Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Robot Learning of Visual Servoing with Deep Reinforcement Learning

Supervisor:
Dr. Edward Johns

Author:
Apoorva Verma

Second Marker:
-

January 26, 2023

Contents

Chapter 1

Introduction

The field of robotics has been rapidly growing in recent years, with new methods to train robots constantly being developed. There are innumerable applications of robots within society. Some examples include manufacturing, where robots are used to perform a multitude of tasks from assembly to welding, healthcare, where robots can be taught to perform tasks such as surgery, and even space explorations, where robots have been sent to other planets to explore

Before the arrival of machine learning, robots were pre-programmed to follow certain rules while interacting with the environment. But the limitations of this method meant robots could not learn from the environment and adapt to changing conditions.

Robot learning is a field that focuses on the application of machine learning to robots. With machine learning, we have overcome the previous limitations of robotics and seen a significant improvement in the capabilities and applications of robots. Machine learning has enabled robots to perform tasks that were previously not possible such as grasping and recognising objects, speech recognition, and natural language processing.

There are many existing methods for robot learning (Section 2.1), e.g. reinforcement learning where a robot learns to perform a task by interacting with the environment and receiving rewards or penalties based on its actions. Another method of robot learning is imitation learning in which robots learn from demonstrations given by humans. Imitation learning can be quicker and easier to implement, particularly if it's difficult to mathematically model the rewards in reinforcement learning. However, imitation learning can still be inefficient as it can require a lot of samples of human demonstrations to be able to learn.

In 2017, Duan et al. at OpenAI, developed *one-shot imitation learning*, a new type of imitation learning in which only a single demonstration is required. The Course-to-Fine framework [1] is a new method for one-shot imitation learning developed in 2021. The method splits each task into two sub-tasks:

1. **Course** approach trajectory which represents the robot's motion as it moves from its initial (random) starting position to the *bottleneck* pose. The bottleneck pose is the pose at which the demonstration began.

2. **Fine** interaction trajectory which represents the robot's motion from the bottleneck pose as it carries out the task. This is done by simply replaying the end effector's velocities recorded from the original demonstration.

The method is discussed in more detail in Section 3.1.

Visual servoing is a field in robotics that uses visual data, e.g. from a camera, to control the motion of a robot. In this project, we will aim to implement the approach trajectory using visual servoing with deep reinforcement learning, rather than the current approach of supervised learning, and compare the results.

1.1 Objectives

The main objective of this project is to compare the performance of reinforcement learning against supervised learning for the *approach trajectory* in the Course-to-Fine imitation learning method (discussed in more detail in Section 3.1). Our goals for this project are summarised below:

- Explore current methods for visual servoing with deep reinforcement learning (DRL) and compare different DRL algorithms on a simple task, e.g. moving a vision sensor within a specific distance of an object.
- Implement the approach trajectory using multiple DRL algorithms and evaluate them.
- Compare the best implementation in simulation with the supervised learning approach in Course-to-Fine using the evaluation metrics discussed in Section 5.2.
- Deploy the best agent on a real robot using sim-to-real transfer and compare the final results with the supervised learning approach in Course-to-Fine using the evaluation metrics.

Chapter 2

Background

2.1 Robot Learning

Robot learning is the application of machine learning in the context of robotics. In robot learning, robots are trained to perform tasks by applying machine learning techniques. There are many types of robot learning methods:

1. Supervised and unsupervised learning (Section 2.2) use labelled and unlabelled datasets respectively to train machine learning models.

2. Reinforcement learning (Section 2.4) involves training a robot through trial and error and using a reward system so the robot can learn which actions to take.

3. Imitation learning (Section 2.6) is a technique where robots are given demonstrations on how to perform a task, which they learn from.

2.2 Machine Learning

Machine learning (ML) is a sub-field of Artificial Intelligence (AI) that aims to imitate the way humans learn. It focuses on the use of data and algorithms to enable systems to learn from experience autonomously, without explicitly being programmed to. ML algorithms have been around for a long time but technological advances in storage and processing power over the last couple of decades have exponentially improved the algorithms' abilities. The speed and scale at which the algorithms can be applied to big data is growing rapidly.

ML methods can be classified into one of the three sub-categories:

- **Supervised Learning**
  Supervised learning algorithms learn from labelled datasets that specify the correct output for each input. The algorithms approximate a function that best maps inputs from the training dataset to their respective outputs. This results in models that can classify inputs and accurately predict outputs on unseen data [2].

- **Unsupervised Learning**
  Unsupervised learning algorithms identify hidden patterns and structure in data from an unlabelled dataset [2]. As there is no specified correct output, the algorithms aim to organize and analyse the data, often resulting in clustering.

- **Reinforcement Learning**
  Reinforcement learning algorithms aim to train agents to learn an optimal strategy through trial and error by interacting with the environment. The algorithms use a reward system to evaluate different strategies and aim to maximise the reward. Reinforcement Learning is discussed in more detail in Section 2.4.

2.3 Neural Networks

Neural networks, also known as Artificial Neural Networks (ANNs), are mathematical models central to ML, particularly Deep Learning.

There are many types of ANNs, e.g. Multi-Layer Perceptron (MLP), Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). We will focus on MLP and CNN as these are the most popular and the ones used in the Deep Reinforcement Learning algorithms we discuss in Section 2.5.

