RAM ROBOTICS AND MECHATRONICS

Reinforcement learning based approach for the navigation of a pipe-inspection robot at sharp pipe corners

X. (Xiangshuai) Zeng

MSC ASSIGNMENT

Committee:
prof. dr. ir. G.J.M. Krijnen
N. Botteghi, MSc
dr. ir. E. Dertien
dr. B. Sirmaçek
dr. M. Poel

September, 2019

039RaM2019
Robotics and Mechatronics
EEMathCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Summary

The PIRATE is a Pipe Inspection Robot for AuTonomous Exploration currently being developed at the Robotics & Mechatronics (RaM) research group at the University of Twente. In this thesis, a reinforcement learning (RL) based approach for navigating the PIRATE robot to move through sharp pipe corners is designed and researched. The overall movement of PIRATE is broken down into two separate parts: the process for the front part of the robot and the process for the rear part of the robot, with each simulated by using a 4-DOF robotic arm that has a similar autonomy as half of the PIRATE. A laser scanner is installed on the end-effector in simulation in order to perceive its surrounding environment. Specifically, reinforcement learning is employed for developing the path planner for the front part of PIRATE and the training task is formulated as letting the 4-DOF robotic arm reach a given target inside pipe-like obstacles. Furthermore, two supplementary approaches are developed, with the first one for letting the robot locate a target point in the pipe by itself and the second one as a navigation policy for the rear part of PIRATE to move through the corner of pipes. The running RL algorithm in this thesis is chosen to be Proximal Policy Optimization (PPO) and deep artificial neural networks are deployed as the function approximators in the algorithm. Most of the experiments are done in simulation where the software environment is established with Robot Operating System (ROS) and Gazebo simulator. In addition, a real robotic setup is also built for evaluating the proposed approaches in the real world.

During the experiments, the performance of the RL agent is exploited under torque control scheme and position control scheme respectively. It is found that the resulting agent can be generalized to navigate the robot inside multiple different environments; the laser data plays an important role on whether the agent can find an optimal policy. In addition, the RL agent performs better in general as the robot is controlled with torque commands than with position commands, but including the information from the past and an extra penalty on the change of the joint positions helps improve the performance of the agent under position control. Next, the two supplementary approaches are evaluated and are both proven to be effective with a fairly acceptable success rate. Finally, the proposed approaches are assessed onto the real robotic setup to observe the differences between the simulation and the real world.

Contents

1 Introduction

1.1 Context

Industrial pipelines need to be regularly inspected and maintained in order to guarantee a safe and reliable usage. Typical inspection tasks include locating defects and deformations, and verifying the wall thickness. However, pipelines with turns, splits or sections with varying diameters cannot be inspected by a PIG (Pipeline Inspection Gauge) and currently have to be examined from outside by technicians. This process usually takes quite some time since the technicians will have to first remove the isolation layers that are covering the pipes before the inspection can be done, which adds a large amount of extra work.

To make the inspection safer and more cost-efficient, the Pipeline Inspection Robot for AuTonomous Exploration (PIRATE) is being developed at the Robotics & Mechatronics (RAM) group in the University of Twente so that the robot can be placed inside the pipelines and carry out the inspection autonomously. A latest iteration of PIRATE as well as a model of the prototype is shown in figure 1.1.

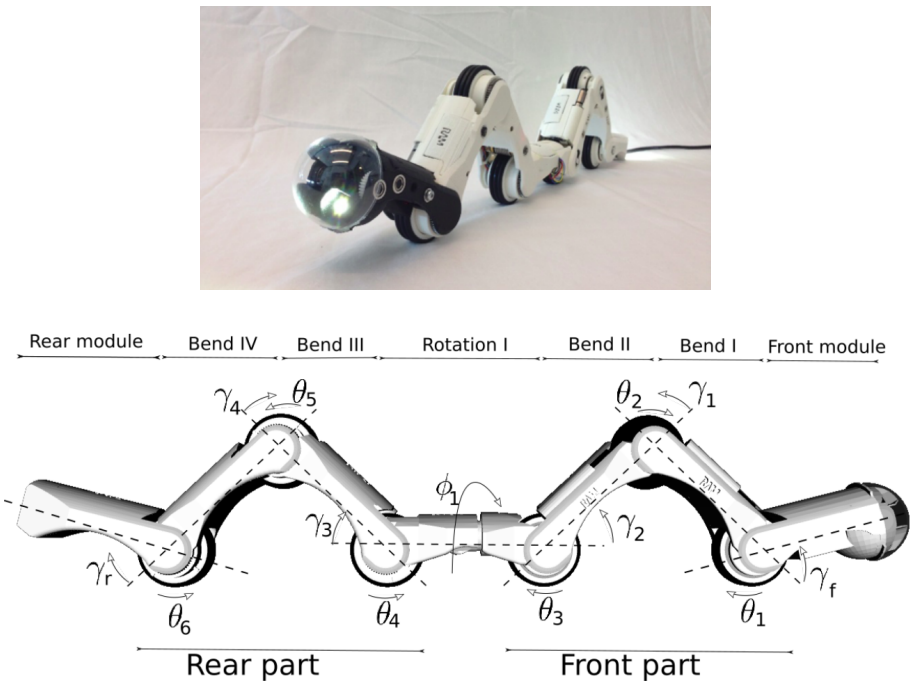


Figure 1.1: Upper: a realization of PIRATE model; Lower: the kinematic model of the PIRATE Prototype II, where γ, θ and ϕ represent the angles for the bending between links, the orientation of wheels and the rotation between the two parts respectively[3]

As can be seen, the PIRATE (Pipeline Inspection Robot for Autonomous Exploration) consists of six bending joints, six wheels and one rotational joint. When the robot is placed inside a pipe, the wheels will have contact with the wall and drive the whole body of the robot to move. Based on the dimensions of the current pirate robot the range of pipe diameter that the robot can navigate through is $70 \sim 120mm$. Meanwhile, the pipes also have sharp corners, T-junctions and other configurations, which gives many restrictions on the behaviour of the robot. In principle, PIRATE should be able to move through these segments without getting stuck.

1.2 Problem statement

The development of PIRATE is part of the Smart Tooling project, aiming towards an autonomous exploration inside industrial pipelines. One critical issue regarding the navigation

of PIRATE arises when the robot needs to make turns at a sharp corner. Figure 1.2 shows a sequence of images illustrating the process when PIRATE moves through a pipe corner with 90° of bending angle. Basically, the movements of PIRATE in the images can be divided into the following different steps:

(a) Unclamp the front (image 3)

(b) Move the front through the corner (image 4, 5, 6)

(c) Clamp the front and unclamp the rear (image 7)

(d) Move the rear through the corner (image 8, 9)

(e) Clamp the rear (image 10)

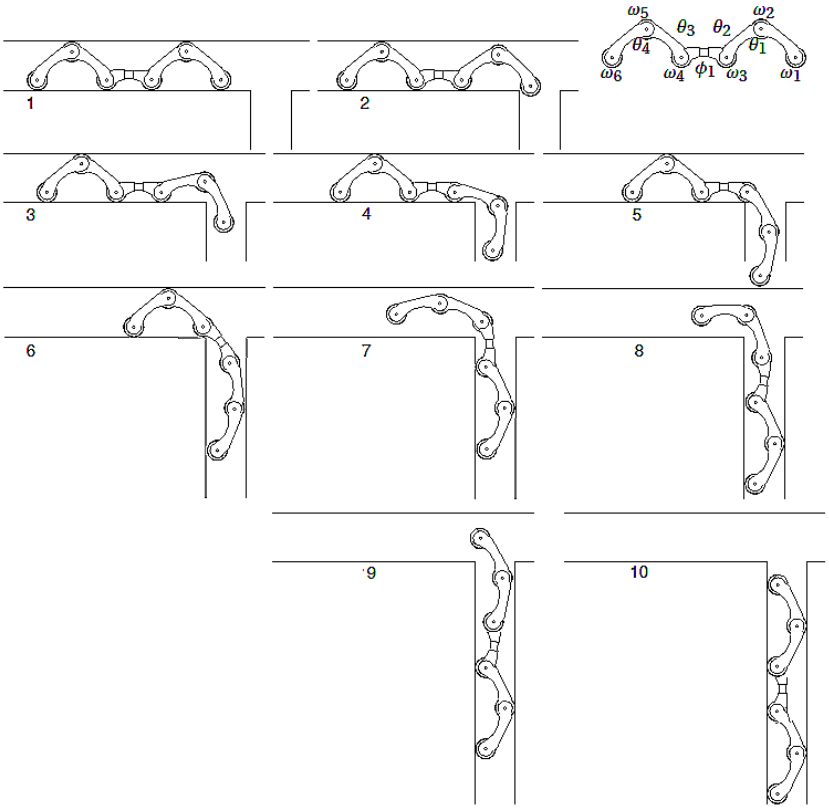


Figure 1.2: Turning sequence of PIRATE at a sharp corner with 90° [1]

The clamping and unclamping behaviours of PIRATE in step (a) (c) (e) are simple to design and can be easily applied into pipes with various diameters. However, the robot needs to bend its joints in different ways in order to move its links through different kinds of pipe corners in step (b) and (d). This adds considerable difficulties in the design of the high-level path planner. Precomputing the trajectories cannot work since the robot is not able to perceive the exact angle and diameter of the pipe corner, thus, an approach that navigate the robot to move through various different pipe corners need to be developed.

One possible solution is to let the robot “discover" an optimal strategy by leveraging reinforcement learning. Reinforcement Learning (RL) is a machine learning technique where an agent is trying to find an optimal control policy for a certain task by continuously interacting with the environment and improving the policy with the gained experience. Reinforcement learning does not require any prior knowledge about the environment so it provides a general framework which is suitable for different kinds of scenarios, such as video games, robotics tasks, web

system configuration, etc. Under the scope of this project, the agent is the path planner for PIRATE while the environment includes the robot with its surroundings.

1.3 Related work

PIRATE was first designed and prototyped in [1], where several mechanical structures of the robot and multiple control frameworks were specified. In [2], another software framework was also designed for increasing the autonomy of the robot.

In the both cases above, PIRATE is controlled by manual inputs via a MIDI panel where each part of the robot, such as bending and rotational joints, wheels, cameras, etc., is controlled separately. Hence, the behaviour of PIRATE when it moves through a sharp pipe corner (figure 1.2) is specified by a human operator who has to observe the robot from the outside of the pipe (in these cases the pipes used for the experiments are transparent [1]) and control the robot joint by joint in order to realize a successful passing.

Later in [3], the autonomy of PIRATE is further increased by adding a high-level control layer and the robot can autonomously move through a 90° sharp bend inside a 2D pipe with a diameter which is fixed and known a priori. No other pipe configurations were tested so [3] only proposed an approach for the robot to move through one kind of pipe corners.

The possibility of leveraging reinforcement learning to solve the PIRATE's turning problem was first explored by [4]. In this work, a planar robotic arm with 3 rotational joints is trained under the framework of RL so that it can reach a given target with the presence of pipe-like obstacles. This robotic-arm setup mimics the behaviour of the front part of PIRATE after it unclamps from the pipe and starts to move through the sharp corner, as illustrated in image 4, 5, 6 in figure 1.2. The results in [4] shows that the robotic arm is able to reach given targets inside sharp bends with acute, right and obtuse angles as well as T-junctions. However, the obtained learning-based path planner needs to be re-trained every time a different environment is encountered.

1.4 Objectives

The objective of this project is to develop a robust navigation policy for PIRATE so that it can move through sharp pipe corners by leveraging reinforcement learning. The experiments should focus on evaluating the generalizability of the resulting approach to see how it would perform in different environments, i.e., inside pipes with different diameters and turning angles.

However, the experiments will not be conducted with the PIRATE robot either in simulation or in real world due to the fact that there are currently no reliable models available at the RAM lab of the complete robot; instead, a similar strategy as in [4] will be employed that a 4-DOF planar robotic arm with 3 rotational joints and one translation DOF will be built and employed as the working robot. The reasonability of deploying this model-changing will be justified in chapter 3.

