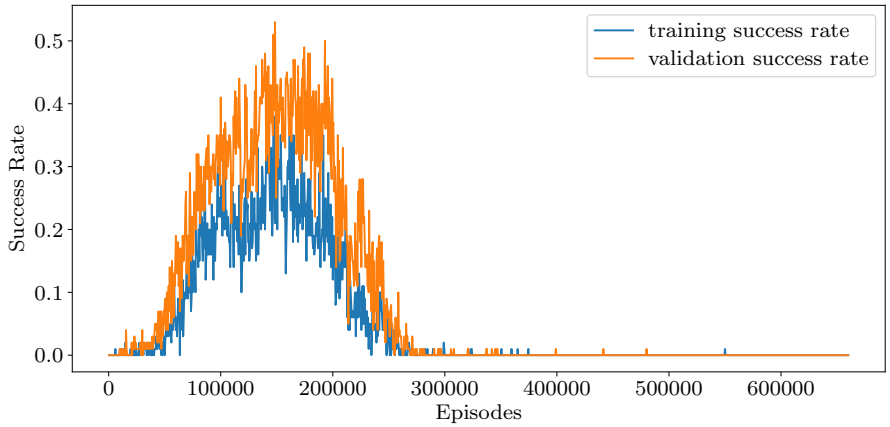


Figure 24: SAC Success Rate during Training (Random Starting Position)

4.3 ROS Integration

The best trained model was integrated with the ROS environment to do some initial tests in the real collaborative cell. Given the specific dependencies of the software used, this integration was done by using a docker container with ROS Noetic, stablebaselines3, and Gymnasium. As shown in Figure 25, a reinforcement learning model is loaded in a ROS node and whenever a new target position and orientation is received from the ROS environment, the model would

predict the actions to reach the goal in the simulated environment. The final joint positions in the simulator are then sent to the ROS environment where another node is responsible by the robot's movements.

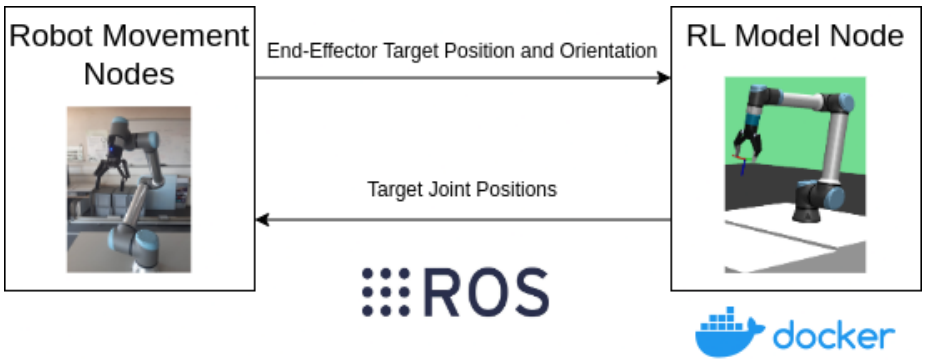


Figure 25: Reinforcement Learning Integration with ROS

This solution only replaces the computation of inverse or differential kynematics and still needs to be improved to not only completely replace MoveIt! but also to allow for real-time feedback, which would require a model capable of starting from different robot states.

5 Robot Controller

6 Conclusion

7 Plano Inicial de Atividades (To Remove)

Bolseiro: Pedro Miguel Loureiro Amaral

Orientadores: Filipe Miguel Teixeira Pereira da Silva; Vitor Manuel Ferreira dos Santos

Duração /início e fim do contrato: 6 meses - 16/01/2024 a 15/07/2024

Projeto: UIDB/00127/2020

Centro de custos: 3.62.80.11

Local de trabalho: Instituto de Engenharia Eletrónica e Informática de Aveiro (IEETA)

Título plano de trabalhos: Aprendizagem por reforço para antecipar a intenção humana em tarefas colaborativas

Contexto e objetivo da bolsa: A aprendizagem por reforço oferece uma metodologia para desenvolver políticas que maximizam a eficiência da colaboração humano-robô, levando em consideração as ações e intenções humanas. O plano de atividades visa a integração destas metodologias com ferramentas computacionais desenvolvidos no âmbito do projeto Augmanity (https://www.augmanity.pt). Neste contexto, o objetivo da bolsa está centrado no desenvolvimento de um sistema robótico capaz de antecipar as ações e/ou intenções de um operador humano durante a realização de tarefas colaborativas usando técnicas de aprendizagem por reforço, tendo em vista melhorar a eficiência e a segurança dos parceiros envolvidos.

Tarefas a desenvolver:

T1.Tarefa 1 - Levantamento do estado atual de conhecimento das tecnologias e metodologias associadas à capacidade de antecipação em robótica colaborativa.

T2.Tarefa 2 - Desenvolvimento de uma solução integrada baseada em técnicas de aprendizagem por reforço que permitam a um robô melhorar as suas capacidades para anticipar as ações e/ou intenções humanas.

T3.Tarefa 3 - Teste e avaliação de vários algoritmos de aprendizagem por reforço aplicados em diferentes cenários em ambientes reais.

T4.Tarefa 4 - Escrita de documentação relevante e preparação de um protótipo demonstrador.

Where should I describe the hand following?

8 References

[1] M. Quigley, K. Conley, B. Gerkey, *et al.*, "Ros: An open-source robot operating system," in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.

[2] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, eabm6074, 2022. DOI: 10.1126/scirobotics.abm6074. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[3] Orbbec, *Astra pro plus*, Last accessed 30 March 2023. [Online]. Available: https://shop.orbbec3d.com/Astra-Pro-Plus.

[4] U. Robots, *The ur10e*, Last accessed 9 May 2023. [Online]. Available: https://www.universal-robots.com/products/ur10-robot/.

[5] U. Robots, *Robotiq 2f-140*, Last accessed 24 October 2023. [Online]. Available: https://www.universal-robots.com/plus/products/robotiq/robotiq-2f-140/.

[6] WiredWorkers, *Ur10e*, Last accessed 9 May 2023. [Online]. Available: https://shop.wiredworkers.io/en_GB/shop/universal-robots-ur10e-87.

[7] C. Lugaresi, J. Tang, H. Nash, *et al.*, "Mediapipe: A framework for perceiving and processing reality," in *Third workshop on computer vision for AR/VR at IEEE computer vision and pattern recognition (CVPR)*, vol. 2019, 2019.

[8] F. Zhang, V. Bazarevsky, A. Vakunov, *et al.*, "Mediapipe hands: On-device real-time hand tracking," *arXiv preprint arXiv:2006.10214*, 2020.

[9] Google, *Mediapipe*, Last accessed 13 October 2023, 2023. [Online]. Available: https://developers.google.com/mediapipe.

[10] G. Amprimo, G. Masi, G. Pettiti, G. Olmo, L. Priano, and C. Ferraris, "Hand tracking for clinical applications: Validation of the google mediapipe hand (gmh) and the depth-enhanced gmh-d frameworks," *arXiv preprint arXiv:2308.01088*, 2023.