2.3.1 Multi-Layer Perceptron (MLP)

2.3.2 Convolutional Neural Network (CNN)

2.4 Reinforcement Learning

Reinforcement learning (RL), a sub-field of ML, consists of a range of algorithms and techniques used to train agents how to optimally behave in different environments. With trial-and-error at the core of the method, agents learn by repeatedly interacting with the environment and receiving rewards as feedback based on their actions.

2.4.1 Markov Decision Processes (MDPs)

Markov Decision Processes (MDPs) are a mathematical framework used to model RL problems. MDPs can be represented as a tuple $<\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \lambda>$ where:

- State space ($\mathcal{S}$): The set of all possible states of the environment. At each time step, t, the agent receives the current state of the environment, s_t.
- Action space ($\mathcal{A}$): The set of all possible actions that the agent can take. At each time step, t, the agent chooses an action, a_t, to take based on the current state of the environment.
- State transition probability function ($\mathcal{P}$): A function that defines $\mathcal{P}^{a_t}_{s,s_{t+1}}$, the probability of transitioning from s_t to s_{t+1} given action a_t is taken, for all states and actions.
- Reward function ($\mathcal{R}$): A function that defines $\mathcal{R}^{a_t}_{s,s_{t+1}}$, the immediate reward received for transitioning from s_t to s_{t+1} given action a_t is taken, for all states and actions.
- Discount factor (λ): A scalar value between 0 and 1 that determines how important future rewards are.

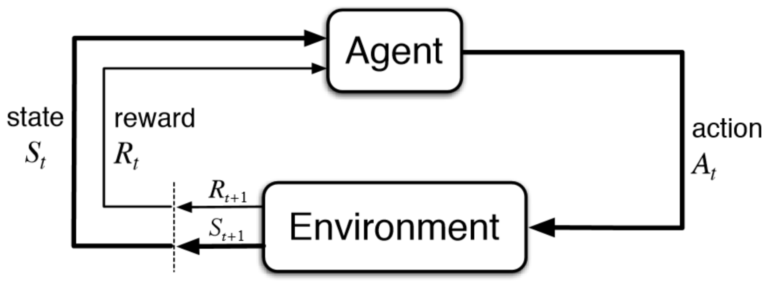

Figure 2.1: The agent-environment interaction in reinforcement learning [3]

An *episode* τ is a sequence of states, actions and observations that an agent experiences as it interacts with the environment. An episode continues until the agent reaches a terminal state or infinitely, unless you set a limit on the number of time steps. Note that in Figure 2.1, $\mathcal{R}_{t+1}$ and $\mathcal{S}_{t+1}$ are there to demonstrate that this process is repeated for all time steps.

$$\tau = (s_1, a_1, r_2, ..., s_t) \tag{2.1}$$

The *return* R_t is the sum of the immediate reward and discounted future rewards:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.2}$$

A *policy* π is a function that defines the behaviour of an RL agent. It is represented as a mapping from states to actions the agent will take. A policy can be:

- **Deterministic**: A deterministic policy will always choose the same action for a given state.

$$a = \pi(s) \tag{2.3}$$

- **Stochastic**: A stochastic policy assigns a probability distribution over all actions for each state, allowing the agent to explore different actions.

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s] \tag{2.4}$$

MDPs are used to model RL problems as optimisation problem where the goal is to learn the *optimal* policy π^*, the policy that maximises the expected return. In order to determine the effectiveness of a policy, we need to be able to evaluate it so we can compare policies and choose the optimal one.

To evaluate a policy we can use the state-value and action-value functions:

- **State-value function**: The state-value function calculates the expected return of a policy π when starting in state s and following policy π from then on.

$$V^\pi(s) = \mathbb{E}[R_t | S_t = s] \tag{2.5}$$

- **Action-value function**: The action-value function calculates the expected return of a policy π when starting in state s, taking action a and following policy π from then on.

$$Q^\pi(s, a) = \mathbb{E}[R_t | S_t = s, A_t = a] \tag{2.6}$$

The state-value function can be calculated by using the *Bellman Equation*, a recursive equation which defines the value of a state s using the value of all states s' reachable from it:

$$V^\pi = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} (\mathcal{R}^a_{ss'} + \gamma V^\pi(s')) \tag{2.7}$$

The relationship between the state-value and action-value function is:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q^\pi(s, a) \tag{2.8}$$

The maximum possible state-value or action-value for each state is given by the *Bellman Optimality Equation* (BOE):

$$V^*(s) = \max_a \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} (\mathcal{R}^a_{ss'} + \gamma V^*(s')) \tag{2.9}$$

To follow the optimal policy, the agent should take the action that maximises the BOE in any state. Therefore, solving an RL problem involves solving the BOE. The BOE can be solved directly in some situations, e.g. if we have a full model of the environment and it is deterministic and stationary. In these situations, the BOE is a linear equation so it can be solved using simple linear algebra methods, e.g. matrix inversion. However, if the environment is stochastic and non-stationary then the BOE is non-linear as transition probabilities and rewards can change over time, so it cannot be solved directly and other methods must be used, e.g. deep reinforcement learning for stochastic environments.