The simulation environment is decided to be built under the ROS (Robot Operating System) framework due to its high flexibility; the reinforcement learning algorithm is chosen as Proximal Policy Optimization and will be implemented with Python Tensorflow . In the end, the resulting navigation policy will be tested onto a real 4-DOF robotic arm adapted from the work of [4].

1.5 Report outline

The reminder of this thesis is structured as follows:

In **chapter 2** the background about reinforcement learning, deep reinforcement learning and ROS/Gazebo is briefly introduced. Then in **chapter 3** the approaches employed in this thesis

are proposed and analyzed. Furthermore, in **chapter 4** the design and implementation of the simulation and the real world environment as well as the realization of the reinforcement learning algorithm are presented in details. Moreover, the setup and goal of each experiment conducted in the project is briefly introduced. Later, in **chapter 5** the corresponding experiment results are presented with a general discussion at the end. Finally, in **chapter 6** the conclusions for the whole project and the possible future works are drawn.

2 Background

2.1 Reinforcement learning

Reinforcement learning (RL) is a branch of machine learning techniques where an agent is enabled to learn how to behave by interacting with the environment and gain experiences by trial and error. It is inspired from the nature of how human beings and animals learn: we explore the environment by executing some actions and perceive it by using our eyes, ears and the signals from touching an object. By observing what the environment responses to us, we adjust our behaviours in order to reach some goals.

In this section, some basic but important concepts of reinforcement learning will be introduced together with several different branches of RL algorithms.

2.1.1 General introduction

To form a reinforcement learning problem setup, an agent and an environment need to be defined and connected. An agent is normally a high-level controller for a game, a robot or a simple machine, while the environment is everything else the entire game, or the robot with its surroundings. It is important to understand how the agent and the environment interacts with each other, which can be interpreted by the picture below:

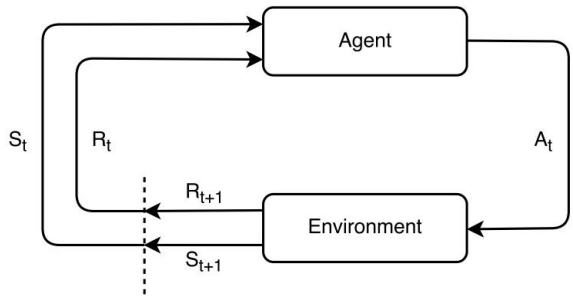


Figure 2.1: Basic concepts of reinforcement learning [5]

At any time step t, the agent will take an action A_t in the environment based on its observation containing the message about the state S_t of the environment. The action will then have an influence on the environment which will change into the next state S_{t+1} and also response with a reward signal R_{t+1}. Meanwhile, the agent can improve its policy of selecting action based on the reward signals. By executing this closed loop circle consecutively, the agent will manage to find an optimal strategy so that a maximum cumulative rewards can be achieved.

Here are some other important concepts deployed in the reinforcement learning framework:

Markov Decision Process

It seems skeptical that the agent takes an action only based on the current state, since the information from the history may also influence the future of the environment. This is because Reinforcement learning problems are defined in terms of a Markov Decision Process (MDP).

The whole idea of MDP is built on the so-called Markov property, which basically states that:

"The future is independent of the past given the present."

In mathematical terms, a state S_t has Markov property if and only if:

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_0, S_1, ..., S_t) \tag{2.1}$$

The current state captures all the information from the past.

Therefore, a Markov Decision Process defines an environment in which all the states S_t are Markov. It consists of a tuple $< S, A, P, R >$ where:

- S is a finite set of states.
- A is a finite set of actions.
- P is the state transition probability matrix,

$$P^a_{ss'} = P[S_{t+1} = s' | S_t = s, A_t = a] \tag{2.2}$$

- R is a reward function, where $R^a_{ss'}$ denotes the immediate reward received after transitioning from state s to state s' after taking action a.

Policy

A policy is how the agent selects actions based on the state, i.e., it gives a map from the state space to the action space. There are two kinds of policies, stochastic and deterministic, where the former one outputs a probability distribution of the action over the given state and the latter directly gives a certain action. Mathematically, these two types of policy can be expressed as:

- Stochastic policy:

$$\pi(a|s) = P[A_t = a | S_t = s] \tag{2.3}$$

- Deterministic policy:

$$\pi(a|s) = F(s) \tag{2.4}$$

  where F is a deterministic function.

Trajectories

A trajectory is a sequence of states, actions and rewards that have happened during the interaction between the agent and the environment.

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, s_2 ...) \tag{2.5}$$

The very first state s_0 is randomly sampled from the start-state distribution denoted as ρ:

$$s_0 \sim \rho_0(\cdot) \tag{2.6}$$

A trajectory is frequently called an **episode**. It can be either finite or infinite, where in the latter case the episode will never be ended.

Return

The goal of the agent is to maximize some notion of cumulative reward called return over a trajectory:

$$G(\tau) = R_0 + R_1 + ... = \sum_{t=0}^{T} R_t \tag{2.7}$$

represents the return starting from the time step t until the episode is over; T denotes the end of an episode. Sometimes a discounted factor $\gamma \in (0, 1)$ is integrated into the formulation of the return:

$$G^\gamma(\tau) = \sum_{t=0}^{T} \gamma^t R_t \tag{2.8}$$

The motivation of using the discounted factor is to let the agent value the rewards from the current more than those from the far future.

The RL problem

The objective of a RL agent is to find an optimal policy π^* which maximizes the **expected return**.

For instance, suppose both the state transitions and the policy is stochastic. Then the probability of a T-step trajectory is:

$$P_\pi(\tau) = \rho_0(s_0)\prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{2.9}$$

The expected return, denoted as $J(\pi)$ is expressed as:

$$J(\pi) = \int_\tau P_\pi(\tau)G(\tau) = E_{\tau\sim\pi}[G(\tau)] \tag{2.10}$$

Then the optimization problem in reinforcement learning can be formulated as:

$$\pi^* = \arg\max_\pi J(\pi) \tag{2.11}$$

Goal

A goal in reinforcement learning is a high-level objective that the agent needs to be achieved and is directly related to the optimization of the reward function. For instance, if the goal of the agent is to let a robot move as fast as possible, then the reward function can be defined to be proportional to the value of the robot velocity. By maximizing the expected accumulative rewards, the agent can find an optimal policy (if possible) that controls the robot to move in a high speed.

Value function

In order to take a proper action at a certain time step, it would be helpful for the agent to know how good the current state is. A way to “measure" the goodness of a state is by means of the value function. In short, a **state value function** is defined as the expected return starting from a state s while following a particular policy π,

$$V^\pi(s) = E_{\tau\sim\pi}[G(\tau)|s_0 = s] \tag{2.12}$$

Similarly, an **action-state value function** is defined as the expectation of the return after taking an action a at state s while following policy π,

$$Q^\pi(s, a) = E_{\tau\sim\pi}[G(\tau)|s_0 = s, a_0 = a] \tag{2.13}$$

It measures how good a state-action pair (s, a) is and is often referred as Q *value*.

Advantage

The difference between the state-action value function and state function is called *advantage*. It basically measures how much better or worse of an action compared to the value of the current state.

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \tag{2.14}$$

Accordingly, if $A(s, a)$ is greater than zero, we think the action is a "good" one, and vice versa.

2.1.2 Policy iteration

The core of reinforcement learning is to learn an optimal policy so that the expected accumulative rewards can be maximized. The process of the learning is developed upon the idea of *Generalized Policy Iteration* (GPI). GPI is an iterative scheme which consists of two steps: the first one tries to evaluate how good the current policy is, known as *policy evaluation*; while the second one updates the policy in the direction of improving the evaluation, known as *policy improvement*. By executing these two steps iteratively, an optimal policy can be learned.

Based on how the policy iteration is formulated, the reinforcement learning algorithms can be classified into three categories: value function based, policy based and actor-critic, which will be briefly introduced below.

2.1.2.1 Value function based approach

A general structure of value function based reinforcement learning is illustrated in figure 2.2. It can be seen that the policy, denoted by π, is evaluated by calculating the value function V_π measuring the “goodness" of each state or state-action pair. Meanwhile, the policy itself is actually a greedy behavior over the value function, which means the action a_t is selected based on the optimum value of V_π.

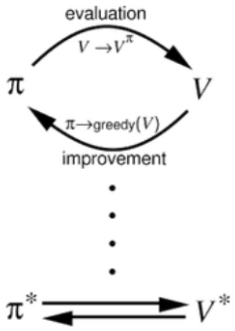


Figure 2.2: Value function based policy iteration[5].

Popular value-based RL algorithms include: *Monto Carlo method* ,*SARSA (State-Action-Reward-State-Action), Q learning, double Q learning,* etc. Apart from *Monto Carlo method*, all the other value based approaches are developed on the idea of Temporal difference (TD value). Shortly speaking, these methods calculate a temporal error, which is the difference between a new estimate and an old estimate of the value function, by considering the reward receives at the current time step to update the value function.

The simplest TD method, known as *TD(0)*, can be expressed mathematically as below:

$$V(S_t) \leftarrow V(S_t) + \alpha(r_t + \gamma V(S_{t+1}) - V(S_t)) \tag{2.15}$$

where α is the learning rate and γ the discount factor. The value function in the equation can either be state value function $V(s)$ or state-action value function $Q(s,a)$.

Take *Q learning* as an example. The pseudo-code of the algorithm is shown in algorithm 1. It can be seen that a state-action value function $Q(s,a)$ is continuously evaluated with *TD* method during the learning while the policy is to perform a ϵ-greedy (greedy selection with a probability of $1-\epsilon$ and random selection with a probability of ϵ) over Q so that the exploitation of “good" actions can be reinforced without a lack of exploration on unseen behaviors.

Algorithm 1 Q learning[5]

Initialize $Q(s,a), \forall s \in S, a \in A(s)$ arbitrarily, and $Q(terminate-state,\cdot) = 0$
for each episode **do**
  Choose initial state s_0
  for each time step t **do**
    Choose a_t from s_t using policy derived from Q (e.g., ϵ-greedy)
    Take action a_t, observe r_t, s_{t+1}
    Update the policy: $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)]$

Although TD method introduces bias in the estimation of the value function, it reduces the variance so that the policy evaluation step is faster and more stable, which is the main reason why TD method is widely researched in reinforcement learning.

2.1.2.2 Policy based approach

One of the most significant differences between policy based and value based RL is that the former deploys a parameterized policy $\pi(a|s,\theta)$ represented by a differentiable function approximator with a parameter set θ.

The second difference is that instead of calculating some value function, the policy is directly evaluated by computing the expected sum of rewards, denoted as J, under the policy. Mathematically, J can be expressed as:

$$J = E_{\tau \sim P_\theta(\tau)}[\sum_t r(s_t, a_t)] \tag{2.16}$$

where τ represents a state-action trajectory $(s_0, a_0, ..., s_T)$ and $P_\theta(\tau)$ corrsponds to the probability of having τ by following policy $\pi(\theta)$.

According to [5], the gradient of the return can be calculated as:

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(\sum_{t=1}^{T}\nabla_\theta log\pi_\theta(a_{i,t}|s_{i,t})(\sum_{t=1}^{T} r(s_{i,t}, a_{i,t})) \tag{2.17}$$

Here N represents the number of trajectories and by averaging over different trajectories, an approximation of the expectations can be roughly obtained. Therefore, it is natural to apply gradient ascent to update the policy towards the direction of increasing J, which is exactly what the REINFORCE algorithm does:

Algorithm 2 REINFORCE algorithm

1. sample several trajectories τ^i from $\pi_\theta(a_t|s_t)$
2. $\nabla_\theta J(\theta) \simeq \sum_i(\sum_t \nabla_\theta log\pi_\theta(a_t^i|s_t^i))(\sum_t r(s_t^i, a_t^i))$
3. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

There are mainly two advantages of policy-based RL: first, it can learn stochastic policy which valued-based methods cannot (note $\pi(a|s)$ itself represents the probability of taking action a at state s); second, policy-based RL is suitable for continuous action space due to the reason that the action is actually sampled over $\pi(a|s)$, resulting continuous values.

However, one major drawback of policy based algorithms is that the evaluation of the expected return suffers from a large variance and thus, making the updates of the policy unstable and slow, sometimes failing to find an optimal solution. To overcome this challenge, it is natural to consider the cooperation with value-based approaches since the advantage of *TD* method is that it significantly reduces the variance.

This combination of policy-based and value-based methods lead to the third category of reinforcement learning: actor-critic.

2.1.2.3 Actor critic approach

In the actor-critic RL setup, a parameterized policy (the “actor"), which continuously maps the state of the environment to the selected action, is improved in the direction suggested by a value function (the “critic") updated by using *TD* method. Those two models participate in the learning process where they both get better in their own role: the actor learns an optimal policy while the critic learns the optimal value function.

Therefore, the actor-critic method has both the advantages of value-based and policy based algorithms:

1. Low variance in the estimate of the expected sum of rewards, resulting stable convergence.
2. Suitable for continuous action space

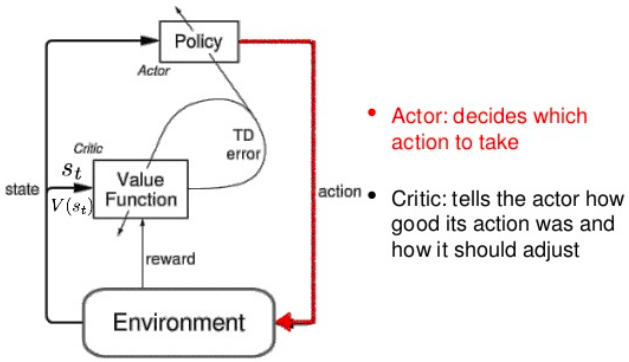


Figure 2.3: Structure of actor-critic RL[6]

3. Learning of stochastic policy

Almost every newly-developed reinforcement learning algorithm incorporates the idea of actor-critic, such as A2C (Advantage Actor Critic), DDPG (Deep Deterministic Policy Gradient), TRPO (Trust Region Policy Optimization), PPO (Proximal Policy Optimization), etc. In the next section, the algorithms of TRPO and PPO will be broken down and introduced in detail since they are the core approaches employed in this thesis.

2.1.3 Proximal Policy Optimization

In the update of the parameterized policy $\pi(a|s,\theta)$, there is another vital problem that disturbs the training: the choosing of the learning rate. As can be seen from the third step of REINFORCE algorithms in algorithm 2:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

where the parameter of the policy θ is updated in the direction of the gradient of the expected return. It raises difficulties on choosing a proper learning rate α: if α is too small, the update of θ will become almost vanished when the gradient of the expected return is also small, which makes the convergence of the policy become desperately slow. On the other hand, if α is too large, the updates will become too drastic when the policy is located at a steep area in its function space so the training becomes very unstable. Furthermore, policy gradient methods often have very poor sample efficiency, taking millions (or billions) of time-steps to learn simple tasks.

Researchers have sought to solve these issues with approaches such as TRPO (Trust Region Policy Optimization)[7], ACER (Sample Efficient Actor-Critic with Experience Replay)[8] and PPO (Proximal Policy Optimization)[9], by either constraining or optimizing the size of one policy update. However, ACER is rather complicated, requiring the addition of code for off-policy corrections and a replay buffer; TRPO — though useful for continuous control tasks — is not easily compatible with algorithms that share parameters between a policy and value function or auxiliary losses and other domains where the visual input is significant[11].

PPO successfully makes a good balance between ease of implementation, sample complexity, and ease of tuning. There are two primary variants of PPO: PPO-Penalty and PPO-Clip. Here, we focus on PPO-Clip which will be employed as the running algorithm for this project.

Both PPO and TRPO are trying to maximize an objective function expressed as:

$$L(s,a,\theta_k,\theta) = E_{s,a\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a)\right] \tag{2.18}$$

It is a measure of how policy π_θ performs relative to the old policy π_{θ_k} using data from the old policy. In PPO-clip, this objective is limited by adding a clipping operation[12]:

$$\begin{aligned} L(s,a,\theta,\theta_k)_{clip} &= \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), g(\epsilon, A^{\pi_{\theta_k}}(s,a))\right) \\ g(\epsilon, A) &= \begin{cases} (1+\epsilon)A, & A \geq 0 \\ (1-\epsilon)A, & A < 0 \end{cases} \end{aligned} \tag{2.19}$$

The intuition behind this can be explained by inspecting a single state-action pair (s,a) and thinking of corresponding cases.

Advantage is positive: suppose the advantage for the state-action pair (s,a) is positive, then the objective in equation 2.19 reduces to:

$$L(s,a,\theta,\theta_k)_{clip} = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1+\epsilon)\right) A^{\pi_{\theta_k}}(s,a) \tag{2.20}$$

Because the advantage is positive, the objective will increase if the action becomes more likely—that is, if $\pi_\theta(a|s)$ increases. However, the min in this term puts a limit to how much the objective can increase. Once $\pi_\theta(a|s) > (1+\epsilon)\pi_{\theta_k}(a|s)$, the min kicks in and this term hits a ceiling of $(1+\epsilon)A^{\pi_{\theta_k}}(s,a)$. Therefore, the new policy does not benefit by going far away from the old policy[12].

Advantage is negative: suppose the advantage for the state-action pair (s,a) is negative, then the objective in equation 2.19 becomes:

$$L(s,a,\theta,\theta_k)_{clip} = \max\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1-\epsilon)\right) A^{\pi_{\theta_k}}(s,a) \tag{2.21}$$

Because the advantage is negative, the objective will increase if the action becomes less likely—that is, if $\pi_\theta(a|s)$ decreases. However, the max in this term puts a limit to how much the objective can decrease. Once $\pi_\theta(a|s) < (1-\epsilon)\pi_{\theta_k}(a|s)$, the max kicks in and this term hits a ceiling of $(1-\epsilon)A^{\pi_{\theta_k}}(s,a)$. Therefore, the new policy does not benefit by going far away from the old policy[12].

By integrating this clipping method, the incentives for the policy to change drastically is removed and the hyperparameter ϵ corresponds to how much the new policy can change from the old one while still profiting the objective.

The pseudo-code of PPO-Clip is presented in algorithm 3.

2.2 Deep Reinforcement Learning

From the previous algorithms it is noticeable that both value-based and policy-based reinforcement learning need a representation for either the value function V_ϕ or the policy π_θ. Therefore, it is natural to incorporate any powerful function approximator with RL methods, leading to one of the most promising research topics nowadays: deep reinforcement learning, the combination of deep artificial neural network and reinforcement learning.

2.2.1 Artificial neural network

Artificial neural networks (ANN) is a kind of computation systems inspired by the biological neural networks. It is a framework for many machine learning algorithms to work together and process complex data inputs. They have been applied in the area of image recognition, speech recognition, self-driving cars, etc. and have achieved state-of-the-art performance in these tasks.

An ANN is based on a collection of connected nodes called “artificial neurons", which loosely model the neurons in a biological brain. Each connection can transmit a signal from one node

Algorithm 3 Proximal Policy Optimization-Clip

1: Initialize policy parameters θ_0 and value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $D_k = \tau_i$ by running policy π_{θ_k} in the environment
4: Compute rewards-to-go $\hat{R}_t$
5: Compute the estimate of the advantage $\hat{A}_t$ by using any method of advantage estimation based on the current value function V_{ϕ_k}
6: Update the policy by maximizing the objective by using any advanced gradient descent method:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|D_k|} \sum_{\tau \in D_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), g(\epsilon, A^{\pi_{\theta_k}}(s,a)) \right) \tag{2.22}$$

7: Fit the value function by regression on mean square error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{D_k T} \sum_{\tau \in D_k} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2 \tag{2.23}$$

by using any advanced gradient descent method.

to the other while each node processes the received data and spread the signals to the other ones.

The architecture of a basic ANN can be expressed with the following image:

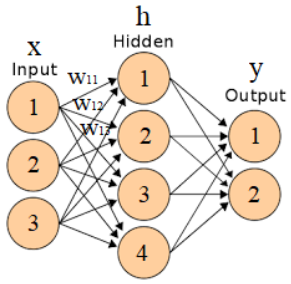


Figure 2.4: Structure of an artificial neural network

The network is fully connected and the first and last layer from the left is called the input layer (x) and the output layer (y) respectively; all the ones between are called hidden layers. The value of each neuron is calculated by a linear combination of the values of neurons from the previous layer plus an activation function. For instance, the value of the first neuron in the hidden layer can be computed as:

$$h_1 = g(W_{11} * x_1 + W_{12} * x_2 + W_{13} * x_3) \tag{2.24}$$

where g adds non-linearity to value transmission and is normally a differentiable function such as sigmoid function or Tanh function.

Sigmoid function:

$$g(x) = \frac{1}{1 + e^{-x}} \tag{2.25}$$

Tanh function:

$$g(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.26}$$

By adjusting the weights, the number of layers and neurons, ANN can approximates any continuous functions,either linear or non-linear according to the *Universal approximation theorem*[16][17][18]. Besides, it is also differentiable due to the reason that the connections between

each layer are basically linear combinations and the activation functions on each node are also differentiable. Therefore, ANN is compatible with many gradient descent methods which can be used to update the network with given data.

2.2.2 Deep reinforcement learning

Deep reinforcement learning (DRL) combines deep learning and reinforcement learning techniques to create powerful and efficient algorithms.

For instance, in actor-critic RL methods (the category which PPO belongs to) both the value function and the policy are represented with a fully connected neural network, and are called a value function net and a policy net respectively. The value function net takes the state of the agent as the input and outputs some kind of value function, depending on which algorithm it serves: in PPO, it is the value of that state; in DDPG (Deep Deterministic Policy Gradient) it is the optimal Q value (state-action value) of the state.

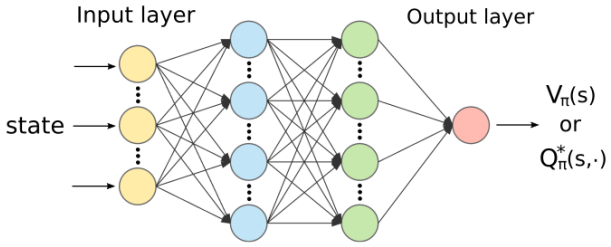


Figure 2.5: One possible configuration for a value function net

Similarly, the parameterized policy π_θ can also be represented by a deep neural network which works as a function that maps from the given state to a probability distribution over the actions. Figure 2.6 shows a policy network used for controlling the motion of a robot. The network takes the joint angles and kinematics information from the robot as input and outputs the mean values of a multivariate Gaussian distribution in the action space. With given standard deviations, the actual actions can be sampled from the distribution and used for the control of the robot at the current time step.

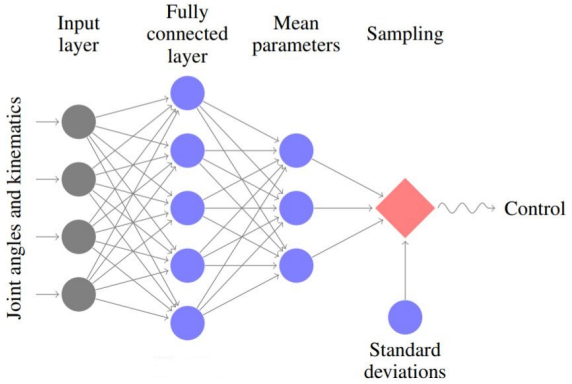


Figure 2.6: Policy network for the control of a robot[7]

There are several advantages of deep RL. First, unlike traditional RL where the value function is represented with a table and the state and the action space need to be discretized accordingly[5], deep RL allows continuous state and action space since ANNs are deployed. Second, deep neural networks have powerful representation capacity and are suitable to approximate the value function and the policy of an agent which are normally quite complicated; second, techniques in deep learning (such as various optimization methods) can be integrated with RL to make the learning more stable and more efficient.