2.4.2 Types of RL algorithms

RL algorithms can be classified as either:

1. **Model-based**: Model-based algorithms explicitly represent the model, and therefore the dynamics, of the environment, which includes $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$ and $\mathcal{R}$. The model could either be given or developed by the agent by exploring the environment. Once the agent has a model, it can use this information to compute the optimal policy directly.
   An example of model-based algorithms are Dynamic Programming algorithms such as value iteration and policy iteration which iteratively update the value function or policy until they converge on the optimal.

2. **Model-free**: Model-free algorithms do not explicitly represent a model of the environment and instead rely on sampling the environment. The agents interact with the environment, directly learning the value function or policy based on the rewards it receives.
   An example of a model-free algorithm is the Monte Carlo algorithm. The Monte Carlo algorithm starts by generating an episode by following the policy. It then updates its estimate of the value function for all states-action pairs experienced in that episode based on the return, improving its policy. Finally, it repeats this process using the improved policy until the value function converges on the optimal value function.

RL algorithms can also be further classified as:

1. **On-policy**: On-policy algorithms learn the value of the policy it is following, i.e. the algorithms learn policy π and use the same policy to sample from the environment.

2. **Off-policy**: Off-policy algorithms learn the value of a different policy than the one that it is following, i.e. the algorithm learns the target policy π but acts based on the behaviour policy π'.

2.4.3 Tabular RL

RL agents need to know the value of a state-action pair so they can choose the best action to take and derive the optimal policy. Early RL algorithms, e.g. Dynamic Programming and Monte Carlo, keep track of the values of all state-action pairs. These algorithms are *tabular* as they maintain the state-action values in a table data structure. Tabular algorithms are a sufficient solution for problems when the state and action spaces are small and discrete. However, tabular RL has some limitations:

- Curse of dimensionality: When the state and action spaces are large and/or continuous it can become infeasible to store all the state-action pairs in a table due to memory limitations and to efficiently search through the table to find the best action. As the number of states and actions increase, the memory requirements increase exponentially.

- Lack of generalisation: Tabular solutions require each state-action pair to be visited to learn about them, so they are very specific to the environment they are trained in and don't generalise to unseen states. Therefore, training an agent on an environment with a large state and action space can be very time-consuming. Generalisation can help reduce the training time as similar state-action pairs frequently have similar expected returns.

2.5 Deep Reinforcement Learning

Deep Neural Networks (DNNs) are ANNs (Section 2.3) with a complex architecture consisting of multiple layers and a vast network of neurons. *Deep learning* is a sub-field of ML focused on training DNNs. DNNs are capable of learning features from raw input data and solving complex problems. Although they require a large amount of data to be trained, once trained, DNNs have the ability to learn and generalise from new data in a way that isn't possible with traditional ML methods.

In 2009, Fei-Fei Li et al. released ImageNet, one of the largest image databases available at the time, with over 14 million labelled images and more than 20,000 different categories [4][5]. ImageNet helped accelerate deep learning research, as DNNs require large amounts of data to be trained on. This led to AlexNet, the first deep learning model to win the ImageNet Large Scale Visual Recognition Challenge[1] (ILSVRC), being released in 2012 [6]. AlexNet is a deep CNN model trained on the ImageNet dataset and it's success sparked a renewed interest in the field.

Deep Reinforcement Learning (DRL) combines the powers of deep learning and reinforcement learning to overcome the limitations of tabular RL. In DRL, instead of using a table to store the value of each state-action pair, DNNs are used to approximate the value functions. Therefore, DRL algorithms can handle continuous, infinite state and action spaces and have the ability to generalise so they won't require the agent to visit all the state-action pairs.

With the growing focus in deep learning after the release of ImageNet and advances in computational power, DRL started to gain popularity as researches started using DNNs to approximate the value functions in RL. In 2013 we saw the biggest breakthrough in the field when Google DeepMind developed DQN, "the first deep learning model to successfully learn control policies" directly from images using RL [7]. The DQN model consisted of a CNN (Section 2.3.2) with the image pixels as raw input data and the value function as output and was trained using a variant of Q-learning [8]. The model successfully learned to play 7 Atari games, outperforming all previous approaches on 6 games and humans experts on 3 games.

As this project's aim is to train a robot to reach the bottleneck pose using RL, the state space will be continuous, so we will be using DRL algorithms to tackle this problem.

2.5.1 Algorithms

There is a vast range of DRL algorithms available, as shown in Figure 2.2, and they usually fall within 2 categories: model-free and model-based. We will be focusing on model-free algorithms as we don't have a model of the environment and learning a model directly from the environment can be very difficult, especially due to bias in the model resulting in sub-optimal performance in the real environment [9]. Model-free algorithms can be further categorised into:

1. **Value-based**: Value-based methods aim to learn the optimal value function $Q^*(s, a)$ and are typically off-policy. These methods are usually less stable, however they have a higher sample efficiency as they effectively reuse data.
2. **Policy-based**: Policy-based methods directly learn the optimal policy, a mapping from states to actions, which is usually represented using a neural network and are often on-policy. As these methods directly optimise the policy, this can result in faster convergence and more stable training [10].