2.2.3 Applications in robotics

The integration of deep reinforcement learning and robotics has become increasingly popular over the last few years due to both the advantages of reinforcement learning (RL) and deep neural network (DNN):

1. RL enables the robot to learn certain strategies by trial and error without the knowledge of the environment, saving tremendous effort on building mathematical models compared to traditional robot control approaches.

2. RL offers to robotics a framework and set of tools for the design of sophisticated and hard-to-engineer behaviors. [20]

3. General-purpose DNN is able to process complex sensory inputs from the robot, such as camera images, laser data, etc.

4. DNNs are powerful function approximators which can be used to represent any complex functions in robotics such as inverse dynamics, path planner, control strategies and so on.

In [21], a 18-DOF robotic arm (with a gripper) is trained to learn target-reaching and pick-and-place in simulation by employing several deep RL algorithms such as TRPO, Deep Deterministic Policy Gradient (DDPG), etc.; the trained agent which achieves the best performance is later deployed onto a real robotic arm and can still finish the tasks well.

In [22] , a method that allows multiple robots to cooperatively learn a single policy with deep reinforcement learning is presented. The method significantly speeds up the learning and is tested on a group of real robotic manipulators to seek for the solution of opening doors with a handle.

Furthermore, robotic platforms with a more complex structure are also researched to learn challenging skills with deep RL. In [23], motor behaviours such as locomotion for quadrupedal and getting up off the ground for the 3D biped are successfully learned in simulation environment by deploying TRPO together with a general advantage function estimation method. In [24], deep RL approaches even outperform traditional model-based methods on a real four legged robot in both the locomotion skill and recovering from falling.

2.3 ROS and Gazebo

2.3.1 General introduction

The simulation environment of this project is built with Gazebo simulator under a ROS (Robot Operating System) framework. A brief introduction to the chosen platforms is therefore presented.

Robotic Operation System (ROS) is a popular-used software suite for the building of autonomous robotic systems. The official introduction of ROS is as below [27]:

> *ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.*

Some fundamental concepts in ROS include: Node, ROS Master, Message, Topic and Service. A Node is basically an executable program which performs computation. Multiple nodes can

communicate with each other and form a network structure with the help of a ROS Master which provides naming and registration services to each individual node. During the communication, a node that needs to transmit information to the others will publish Messages which have particular data types into named Topics; meanwhile, any node that needs these information will subscribe to any of the Topics it is interested in and receive the corresponding Messages. Finally, a ROS Service is basically a Request/Reply mechanism between two nodes and is only triggered when it is needed.

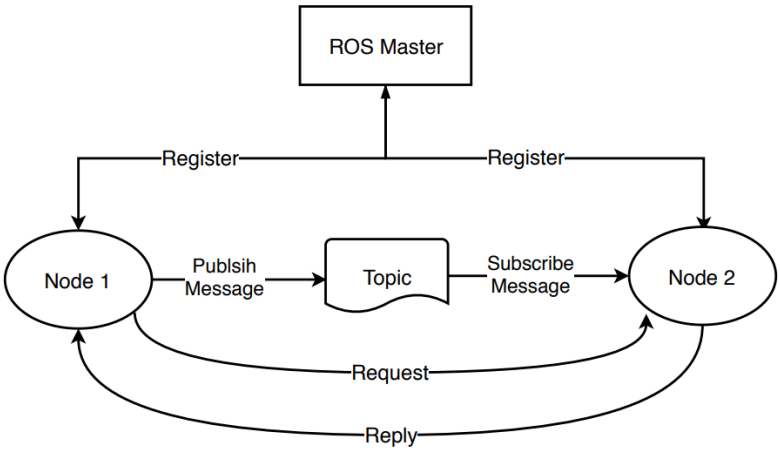


Figure 2.7: A basic software topology in ROS

ROS also provides a hardware interface between the simulation and the real platforms so that it is easy to realize sim-to-real transfer which is planned as the last section of the environment.

Meanwhile, Gazebo is a robot simulator which contains a robust physics engine, advanced 3D graphics, and convenient programmatic and graphical interfaces. It provides powerful solvers for dynamics and also various plugins where users can easily add different sensors such as laser scanner, IMU and odometer on their robots. Gazebo is highly integrated with ROS and is normally worked as a separate node together with other nodes, such as a controller or an AI agent.

2.3.2 OpenAI/ROS interface

The interface offers mainly three modules to facilitate the simulation:

1. **Gazebo connection**: provides control access to Gazebo environment with functions such as pausing and unpausing Gazebo, setting parameters, resetting the simulation, etc.

2. **Task environment**: defines the task that the agent is going to learn. It subscribes the data published by Gazebo and other ROS nodes, process the data to calculate the reward function r_t and formulate the new state s_{t+1}.

3. **General RL framework**: offers high-level API that wraps all the functionalities from the other two modules into two functions “Step" and “Reset", where the former one pass the action a_t to the simulation and receive the corresponding reward and the state for the next time step and the latter one resets the whole process, ending the episode.

3 Analysis and methodology

The first section in this chapter analyzes and justifies the decision of deploying a 4-DOF robotic arm as the simplified model for the PIRATE robot. Then, the other four sections introduce the main approaches employed within this thesis, together forming the methodology in which the robot is controlled to move through sharp pipe corners.

Specifically speaking, the RL-based navigation approach explained in section 3.2 is designed to control the front part of the PIRATE to pass the pipe corners, while in section 3.3 the corresponding RL elements are analyzed and selected. Next, the supplementary approach (I) introduced in section 3.4 makes the RL-based approach more realistic. Finally, the supplementary approach (II) in section 3.5 tackles with the situations when the rear part of the robot needs to move through the corner of the pipe. The whole methodology will be evaluated in several experiments presented in section 4.4.2 and chapter 5.

3.1 Robot simplification

As introduced in chapter 2, in reinforcement learning the agent needs to interact with the environment for a number of times episodically to find the optimal policy. This means the collection of data should be fast and efficient and resetting the episode should not be difficult either.

However, sampling data on real-world robots can only be run in real time which is not fast enough for applying reinforcement learning within a feasible time. In addition, safety issues should also be considered at the beginning phase of the learning, since the agent will take many random actions to explore the unknown environment, which may lead to bizarre behaviours of the robots and cause damage to the platforms.

Therefore, it is chosen to design the RL path planner for PIRATE in simulation to make the tuning of algorithms and data collection more flexible and efficient.

In order to carry out the simulation, models of PIRATE and different pipes are required to be built. However, as can be seen in figure 1.1, PIRATE possesses a complex mechanical structure with not only multiple links and joints, but also several wheels which are employed to drive the robot to move inside the pipe by contacts. Therefore, the modeling of PIRATE is rather difficult and will probably cost a considerable amount of time and effort, which should not be the focus of this project. An alternative model needs to be made.

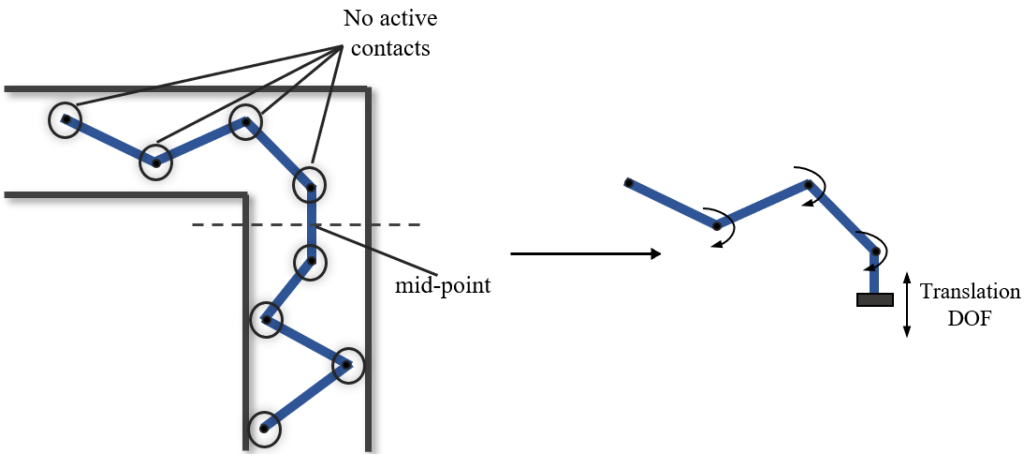


Figure 3.1: A sketch of PIRATE's turning left in a pipe. The upper part of the robot does not have active contacts with the pipe and can be considered as a robotic arm with 3 rotational joints and one translation degree of freedom at the bottom.

However, if we divide the PIRATE from its middle point and neglect the wheels, each half part of the robot can be roughly approximated as a robotic arm with 3 joints and a translation degree of freedom at the other side of the end-effector. The omission of the wheels makes sense when PIRATE take turns in the pipe as shown in figure 3.1: the upper half part passing through the corner does not have to actively use the wheels as long as the joints can bend in such a way to make it go across the turning while the lower half part performs a translation movement along the pipe to “push" the mid-point.

Moreover, figure 3.1 actually depicts two different phases when PIRATE is making turns. If the upper part of the robot in the figure is considered as the front part, it describes the situation when the “head" of PIRATE is passing the corner; on the other hand, it can also be explained as that the “tail" of the robot is following the front part which has already gone through the turning of the pipe. At both the phases, there is one part of PIRATE whose wheels do not have active contacts with the pipe while those of the other part supply forces to perform an overall translation movement.

Therefore, it is decided to simplify the model of PIRATE into **a robotic arm with 3 rotational joints and an extra translation DOF at the bottom**, which should be enough to capture the whole process when the robot is passing through the corner of a pipe as illustrated in figure

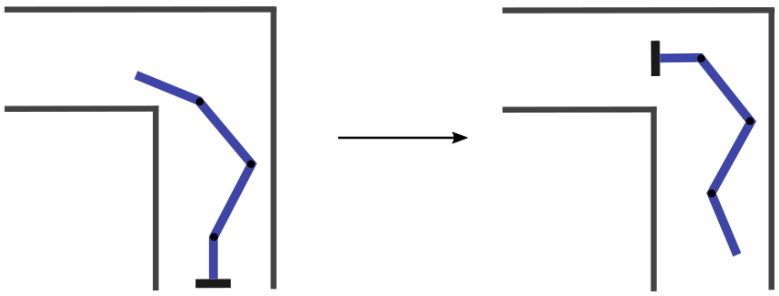


Figure 3.2: A whole image of PIRATE moving through pipe corners depicted by a 4-DOF robotic arm. The left figure shows the moment when the front part of PIRATE is about to pass the corner while the right one indicates a later moment when the rear part unclamps and starts its movement.

In the rest of the report, all the configurations and strategies will be discussed based on this simplified model.

3.2 RL-based navigation approach

The core approach researched within this thesis is an RL-based path planner for the front part of PIRATE. The reason of this preference is due to the fact that the front part of the robot usually has more difficulty moving through the pipe corners than the rear one. If the navigation problem for the front part gets solved, the solution for the rear part will not be a harder issue.

Formulate the learning task

It is important to define a criterion on whether the robot successfully makes turns in the pipe. Consider the robotic arm shown in figure 3.1, it can be regarded as already passing the pipe corner as long as the end-effector reaches any point that is located far enough in the other side of the pipe with respect to the corner.

Therefore, the task, or the goal, of the RL agent is formulated as **learning to make a 4 DOF robotic arm reach a given target point with the presence of pipe-like obstacles being surrounded**.

Adding perception

As described in the objective at section 1.4, the RL-based path planner should be able to navi-

gate the robot to take turns successfully in various pipes with different diameters and turning angles. Hence, the robot should have enough perception to be aware of potential differences of its surrounding.

One basic but important kind of information about the environment is the distance between the robot and the pipe. Ideally, all parts of the robot (links, joints, etc.) should keep a minimal distance away from the pipe in order to perform a successful turning. However, to achieve this complete perception of the environment the robot would require to install multiple sensors onto different links, which is not very realistic since it will significantly increase the weight, size and cost of the robot.