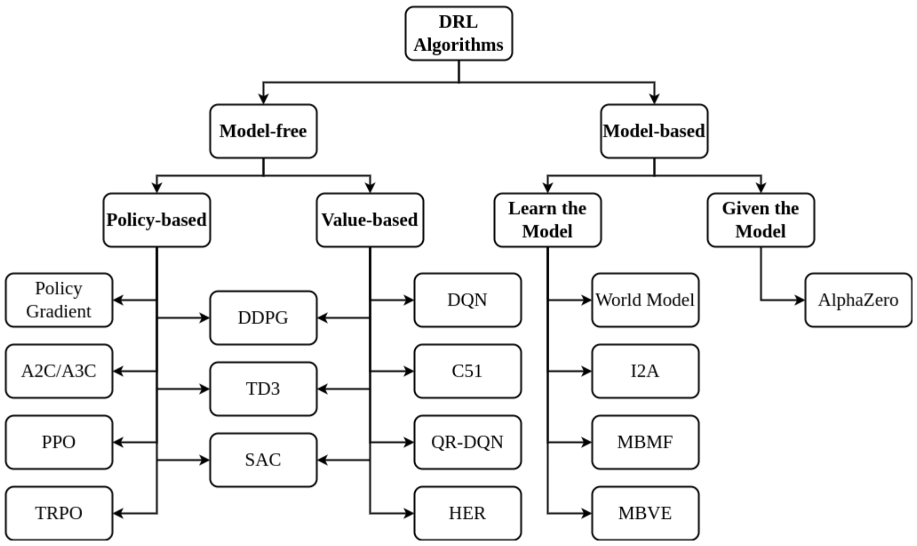

Figure 2.2: Taxonomy of Deep Reinforcement Learning Algorithms [9][10]

[1]https://www.image-net.org/challenges/LSVRC/

2.5.2 PPO

2.6 Imitation Learning

Getting a robot to learn a task from scratch is usually too time consuming so, inspired by how children learn, robot learning from demonstration has become increasingly popular. This resulted in imitation learning, a field within robot learning in which agents learn to perform a task from demonstrations. The goal of the agent is to learn a policy that represents the expert's behaviour from the demonstrations.

2.6.1 Motivation

Deep reinforcement learning has had huge success within the last decade in solving games [7][11]. Until recently, deep reinforcement learning was also the main method used in robotics for a robot to learn a task from scratch. However, using deep reinforcement learning to teach a robot can present some challenges:

1. Time consuming: It can require a lot of episodes and sometimes also human intervention to reset the environment after each episode.

2. Sparse rewards: In some cases, e.g. learning to play a game like Go [11], there is an obvious reward function, the score, which can be directly optimised. However, in robotics there is usually a binary objective, e.g. grasping an object, which leads to sparse rewards. Although sparse reward functions are simple to define, it can require a lot of exploration for a robot in an environment with sparse rewards to receive a reward and therefore learning can be slow [12].

A potential solution for environments with sparse rewards is to manually design a reward function, however this becomes increasingly difficult as we move from simple environments and problems to more complex ones [13], e.g. tying a surgical knot [14]. So, to overcome both these problems we can use learning by demonstration, also known as imitation learning, as it's often a lot easier to provide an expert demonstration. Also, imitation learning has been shown to significantly reduce the need for global exploration and speed up tasks, e.g. the Ball-in-a-Cup task in [15] where they combine imitation learning with reinforcement learning.

2.6.2 Methods

Currently, there are 2 main methods for imitation learning:

1. **Behaviour Cloning**
   In behaviour cloning methods, supervised learning methods are typically used to learn a policy that directly maps the inputs to the demonstrated action.

2. **Inverse Reinforcement Learning**
   As discussed in Section 2.4 on reinforcement learning, given a reward function an optimal policy that maximises the reward can be deduced. However, when using imitation learning the reward function is often unknown. Using expert demonstrations, we can learn the reward function of the environment and then find the optimal policy using reinforcement learning. This method of learning the reward function from demonstrations is called Inverse Reinforcement Learning.

2.6.3 One-Shot Imitation Learning

Imitation learning should aim to minimise the amount of interaction required by the human and the amount of prior knowledge of the task required by the agent [1]. Ideally, robots should require very few demonstrations to be able to generalise what they've learnt and apply it to new contexts for the same task. One-Shot imitation learning is a field which attempts to teach robots from a single demonstration.

This project will be based on a new method of one-shot imitation learning called Course-to-Fine Imitation Learning which is discussed in Section 3.1.

2.7 Visual Servoing

Visual servoing, also known as visual servo control, is a field within robotics which refers to the use of visual data, e.g. from a camera, to control the motion of a robot [16]. It has enabled robots to perform tasks such as grasping [17] and robotic surgery [18] and is hugely beneficial when robots are interacting with non-stationary environments as the robots can act based on the current state of the environment. The aim of visual servoing is to minimise the error $e(t)$ defined as:

$$e(t) = f(m(t), a) - f^* \tag{2.10}$$

where $m(s)$ is a set of image measurements, e.g. coordinates of regions of interest, and a is a set of parameters, e.g. camera intrinsic parameters. The function f is different based on the type of visual servoing but $f(m(t), a)$ represents the actual values, and f^* represents the desired values. There are two main types of visual servoing:

1. Position/Pose Based Visual Servoing (PBVS)
   PBVS uses visual data to compute the 3D pose (position and orientation) of the robot and defines the error as the difference between the current pose and the desired pose. In PBVS, f is defined as a set of 3D parameters that are calculated from the image measurements m.

2. Image Based Visual Servoing (IBVS)
   IBVS extracts features from the image and directly calculates how to move the robot to converge the features on the current image to the desired coordinates by reducing the error, which is defined as the difference between the positions of the features in the 2D images. In IBVS, f is defined as a set of visual features directly available from the image.

As the Course-to-Fine Imitation Learning framework, which this project is based on, uses IBVS, we will focus on that.

Chapter 3

Related Work

3.1 Course-to-Fine Imitation Learning

Course-to-Fine Imitation Learning is a new method of imitation learning introduced by Edward Johns in [1]. The Course-to-Fine Imitation Learning method allows robot manipulation tasks, e.g. closing the lid of a bottle, to be learnt by a robot from a single demonstration by a human. The framework achieves this by modelling the robot's motion with two trajectories:

1. **Course** approach trajectory
   The course approach trajectory represents the robot's motion as it moves from its initial (random) starting position to the *bottleneck* pose. The bottleneck pose is the pose at which the demonstration began.

2. **Fine** interaction trajectory
   The fine interaction trajectory represents the robot's motion from the bottleneck pose as it carries out the task. This is done by simply replaying the end effector's velocities recorded from the original demonstration.

Most other imitation learning methods today focus on teaching the robot a task end-to-end. However, this isn't the most practical approach and the way the Course-to-Fine framework splits the task into two trajectories provides it with several advantages when compared to other existing imitation learning methods:

- Data efficient: The most obvious advantage is that the method is *one-shot* meaning it only requires a single demonstration. This is very data efficient and saves a lot of time during training.

- No environment resetting: As the task is split into two trajectories, and the robot only needs to be trained for the coarse approach trajectory, this means the object won't be interacted with during training. Therefore, the robot can independently learn with no input from a human. In many end-to-end methods, the environment needs to be reset after each episode of training as the robot can move the object.

3.1.1 Method

3.2 DOME

3.2.1 Method

3.3 Visual Servoing with Deep Reinforcement Learning

Chapter 4

Early Work

While researching and testing methods for visual servoing, we need to be able to test different approaches on a robot. Although we could use real a robot and run different algorithms, storing and evaluating the results, this would be very time-consuming.
An alternative to using a real robot is to test our research in simulation, allowing us to explore many different approaches a lot quicker. If there is enough time, we hope to test the final approach on a real Sawyer[1] robot arm.

4.1 CoppeliaSim/V-Rep

To simulate the environment for this project we will be using CoppeliaSim (previously called V-Rep) [19]. CoppeliaSim is a simulation software that allows users to easily create *scenes*, virtual environments, and run simulations on them. To setup a scene, the user can just drag and drop primitive shapes, which can be used to create new objects, and many pre-installed models of real robots, including the Sawyer robot arm.

4.1.1 PyRep

CoppeliaSim provides an extensive Lua API that can be used to create custom scripts to interact with the environment and control the objects and robots during the simulation.

Python has a very large and active community with many tools, libraries, and frameworks available for ML development, therefore we chose to use it for this project. PyRep [20] is a toolkit built for CoppeliaSim with a simple and fast Python API which provides an object-oriented approach for interacting with the simulated environment.

4.2 OpenAI

OpenAI Gym is a toolkit for developing and testing RL agents [21]. It provides a variety of environments, e.g. Atari games, classic control tasks such as CartPole, and robotic simulations. Its aim is to provide a common interface for researchers to test and compare their RL implementations.

OpenAI Gym has a consistent interface for interacting with environments and the API allows you to easily create your own custom environments by implementing the interface "Env" [22], which is what we will be doing for this project. The interface has the following attributes and methods:

- **Action space**: The action space defines the format of valid actions. E.g. a discrete action space could be moving a robot $\pm$1cm in any direction and a continuous action space could be a continuous range of distances over all directions.
- **Observation space**: The observation space defines the format of valid observations/states. E.g. an RGB image could be used to represent the state in which case the observation could be a 3D array with the shape as (image width, image height, 3).

[1]https://www.rethinkrobotics.com/sawyer

- **Reward range**: The reward range is a tuple containing the minimum and maximum possible rewards.
- **reset()**: The reset method is called at the start of each episode. It should reset the environment and return the initial observation.
- **render()**: The render method renders the environment, displaying the current state of the environment to the user.
- **step(action)**: The step method takes an action as an argument and runs one timestep, executing the action in the environment. This will result in the environment being in a new state and a reward is received by the agent. The method returns a tuple: (observation, reward, done, infos) where done is True if the episode has terminated, indicating reset() should be called, and infos is a dictionary that can contain additional information about the environment if necessary.
- **close()**: The close method should close the environment, releasing any resources that are no longer required.

4.3 Stable-Baselines3

Stable-Baselines3 [23] is an open-source RL library that provides reliable implementations of many state-of-the-art DRL algorithms, e.g. PPO [24], DDPG [25] and DQN [26]. We will be using this to experiment with different RL algorithms and find the one that performs best on the visual servoing task we aim to implement in this project.