Based on the formulation of the learning task, a minimal requirement is to avoid collisions between the robot end-effector and the pipe as much as possible. On the other hand, it is acceptable for the “body" of the robot (other links) to touch with the pipe as long as a successful turning can be executed eventually.

Hence, it is decided to **install a 2D laser scanner** which can measure the distance of the objects from it within a certain range on the end-effector of the robotic arm to obtain minimal but sufficient perception.

Final design

Therefore, the final design of the environment where the RL agent should perform learning is described by the figure below:

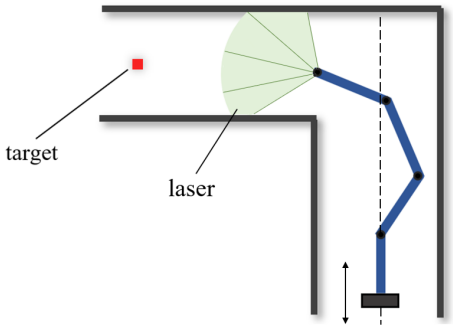


Figure 3.3: The final design of the environment for reinforcement learning: a planar 4-DOF robotic arm needs to reach a given target around the pipe corner with its end-effector on which a laser scanner is mounted. The dash line indicates the translation path that the bottom of the robotic arm can move on

3.3 RL elements selection

As mentioned in chapter 2, Proximal Policy Optimization (PPO) is chosen to be the learning algorithms for this thesis. There are two main reasons for making this decision. First, in [4] the author has shown that value-based reinforcement learning (Q learning, SARSA) is able to control a 3-DOF robotic arm to reach targets with the presence of pipe-like obstacles, but policy search and actor-critic methods have not been discussed yet. Second, researchers has shown that PPO is suitable of learning locomotion skills for robots as well as controlling multi-DOF robotic arms [9][13][14] while it is also easy to implement and scalable to large ANN.

3.3.1 State

The state of the environment should be chosen as to carry enough information about the environment from which the RL agent can induce proper actions. As shown in figure 3.3, the task for the robotic arm can be broken down into two parts:

- Reaching a given target with the end-effector.
- Avoid the collisions between the end-effector and the pipe.

For the target reaching, it is suggested in [15] that the state should contain the joint states (positions and velocities), position of the end-effector and position of the target. For the collision avoiding, the data from the laser scanner should be included so that the agent is aware of whether the end-effector is close to or far away from the pipe.

Therefore, the state of the environment can be expressed as the following formula:

$$s = [q^T \quad \dot{q}^T \quad p_{ee}^T \quad p_{goal}^T \quad D^T]^T \in \mathbb{R}^{22} \tag{3.1}$$

where $q = [q^1 \quad q^2 \quad q^3 \quad q^4]^T$ denotes the vector of the joint positions, $\dot{q}$ is the time derivative, p_{ee} and p_{goal} are the XY coordinates of the end-effector and the goal respectively and $D \in \mathbb{R}^{10}$ is the data from the laser scanner over a 180° range. Meanwhile, in [24] it mentioned that the joint state history was essential in training a locomotion policy; the authors hypothesize that it enables contact detection. Similarly, it is also possible that the history data from the laser points can help the RL agent detect the structure of the pipe in a more detailed manner. Therefore, an advanced version of state vector is formulated where the history data is integrated. At time step t, the state vector s_t is expressed as:

$$\begin{gathered} s_t^{advanced} = [q_t^T \quad \dot{q}_t^T \quad D_t^T \quad q_{t-1}^T \quad \dot{q}_{t-1}^T \quad D_{t-1}^T \quad a_{t-1}^T \quad p_{ee_t}^T \quad p_{goal}^T]^T \in \mathbb{R}^{44} \\ t = 0, 1, 2, 3... \end{gathered} \tag{3.2}$$

where a_{t-1} represents the action of the agent from the last time step, as suggested in [24]. When $t = 0$, the joint states and the laser data from the previous time step are chosen to be the same as the current one and the action is chosen as zero vector since there is no -1 time step. At the same time p_{goal} remains the same during the whole episode so the time notion is omitted.

3.3.2 Action

The action that the agent takes in the environment is chosen to be simply the control commands for the joints of the robotic arm. There are mainly two options: torque control and position control, where in the former one the agent directly send torque (force) signals to the joints while in the latter the actions from the agent are position commands sent to the joint controllers.

- Action for torque control:

$$a_{torq} = [\tau^1 \quad \tau^2 \quad \tau^3 \quad f] \in \mathbb{R}^4 \tag{3.3}$$

  where τ_i denotes the torque sent to the rotational joints and f the force command for the prismatic joint.

- Action for position control:

$$a_{pos} = [\Delta q^1 \quad \Delta q^2 \quad \Delta q^3 \quad \Delta q^4] \tag{3.4}$$

  It can be seen that the action is chosen to be the desired position change for each joint as an aim of limiting the movement of the robotic arm at the vicinity of the current configuration at each time step. Therefore, the set-points for the position controllers of the joints are calculated as:

$$q_{goal} = q + \alpha \cdot a_{pos} \tag{3.5}$$

  where $\alpha \in (0, 1)$ is used for smoothing the trajectory.

3.3.3 Reward functions

The selection of reward functions are crucial in reinforcement learning. It determines what kind of policy the agent is able to learn and also the convergence rate of the learning. For the simulation scheme shown in figure 3.3 there are mainly two reward functions to be designed: one for the task of target reaching and the other for collision avoiding.

A basic principle for choosing the reward function for target-reaching is to give a higher reward when the end-effector is closer to the target and vice versa. In [21] the reward is defined as the negative value of the linear distance between the end-effector and target (superscript *TR* denotes “target-reaching"):

$$r_{TR\_(1)} = -||p_{ee} - p_{goal}|| \tag{3.6}$$

In [22] a Huber loss function is cooperated with the linear distance to add some non-linearity to the part where the distance is close to 0:

$$r_{TR\_(2)} = \begin{cases} -\frac{1}{2}||p_{ee} - p_{goal}||^2, & ||p_{ee} - p_{goal}|| \leq \delta \\ \delta||p_{ee} - p_{goal}|| - \frac{1}{2}\delta^2, & otherwise \end{cases} \tag{3.7}$$

In fact, any monotonically increasing functions (with its negative as monotonically decreasing) can be cooperated with the linear distance and still follows the basic principle.

However, in [4] the author concludes that the value of the reward should not only increase as the end-effector approaches the target, but should increase faster as well. With a reward function following this property, the agent needs less time to find an optimal policy and the resulting policy can also lead the end-effector closer to the target.

Therefore, the reward function for the target-reaching task is determined by the equation below:

$$r_{TR\_(3)} = -\ln(\alpha * ||p_{ee} - p_{goal}|| + c) + \ln(c) \tag{3.8}$$

where α is for adjusting the slope and c is to guarantee the value is no more than 0. The formula in 3.8 guarantees an increasing of the slope when the distance between the target and the end-effector decreases. The values for α and c are then determined to be:

$$\alpha = 50, \quad c = 0.1 \tag{3.9}$$

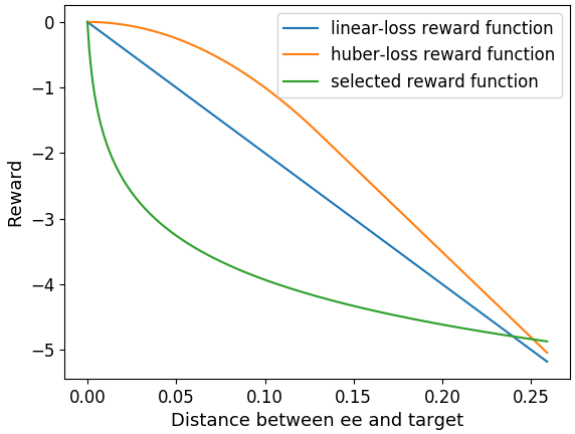


Figure 3.4: Different reward functions for target-reaching. Note the functions have been timed with different constants and scaled into a similar dimension. The reward function from equation 3.8 is the chosen one.

On the other hand, the reward function that keeps the end-effector away from the pipe should satisfy that the smaller the distance between the pipe and the end-effector is, the larger the

penalty should be, meaning the reward function should be inversely proportional to the data from the laser (in a negative sense). Hence, it is decided to use the following function to compute the reward (superscript *LD* denotes “laser data"):

$$r_{LD} = -\frac{1.0}{\beta * \min(D) + d_0} \tag{3.10}$$

where β is for modifying the shape of the curve, D the data from the laser scanner and d_0 is to prevent the occurrence of overwhelmingly large value when the end-effector hits the pipe (at this moment the data from the laser becomes extremely small). The parameters are then chosen to be:

$$\beta = 5.0, \quad d_0 = 0.005 \tag{3.11}$$

It can be seen from the curve that the value of the reward decreases drastically when the minimal value from the laser data approaches zero. This is designed for letting the RL agent “know" that it is dangerous to get very close to the pipe, but acceptable to keep some distance away.

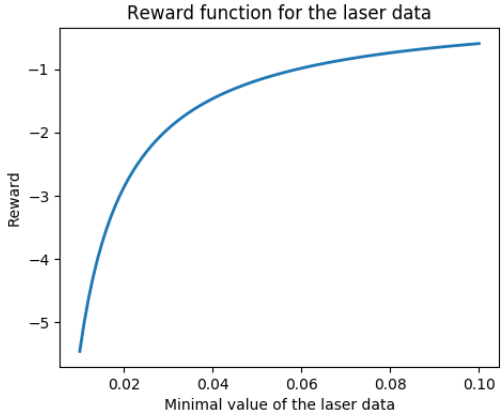


Figure 3.5: An example of the reward function on the laser data.

Furthermore, the movement of the joints of the robot should be limited under certain ranges such that no abnormal behaviour will occur (for instance, any rotational joint is not allowed to move over 180°). To accomplish this, the agent should be given a penalty once any joint of the robot reaches its limit; in addition, the episode should also be ended since the configuration of the robot must be strongly undesirable at this moment and any further movements are not desirable either; a new episode should be started.

Therefore, the total reward function in each episode can be expressed as:

$$r_{total} = \begin{cases} M & , \quad \textit{when any joint hits its limit} \\ C_1 \cdot r_{TR\_(3)} + C_2 \cdot r_{LD}\,, & \textit{other conditions} \end{cases} \tag{3.12}$$

where M is a negative constant and $C_i, i = 1,2$ are non-negative constants for adjusting the weights of each reward function. The criterion for selecting M is to make sure the cumulative reward of the episode is smaller when any joint hits the limit than that when it does not, so that the agent can “realize" reaching joint limits is a worse case than the others. Therefore, M should be a large negative value and is chosen as:

$$M = -1000 \tag{3.13}$$

Note equation 3.12 serves as a basic one which cooperates two kinds of behaviours for the agent. However, when the state vector is chosen as the more complicated one as shown in

equation 3.2, the reward function can also become more complex in order to let the RL agent learn sophisticated behaviour.

The basic assumption behind the state-history configuration is that the information from the past may help the agent infer the current situation of the robot and its environment more accurately. For instance, the joint position from the previous time step may contain the message about whether the robot is stuck in the pipe, since when the total change of the joints position is very small, it means that the robot almost stops moving and possibly, by being stuck at somewhere in the pipe.

Hence, an extra reward function is integrated into the original one to reduce the possibility for the robot of being stuck in the pipe. It is a function of the overall change of the joint positions and should follow the principle that the larger the change is, the higher the reward should be. Hence, the formula for this new part of reward function is chosen to have the same structure as the one for the laser data in equation 3.10:

$$r_{JC} = -\frac{1.0}{5.0 * J + 0.005}, \quad J = \sum_{i=1}^{4} |q_t^i - q_{t-1}^i| \tag{3.14}$$

where $q^i, i = 1,2,3,4$ refers to the position of each joint. As a result, the overall reward function is then changed into:

$$r_{total}^{advanced} = \begin{cases} M & , \quad \textit{when any joint hits its limit} \\ C_1 \cdot r_{TR\_(3)} + C_2 \cdot r_{LD} + C_3 \cdot r_{JC} , & \textit{other conditions} \end{cases} \tag{3.15}$$

3.4 Supplementary approach (I)

One issue regarding the RL-based approach described in section 3.2 is that the target position inside the pipe is given beforehand by the programmer, but in real world situations the robot needs to determine a goal by itself. However, the position of the target cannot be arbitrarily decided but should satisfy the following two requirements:

(i) The target must be located within the area which the robot has already detected.

(ii) The target should be located farther enough in the other side of the pipe with respect to the corner such that when the end-effector of the robotic arm (the front part of PIRATE) reaches it, the whole body of the robot can pass the corner at the same time.

As shown in figure 3.3, the only perception for the robot is the laser scanner at the end-effector. Therefore, as long as the target is located within the area that the laser can cover, requirement (i) should be satisfied. One way of computing the coordinate of a point from the laser data under the selected world frame can be described as follows:

Assume the configuration of the laser together with its frame is shown in figure 3.6. A point P is located on the i^{th} beam with a distance L away from the end-effector. The X and Y axis of the coordinate frame of the laser is aligned and vertical to the end-effector link. Assume the horizontal range of the laser is Φ (in degree) and there are totally N laser beams.

Therefore, the angle of the i^{th} laser beam under the laser frame can be computed as:

$$\beta = \frac{180 - \Phi}{2} + \frac{\Phi}{N}(i-1) \tag{3.16}$$

The coordinate of the point P is then calculated as:

$$\begin{aligned} P_x &= L \cdot cos(\beta) \\ P_y &= L \cdot sin(\beta) \end{aligned} \tag{3.17}$$

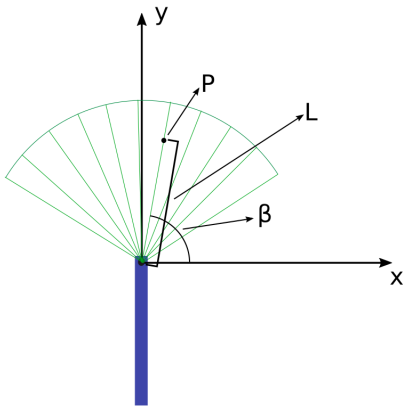


Figure 3.6: The laser scanner and the end-effector of the robot

Then, the absolute coordinates of the point P in the world frame can be computed by transforming the coordinates in equation 3.17 with a rotating and translating operation, which can be easily achieved by any corresponding method.

Next, to fulfill requirement (ii) the laser has to first scan over the other side of the pipe to determine a far-enough goal point before it starts executing the target-reaching task formulated in section 3.2. To design this “detecting" behaviour, there is one assumption that needs to be made: the robot should have already known whether it will need to turn left or right when it approaches the pipe corner, although it is not aware of the exact bending angle. This assumption is supposed to be realized in other software modules of PIRATE (for instance, mapping and localization module) and will not be discussed under the scope of this thesis.

Under this assumption and inspired by the video about PIRATE on [25], a “head-turning" behaviour is designed to help the robot locate a suitable target point. A sequential explanation for the behaviour is illustrated in figure 3.7. As shown, in image (a) the robot is preparing to turn left; next in (b), the head link (the link with the laser) is gradually turned to the left and at the same time the whole body of the robot is moving forward to get closer to the corner, the other two rotational joints remain at the initial positions; finally in (c) the head-link faces to the other side of the pipe and the laser detects an open space, then a target point (the red dot) is determined by using the method described in figure 3.6.

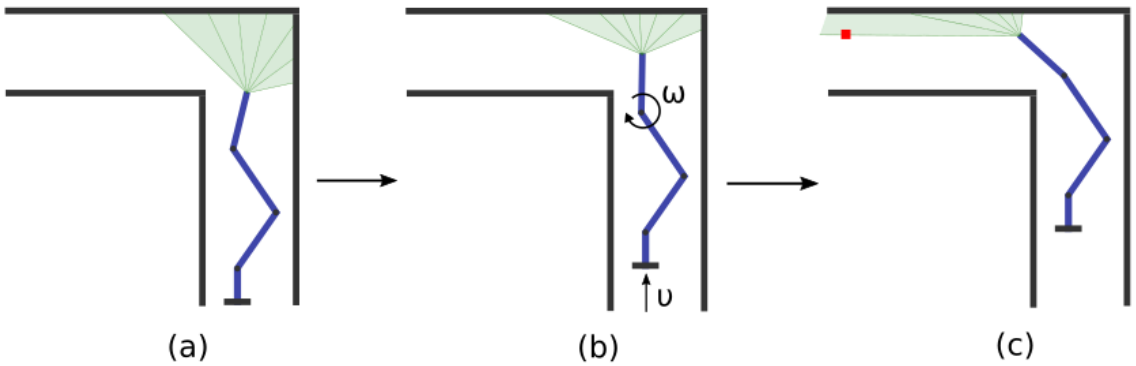


Figure 3.7: The process of locating a target when the robot is turning left in the pipe

Figure 3.7 depicts the process when the robot is turning left in a 90° pipe corner. However, in principle this approach should also be able to find a target inside pipes with acute and obtuse turning angles. Therefore, the translation velocity v of the prismatic joint at the bottom and the rotational velocity ω for the joint that connects the head-link need to be carefully designed.

For the v, it should not be too large otherwise the end-effector will quickly hit the pipe and cause the head-link to stop rotating further; on the other hand, if v is too small, it will take too much time before the laser can face to the other side of the pipe and detect an open space. Therefore, one possible pattern for the translation behaviour is to have a relatively large velocity when the end-effector is far away from the pipe, then drastically decrease the velocity if the pipe and the end-effector are too close to each other.

For the ω, it should be small to prevent the head-link hitting the left side of the pipe strongly, but not too small otherwise the laser cannot face to the left before the end-effector touches the pipe.

As a result, it is decided to employ a constant and small ω for the whole process and a constant v for the moments when the minimum value of the part of the laser data which represents the distances in front of the end-effector is larger than a threshold; if the threshold is crossed, start decreasing v drastically. Mathematically,

$$\begin{aligned} \omega_t &= \omega_0 \\ v_t &= \begin{cases} v_0, & \min(D_{la}) > \delta \\ \frac{v_{t-1}}{N}, & \min(D_{la}) < \delta \end{cases} \end{aligned} \tag{3.18}$$

where $t = 0, 1, 2, ...$ denotes the current time step, D_{la} the part of the laser data which represents the distances in front of the end-effector, δ a chosen threshold and N the decreasing ratio. The value for each parameter in the equation will be tuned and determined in the experimental part.

In addition, an open area in the pipe is detected when:

$$\exists d_{la} \in D_{la}, d_{la} > L_o \tag{3.19}$$

which means if any data point from the laser scanner is larger than a threshold, it should be considered that an open space is sensed.

3.5 Supplementary approach (II)

Note the configuration in figure 3.3 only covers the process when the front part of PIRATE is moving through pipe corners; an approach that tackles with the situations for the rear part should be designed.

In section 3.1 it is assumed that the front part and the rear part of PIRATE is symmetric around the mid-point of the robot (which is actually a feasible approximation for the real case), that is why a robotic arm can be used as a simplified model. Similarly, the movements for the two parts when moving through pipe corners, as illustrated in figure 3.2, also have some “symmetry" within each other; in fact, the process for the rear part of the robot when it moves through a pipe corner that goes left is just like a reversed process for the front part passing a corner that goes right. The two processes are reverted and mirrored to each other.

Therefore, one possible solution for the motion of the rear part of PIRATE is to record the trajectory of each joint on the front part along its moving through the pipe, then transform the data and apply them to the joints of the rear part so that it can also pass the corner.

The solution becomes simple to realize when the robot gets controlled under position controllers since the position signals can be easily transformed to make the trajectory of the robot inverted and mirrored. Reversing the trajectory of the robot can be done by simply reverse the order of the position data for each joint. Mirror the robot configuration can also be easily achieved by replacing each position value with its addictive inverse if the initial positions for the joints of the robot is particularly selected, as explained in figure 3.8. Of course, for the trajectory of the translation joint, a reversing transformation will be enough.

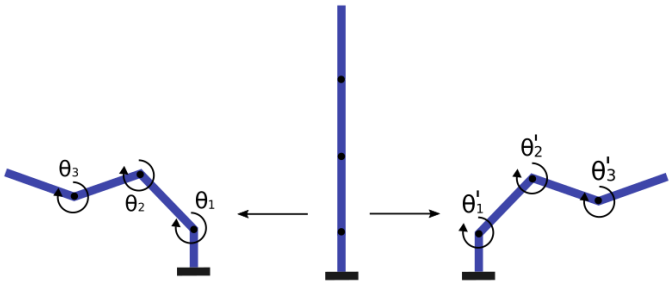


Figure 3.8: When the configuration of the robotic arm shown in the middle is selected as the initial one, then any two configurations which are geometrically symmetric to each other should satisfy that: $\theta_1 = -\theta'_1, \quad \theta_2 = -\theta'_2, \quad \theta_3 = -\theta'_3$

The resulting approach is illustrated in figure 3.9. It should be mentioned that the prerequisite of this solution for the rear part of PIRATE is that the RL-based navigation approach for the front part can have a satisfactory performance under position control. Therefore, the experiments which validate this approach will be designed and conducted after the ones for the front part of the robot is finished.

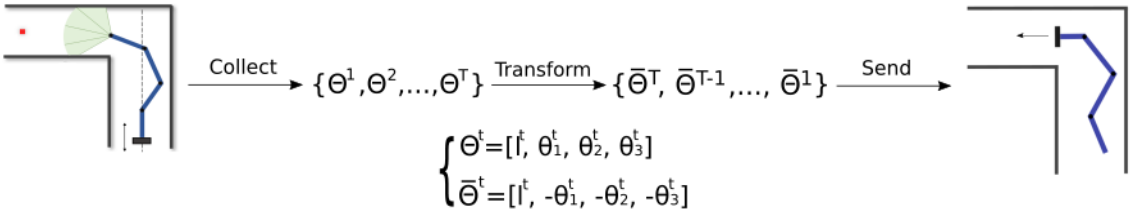
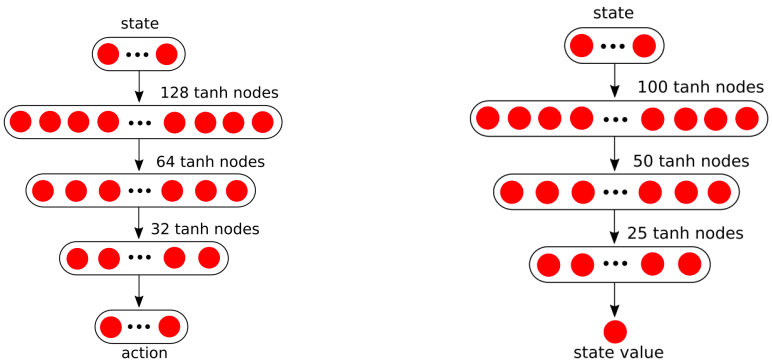


Figure 3.9: The basic scheme that may navigate the rear part of PIRATE to move through sharp pipe corners.

4 Design and Implementation

4.1 RL Algorithm realization

The algorithms of PPO are realized with Python Tensorflow. The codes are adapted from [29] [30] where the advantage function A_t is estimated by using “General Advantage Estimation" (GAE). Most of the hyperparameters remain the same as in the original implementation except the topology of the policy network and the value function network are specifically redesigned, shown in figure 4.1.



(a) Policy net **(b)** Value function net

Figure 4.1: Topology of the policy net and value function net

It should be pointed out that there is no standard procedure for selecting the topology of a fully connected neural network. However, the wider and deeper the network is, the more powerful its representation capacity will be since any smaller network can be represented with a bigger one by setting certain weights as zero. The topology of the policy net in figure 4.1 is designed based on [7][15][24] where a network activated with $tanh$ function and possessed with a structure of decreasing numbers of neurons layer after layer is often employed for the learning of robot locomotion skills. Meanwhile the value function network is simply added with an extra hidden layer compared to the original implementation in [30] for the aim of increasing its expression capacity. The hyper-parameters for PPO are kept as the same as those in the original code on [29] [30].