4.4 Progress

For these experiments, we setup a basic PPO pipeline using Stable-Baselines3's implementation of PPO. The environment in CoppeliaSim consisted of a vision sensor and an (a-symmetrical) object, with a distance of 2m between them as shown in Figure 4.1. The object the vision sensor is trying to reach is called the *target*. The starting positions of both the vision sensor and the target remained the same for each episode.

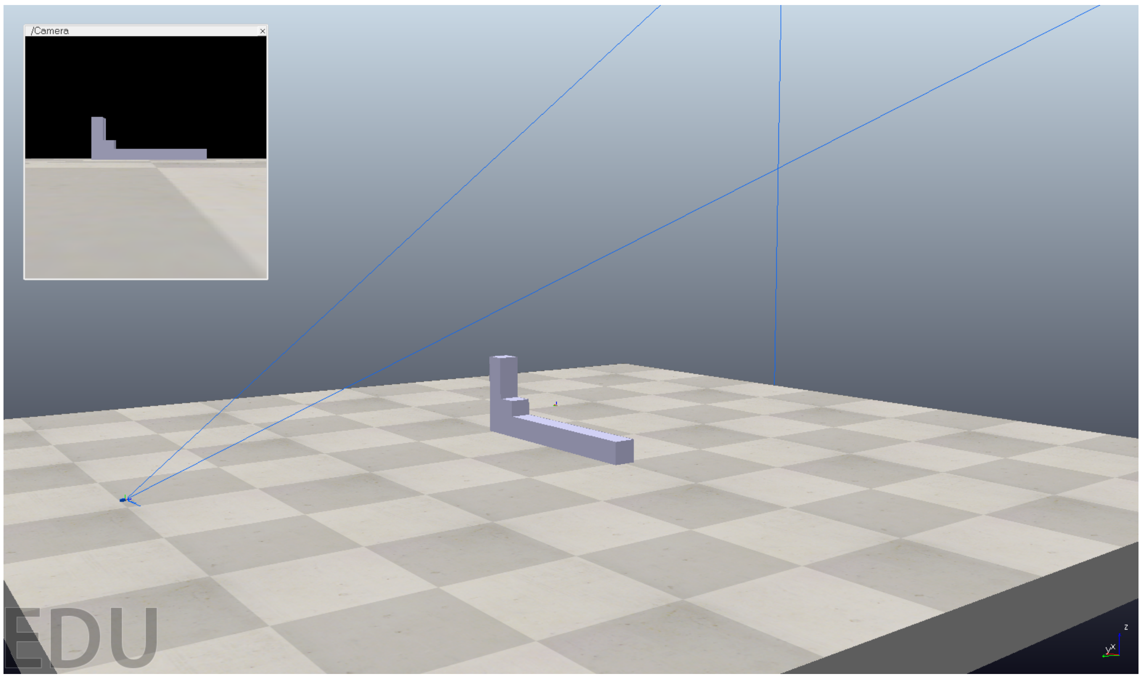

Figure 4.1: Environment set-up in CoppeliaSim

So far, we have carried out the following experiments:

4.4.1 Experiment 1

Task

The goal of this experiment was to move the vision sensor to within 1m of the target by using the exact positions of the vision sensor and target as the state.

Method

1. Setup the environment using OpenAI Gym as explained in Section 4.2.
   - Observation space: an array of size 6 containing the two 3D coordinates (x, y, z): the vision sensor's position and target's position.
   - Action space: a discrete action space with 6 actions, ± 0.1m in any direction (x, y or z).
   - Step function: performs the chosen action by updating the position of the vision sensor and calculates the reward for the step.
   - Reward: negative distance between the vision sensor and target. If the vision sensor reaches the goal of being within 1m of the target, the reward is 500 and the episode is complete.
   - Reset function: resets the vision sensor back to the same initial position.
2. Run the Stable-Baselines3 PPO algorithm on the custom environment.

Results

Below you can see the results of training the agent over 100,000 episodes. It takes the agent $\approx$50000 episodes to learn the optimal policy. The agent successfully reaches the maximum reward of $\approx$250 in 100 steps ($\frac{1}{0.01}$m = 100 steps).

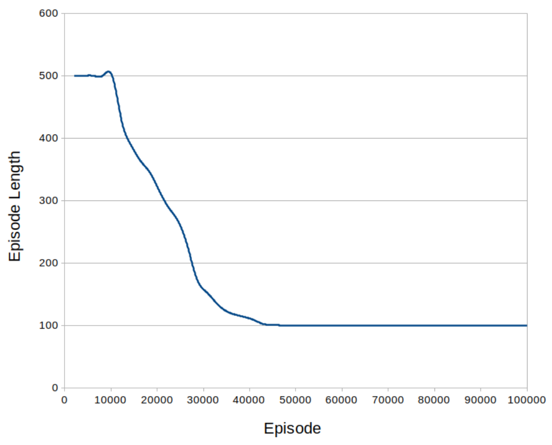

Figure 4.2: Episode length over 100,000 episodes for Experiment 2

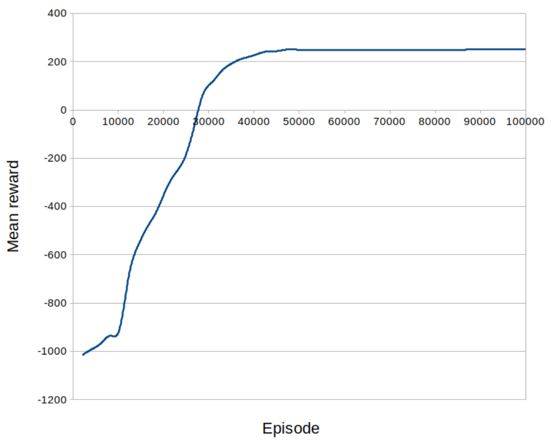

Figure 4.3: Mean reward over 100,000 episodes for Experiment 2

4.4.2 Experiment 2

Task

The goal of this experiment was also to move the vision sensor to within 1m of the target however, instead of using the exact positions of the vision sensor and target as the state, we used the image captured by the vision sensor.