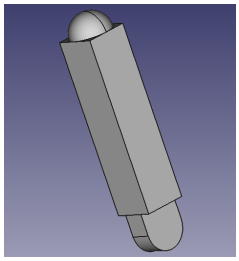
4.2 Simulation environment

4.2.1 Robotic arm

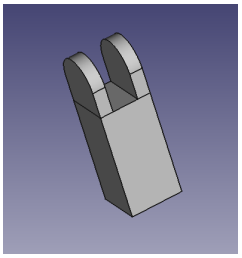
The 3D mesh for the links (figure 4.2) of the robotic arm are built with FREECAD on Ubuntu and the dimensions are chosen to be similar as those of PIRATE to let the simulation close to the real situations.

The robotic arm is described by a URDF (Unified Robot Description Format) file which defines the structure of the robot, the physical properties of the links and joints and also specifies the details of the sensors. Load the file into Gazebo and the model of the robotic arm can be generated automatically as shown in figure 4.3. The prismatic joint connected with the bottom link is placed $3mm$ above the ground so that there is no friction and collision happening between the robot and the floor.

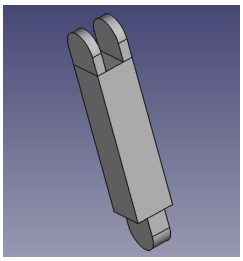
The ranges of the laser are listed in the table 4.1. The horizontal range is chosen to be $180°$ so that the laser can measure the distance from the left, front and right side of the end-effector to the pipe. It should be mentioned that the laser in real life does not always reach the same



(a) Mesh for end-effector link
Size: $20 \times 20 \times 110 mm$

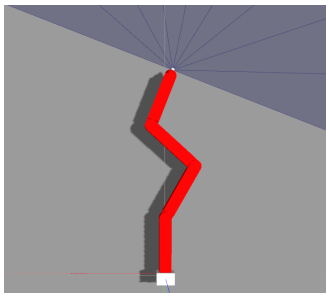


(b) Mesh for bottom link
Size: $20 \times 20 \times 40 mm$

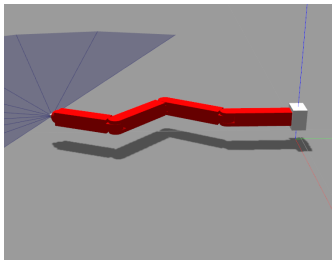


(c) Mesh for other links
Size: $20 \times 20 \times 120 mm$

Figure 4.2: Mesh files for the links of the robotic arm



(a)



(b)

Figure 4.3: Gazebo model of the robotic arm

horizontal range, but 180° is still used as at the beginning phase for the simulation. Later in the experiments, the range will be decreased to a more realistic value.

	range	resolution
horizontal	0 ~ 180°	18°
vertical	5 ~ 500 mm	5 mm

Table 4.1: Range and resolution of the laser scanner on the robotic arm

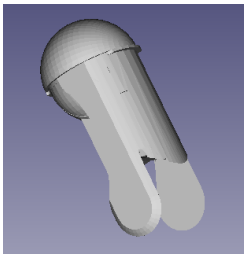
4.2.2 High fidelity model

The robotic arm shown in figure 4.3 will be used in the early experiments to quickly realize any approaches since it has simple meshes and a large laser range. In later experiments, the fidelity of the simulation will be improved by making the running model more realistic; the mesh file for each link is replaced by that from PIRATE and the range of the laser is also decreased because a horizontal range of 180° barely appears on the laser scanner in real life.

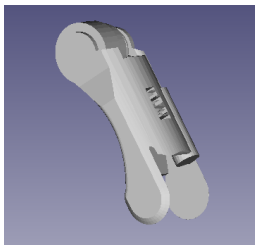
The mesh files for the end-effector link and for the other links are found in the RAM code repository, shown in figure 4.4. Meanwhile, the mesh for the bottom link remain the same as that in figure 4.2. The horizontal range of the laser scanner is then decreased into 86° according to the data from one of the laser in the RAM lab. In the end, the resulting Gazebo model is shown in figure 4.5.

4.2.3 Pipes

The mesh files of the pipes are made by FreeCAD on Ubuntu. A python macro is created on FreeCAD to generate pipes with any specified diameters and turning angles. Some examples are shown in figure 4.6.

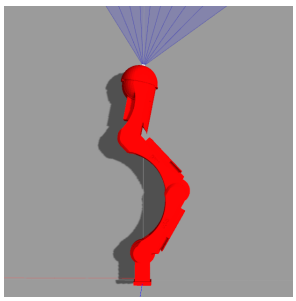


(a) Mesh for end-effector link

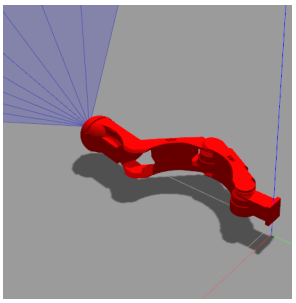


(b) Mesh for other links

Figure 4.4: Mesh files from the PIRATE model

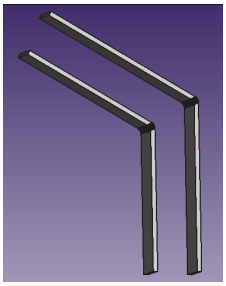


(a)



(b)

Figure 4.5: Gazebo model with a higher fidelity



(a) 120 mm, 120°

(b) 110 mm, 90°

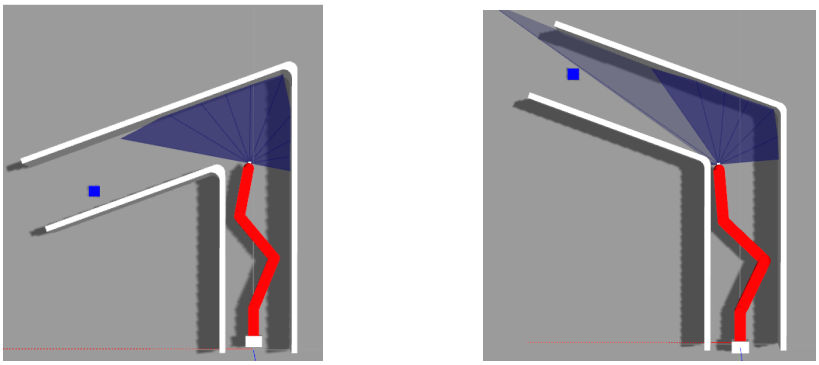
(c) 120 mm, 60°

Figure 4.6: Mesh files of different pipes. The title of each sub-image indicates the diameter and the turning angle of the pipe

It can be seen that each pipe mesh consists of two bending boards instead of a hollow cylinder as a real pipe does. This is designed for the convenience of visualization since if the robotic arm is moving inside a cylinder pipe, it cannot be observed from the outside and makes it difficult for debugging and fine-tuning. Besides, the stricture shown in figure 4.6 actually makes no difference from a hollow cylinder since the robotic arm is planar and can only detect obstacles on a plane.

4.2.4 Integration

At the end, the integration of the robotic arm, the pipe and the target point is shown in figure 4.7. The initial configuration of the robot is set as similar as that of the front part of PIRATE when it starts moving through sharp corners, illustrated in figure 1.2. This Gazebo environment will be used in the early experiments and later the high fidelity model in figure 4.5 will be replace in the later ones.



(a) **(b)**

Figure 4.7: Integration of the robotic arm, the pipe and the target in Gazebo environment (two examples). In later experiments, the robotic arm will be replaced by the high fidelity model shown in figure 4.5

4.3 Real-world environment

Most of the experiments in this thesis are done in simulation. Meanwhile, a real robotic environment is set up for the aim of evaluating the policy obtained from the simulation into the real world circumstances.

A real 4-DOF robotic arm is built based on the prototype from [4] in the RAM lab. As shown in figure 4.8, the previous version of the setup is a planer robotic arm consisting of 3 rotational joints with each actuated by a DYNAMIXEL AX12A servo-motor. Compared to the simulation model used in this thesis, another translation DOF as well as a laser scanner needs to be added into this old setup.

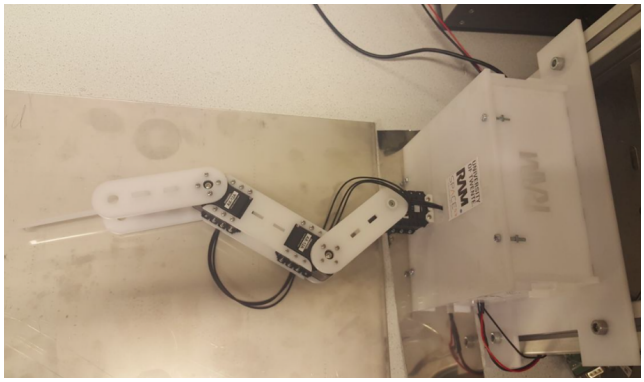


Figure 4.8: The **old** setup of a 3-DOF planar robotic arm in the RAM lab[4]

Figure 4.9 shows the current setup of the robotic arm together pipe-like obstacles around. The translation DOF is realized by installing an extra DYNAMIXEL motor at the bottom of the original setup and attaching it to a linear guide along which the robot can move back and forth; at the same time, a pinion is assembled onto the shaft of the new motor and connected to another linear rack which is placed beside and parallel to the guide. By driving the pinion into rotation, a force will be exerted from the linear rack to the pinion, driving the whole robot to move along the linear guide. In addition, a laser (figure 4.10 (a)) is installed at the end-effector of the robotic arm; it has a horizontal range of 86° and vertical range from $0.1m$ to $1m$. Since the minimum range of the laser is too large to accomplish the requirements for supplementary approach II, an extra distance sensor which can detect the distance in front of it within a range of $0.01m \sim 1.0m$ is also mounted on the laser. The distance sensor is controlled with an Arduino board and the laser is directly connected with the PC via USB.

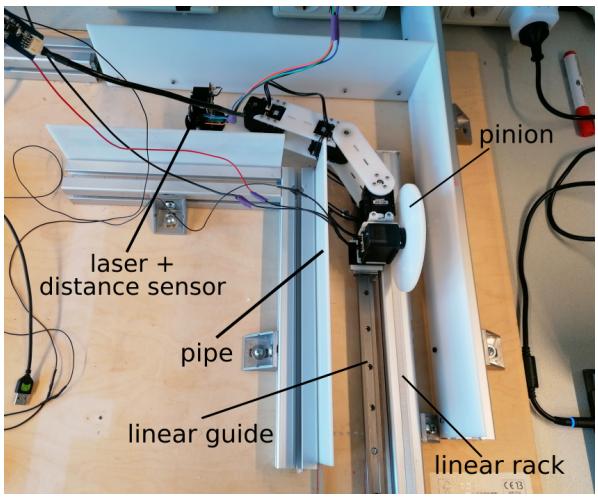
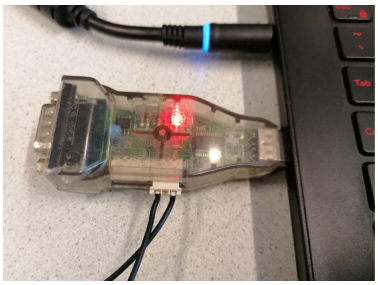


Figure 4.9: The **new** setup of a 4-DOF planar robotic arm with pipe-like obstacles around.

(a)



(b)

Figure 4.10: (a). The laser at the end-effector (b). The device of USB2Dynamixel that connects the motors and the PC

The interface between the motors and the PC is realized through a device called USB2Dynamixel[28], as shown in figure 4.10 (b). USB2Dynamixel provides 3P connectors that connects with the servor-motor at the bottom of the robot and a USB input which goes to the PC and allows the PC to directly send position signals to the motors without any other external control board. The other motors are connected to the bottom one in series through 3P cables which transmit voltage and data signals. The power for the overall setup is supplied by an external power supply.