Method

The method is the same as Experiment 1 (Section 4.4.1) except for the observation space:

- Observation space: a 3D array of shape (255, 255, 3) containing the normalised RBG image captured from the vision sensor.

Results

Below you can see the results of training the agent over 100,000 episodes. It takes the agent longer to learn the optimal policy than Experiment 1, ≈75000 episodes. The agent successfully reaches the maximum reward of ≈250 in 100 steps ($\frac{1}{0.01}$m = 100 steps).

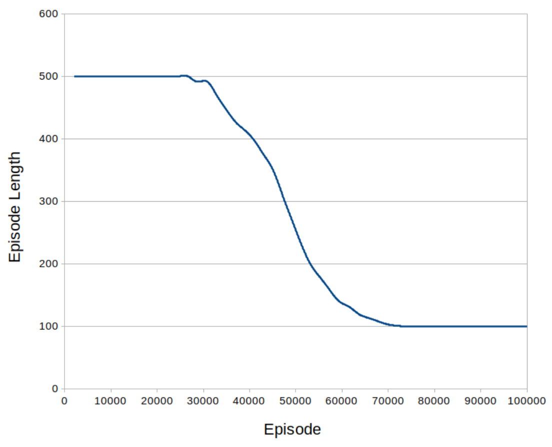

Figure 4.4: Episode length over 100,000 episodes for Experiment 3

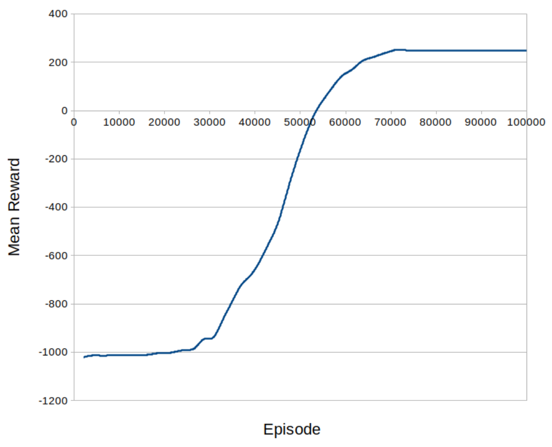

Figure 4.5: Mean reward over 100,000 episodes for Experiment 3

Chapter 5

Project Plan

5.1 Timeline

So far, I have written the Background (Chapter 2) of the report, set up the environment and completed some basic tasks (Chapter 4).

Below is my plan, in 2 week sprints, until the final project deadline and presentation at the end of June:

30th Jan - 12th Feb	Complete background section and related work research. Perform experiments: 1. Same as Experiment 2 (Section 4.4.2) but using non-linear paths. So, the action space will be 3 continuous values, the step size in the x, y and z directions. 2. Continuing from the experiment above, but starting the vision sensor from any random location.
13th Feb - 26th Feb	Test multiple different DRL algorithms on the final experiment using images, with non-linear paths and random starting locations for the vision sensor.
27th Feb - 12th Mar	Start exploring poses. Perform experiments: 1. Learn to reach a specific pose given the target pose (not moving directly to the target pose, but using the target pose to calculate the reward). 2. Main project aim - Learn to reach a specific pose given the image from the target pose.
13th Mar - 26th Mar	Exams and revision
27th Mar - 9th Apr	Finish final experiment. Evaluate the final algorithm using the evaluation plan below and testing on the complete tasks, both the approach and interaction trajectories.
10th Apr - 23rd Apr	Complete final experiment evaluation and write up results. Start exploring sim-to-real transfer. Start final report.
24th Apr - 7th May	Sim-to-real transfer
8th May - 21st May	Sim-to-real transfer. Final report.
22nd May - 4th Jun	Sim-to-real transfer. Final report.
5th Jun - 18th Jun	Final report and presentation
19th Jun - 30th Jun	Finish final report and presentation.

5.2 Evaluation Plan

As you can see in the timeline (Section 5.1), we will train multiple agents to implement the approach trajectory using different DRL algorithms. To compare the performance of these agents, we will use several everyday tasks as shown in Figure 5.1 as benchmarks. These are the same tasks used to test the Course-to-Fine framework, so the results can be compared with the original results from the supervised learning approach as well.

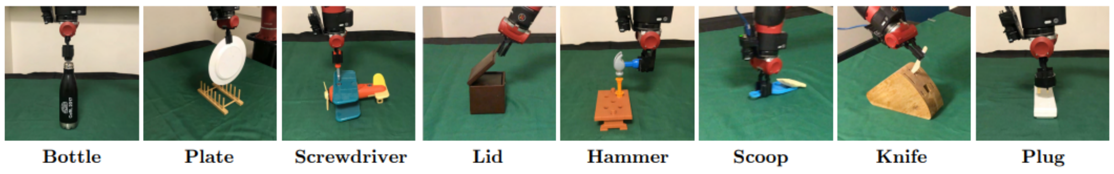

Figure 5.1: 8 everyday tasks that we will use for imitation learning experiments [1]

We will use the following evaluation metrics to evaluate the performance of different approaches:

1. **Training time** - time taken to train the agent
2. **Task time** - time taken for the trained agent to complete the task
3. **Accuracy** - how close the final pose is to the bottleneck pose.
   To compare the accuracy, we will measure the mean error in final position and orientation of the robot when compared to the bottleneck pose over multiple runs.
4. **Reliability** - the success rate of the agent completing the task.
   To compare the reliability, we will run the agent multiple times and calculate the success rate (% of tasks it successfully completed).

Chapter 6

Ethical Issues

There are a few ethical issues to consider for this project as the robot will be learning from human demonstrations which poses a risk as this method could be leveraged by people with harmful intentions.

Firstly, the project raises the issue of safety as the robot could learn to perform actions that are dangerous and harmful. This would be particularly problematic in cases where the robot's actions can have serious consequences e.g. in healthcare or military applications.

Secondly, an issue that comes up is accountability. As the robot is being trained by human demonstrations to complete a task, the question of who's accountable for the robot's actions comes up. Especially if the robot doesn't learn correctly and performs potentially harmful actions that weren't demonstrated by the human.

Finally, if the human training the robot has any biases or discriminates in any way, then the robot will learn this and act in this way, which could be harmful.

Bibliography

[1] Johns E. Coarse-to-fine imitation learning: Robot manipulation from a single demonstration. In: 2021 IEEE International Conference on Robotics and Automation (ICRA). IEEE; 2021. p. 4613-9.

[2] IBM. What is machine learning?; 2023. [Accessed: 17-01-2023]. Available from: https://www.ibm.com/topics/machine-learning.

[3] Sutton RS, Barto AG. Reinforcement learning: An introduction. MIT press; 2018.

[4] Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L. Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. Ieee; 2009. p. 248-55.

[5] ;. Available from: https://www.image-net.org/.

[6] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. Communications of the ACM. 2017;60(6):84-90.

[7] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, et al. Playing atari with deep reinforcement learning. arXiv preprint arXiv:13125602. 2013.

[8] Watkins CJ, Dayan P. Q-learning. Machine learning. 1992;8(3):279-92.

[9] Part 2: Kinds of RL algorithms¶;. Available from: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

[10] Nguyen H, La H. Review of deep reinforcement learning for robot manipulation. In: 2019 Third IEEE International Conference on Robotic Computing (IRC). IEEE; 2019. p. 590-5.

[11] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, et al. Mastering the game of Go with deep neural networks and tree search. nature. 2016;529(7587):484-9.

[12] Nair A, McGrew B, Andrychowicz M, Zaremba W, Abbeel P. Overcoming exploration in reinforcement learning with demonstrations. In: 2018 IEEE international conference on robotics and automation (ICRA). IEEE; 2018. p. 6292-9.

[13] Osa T, Pajarinen J, Neumann G, Bagnell JA, Abbeel P, Peters J, et al. An algorithmic perspective on imitation learning. Foundations and Trends® in Robotics. 2018;7(1-2):1-179.

[14] Osa T, Sugita N, Mitsuishi M. Online trajectory planning and force control for automation of surgical tasks. IEEE Transactions on Automation Science and Engineering. 2017;15(2):675-91.

[15] Kober J, Peters J. Policy search for motor primitives in robotics. Advances in neural information processing systems. 2008;21.

[16] Chaumette F, Hutchinson S. Visual servo control. I. Basic approaches. IEEE Robotics & Automation Magazine. 2006;13(4):82-90.

[17] Wang Y, Lang H, de Silva CW. A Hybrid Visual Servo Controller for Robust Grasping by Wheeled Mobile Robots. IEEE/ASME Transactions on Mechatronics. 2010;15(5):757-69.

[18] Krupa A, Gangloff J, Doignon C, de Mathelin MF, Morel G, Leroy J, et al. Autonomous 3-D positioning of surgical instruments in robotized laparoscopic surgery using visual servoing. IEEE Transactions on Robotics and Automation. 2003;19(5):842-53.

[19] Rohmer E, Singh SP, Freese M. V-REP: A versatile and scalable robot simulation framework. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE; 2013. p. 1321-6.

[20] James S, Freese M, Davison AJ. Pyrep: Bringing v-rep to deep robot learning. arXiv preprint arXiv:190611176. 2019.

[21] Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, et al. Openai gym. arXiv preprint arXiv:160601540. 2016.

[22] Core;. Available from: `https://www.gymlibrary.dev/api/core/`.

[23] Raffin A, Hill A, Gleave A, Kanervisto A, Ernestus M, Dormann N. Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research. 2021.

[24] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. arXiv preprint arXiv:170706347. 2017.

[25] Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, et al. Continuous control with deep reinforcement learning. arXiv preprint arXiv:150902971. 2015.

[26] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. nature. 2015;518(7540):529-33.