4.4 Experimental design

4.4.1 Training process

All of the training processes happen inside the simulation, even within the experiment conducted in the real-world environment.

The integration of the Gazebo environment and the RL agent is accomplished with the help of OpenAI/ROS interface. Figure 4.11 shows the details inside one time step of the simulation under the OpenAI/ROS framework, which extends the basic RL structure shown in figure 2.1. It should be mentioned that the aim of pausing Gazebo is to guarantee the data sent to the Deep RL agent is the latest from the simulation.

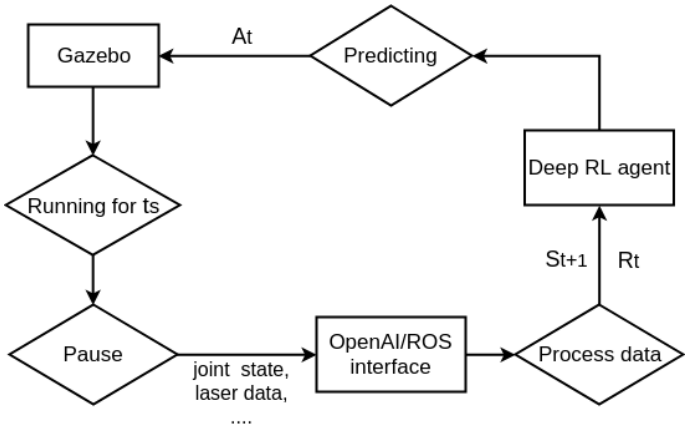


Figure 4.11: Processes within one time step

Next, the processes happening in one episode of the simulation are illustrated in figure 4.12. There are three possible conditions when an episode can terminate: 1. the position of any joint on the robot reaches the pre-defined limit; 2. the total time steps passed by from the beginning of this episode reaches the maximum number; 3. the end-effector of the robot reaches the target. The reward function follows equation 3.12 accordingly.

There is also one detail about the data collection worth mentioning in the training. As illustrated in algorithm 3, the RL agent is updated by using the data from a set of trajectories D_k at iteration k. According to [7][9][23], each collection of the data is only terminated when a maximum value of the accumulative number of time steps is reached; in other words, what matters is not the number of episodes which have been passed, but the total number of time steps spent in these episodes.

Finally, the pseudo-code of the overall training process is shown as follows:

Algorithm 4 Training process

Determine the number of iterations N_{iter}
Determine the maximum of time steps T per episode and the overall time steps T_{iter} per iteration
Determine the weights for the reward function
Initialize the RL agent and simulation environment
for iteration= $1,2,3,...N_{iter}$ **do**
    time_steps_sofar = 0, start data collection
    while time_steps_sofar < T_{iter} **do**
        Execute one episode
        time_steps_sofar += time_steps_this_episode
    Update the RL agent by PPO with the collected data

4.4.2 Experiments to do

In order to approach the objective of this thesis illustrated in section 1.4 — developing a robust path planner that can navigate PIRATE (a 4-DOF robotic arm at current circumstances) to move through sharp corners — several experiments are designed and can be organized into four parts:

1. Torque control experiments
2. Position control experiments

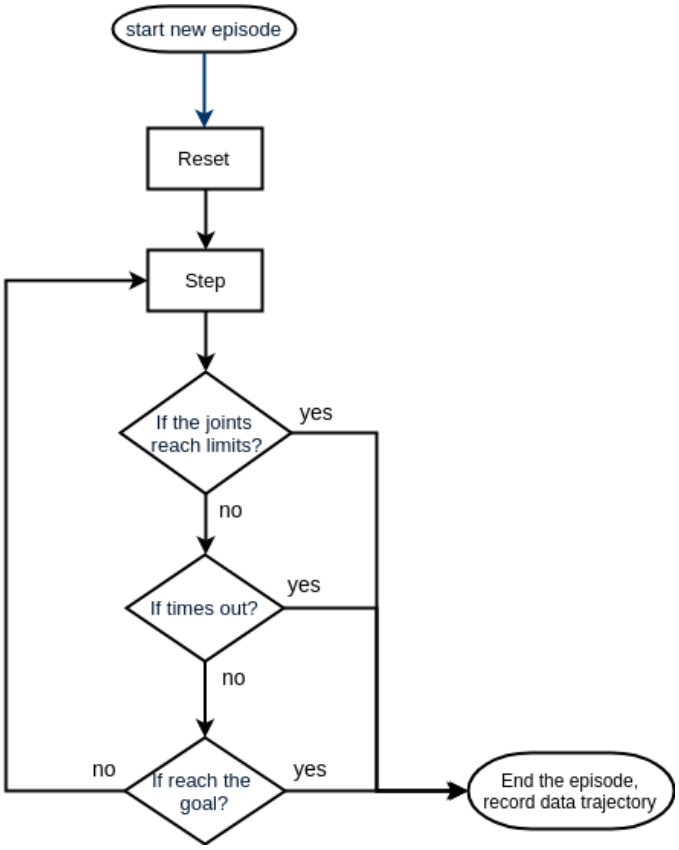


Figure 4.12: Processes within one episode

3. High fidelity experiments
4. Experiments onto the real setup

These four parts are designed in a progressive manner, which means the content and the conclusion of each part is based on those of the previous one, so that the designed path planner can be the most complete and feasible one.

At the same time, there are a few parameters remaining the same throughout all the experiments, which are:

- The number of iterations: $N_{iter} = 150$
- The total time steps per iteration: $T_{iter} = 800$
- The maximum time steps per episode: $T = 100$
- The length of each time step: $t_s = 0.1s$

Therefore, the maximum time that can be spent in one episode is $T \cdot t_s = 10s$ and there are at least $T_{iter}/T = 10$ episodes in each iteration.

In the training phase, the pipe will be randomly selected every iteration among a group of pipes with a fixed diameter $d_{tr} = 120mm$ and turning angle between 60° and 120°, every other 10°. The position of the target point will also be changed randomly within a small area on the other side from the corner.

In the evaluation phase, the generalizability of the resulting agents will be assessed by testing their performance when the robot is put inside pipes which have never appeared during the

training. The diameter of the pipe will be chosen from [90, 100, 110, 120]mm and the turning angle between 55° and 125°, every other 5°.

4.4.2.1 Torque control experiments

It is chosen to explore torque control in the first part so the action output from the RL agent is considered as torque (force) commands, shown in equation 3.3, and will be directly applied on the joints of the robot.

The following experiments will be conducted:

(a) **Explore the influence of the reward for the laser data**

Since the integration of the laser data is something new in learning the target-reaching task compared to [4][15] and [22], the effectiveness of this part of the reward function should be justified. Therefore, C_1 will be fixed as the same in experiment **??** and the only variable in this experiment is the weight C_2 in equation 3.12. At least 4 different values of C_2 will be selected.

The resulting agents will be evaluated and compared.

(b) **Add history data into the state vector**

In this experiment, the influence of adding the data from the previous time step into the state vector will be explored and discussed. Therefore, equation 3.2 for state will be deployed.

First, the basic reward function in equation 3.12 will be used with C_1 remaining fixed and C_2 having the same value which gives the best performance in experiment (a). An agent will be trained under this configuration.

Next, the advanced version of reward function in equation 3.15 will be employed; C_1 and C_2 remain unchanged while C_3 is chosen from a few values to evaluate the influence of the reward on the change of joints position.

The resulting agents will be compared with the best agent obtained from experiment (a).

4.4.2.2 Position control experiments

The latest version of PIRATE is controlled with position controllers, therefore, the performance of RL agent under a position control scheme is investigated in this section. The action output from the agent will be considered as position commands as in equation 3.4, added onto the joint positions of the current time step as in equation 3.5 and sent to the joint controllers (PD controllers at the current circumstance); next, the controllers will convert the position set-points into torque (force) signals and applied onto the joints of the robot.

The designed experiments is as follows:

(c) **A comparison between position control and force control**

The aim of this experiment is to compare the performance of RL agents between position control setup and torque control setup. Therefore, the same experiments as in (b) will be conducted and the resulting agents will be compared with their counterparts under torque control setup.

(d) **Validation of supplementary approach (II)**

In this experiment the feasibility of the approach for navigating the rear part of PIRATE described in section 3.4 will be validated. Since the prerequisite of this approach is a workable RL-based path planner for the front part, so the agent which gives the best performance resulting from experiment (c) will be selected for collecting joint trajectories of

the front part of PIRATE, as shown in figure 3.9. In addition, only those trajectories where the front part successfully moves through the pipe corner should be chosen.

The success rate for the rear part passing the corner will be recorded.

4.4.2.3 High fidelity experiments

In this part of the experiments, the fidelity of the simulation will be improved by replacing the current Gazebo model with the one closer to the condition of PIRATE, as shown in figure 4.5, and decreasing the horizontal range of the laser from 180° to 86°. Since the dimension of the model gets changed, the agents trained from the previous experiments cannot be used anymore. However, the conclusions drawn from the previous experiments can be still used, giving the information on what should be the best RL configuration (state vector, reward function) for the training. Therefore, a new agent in this high fidelity environment will be first trained based on the conclusions so far.

The designed experiments is as follows:

(e) **Validation of supplementary approach (I)**

The approach for choosing a target point inside the pipe described in section 3.5 is designed for realistic situations. In this experiment, suitable values for the parameters in equation 3.18 will be decided so that a maximum success rate of locating a suitable target can be achieved.

(f) **Integration of RL agent with supplementary approach (I) and (II)**

Eventually, the RL agent will work together with supplementary approach I and II such that the robot can determine and reach the target itself, mimicking the front part of PIRATE moving through the pipe corner, then re-spawn the robot and execute supplementary approach I, mimicking the rear part of PIRATE getting out of the corner. In other words, a more autonomous and complete way of moving through the pipe corner for the PIRATE robot is achieved. Especially, the effect of supplementary approach II will be evaluated by running two groups of experiments with or without the approach.

4.4.2.4 Experiments on the real setup

(g) Finally, the real physical setup introduced in section 4.3 will be used for evaluating the complete approach described at experiment (f) in the real world environment, which means the robot will be navigated by the RL agent, the supplementary approach (I) and (II) together to move through pipe-like obstacles.

For the RL agent, a method of sim-to-real model transfer will be used: first, a simulation model which has approximately the same physical properties (dimension, inertia, friction, etc.) as that of the real setup will be built; next, an RL agent will be trained with this model in the simulation; finally, the resulting agent will be directly deployed onto the real setup and work together with the supplementary approach (I) and (II) to navigate the robotic arm.

The differences occurring in the real-world experiment compared to those in the simulation will be observed and analyzed.

5 Results

In this chapter, all the corresponding results are presented. As mentioned in section 4.4.2, each experiment consists of two phases: a training phase and an evaluation phase. In the training phase the RL agents get trained when the robot is moving inside pipes with a diameter of $120mm$ and turning angle of $60° \sim 120°$, every $10°$; while during the evaluation phase, the agents are evaluated when the robot is placed inside pipes with a diameter from $90mm$ to $120mm$ every $10mm$ and turning angle from $55°$ to $125°$ every $5°$.

It can be noticed that there are many different environments in the evaluation phase compared to those deployed in the training. This is done for the aim of testing the generalizability of the resulting agents, in other words, for observing how adaptive they can be when a never-seen environment appears. This aim will be focused throughout the whole chapter.

5.1 Torque control experiments

In this section, the robot is controlled with torque commands and the main goal of the experiments is to explore the influence of different reinforcement learning configurations, such as the state vector, the reward function, on the performance of the resulting agents. The corresponding results are presented as follows.

5.1.1 Experiment (a)

In this experiment, the influence of the reward function on the laser data is verified. As mentioned in the description of the experiment, coefficient C_2 in the reward function 3.12 will be varied while C_1 remain fixed. Hence, the following values are chosen:

$$\begin{aligned} C_1 &= 1.5 \\ C_2 &\in [0.0,\ 0.2,\ 0.4,\ 0.6,\ 0.8] \end{aligned} \tag{5.1}$$

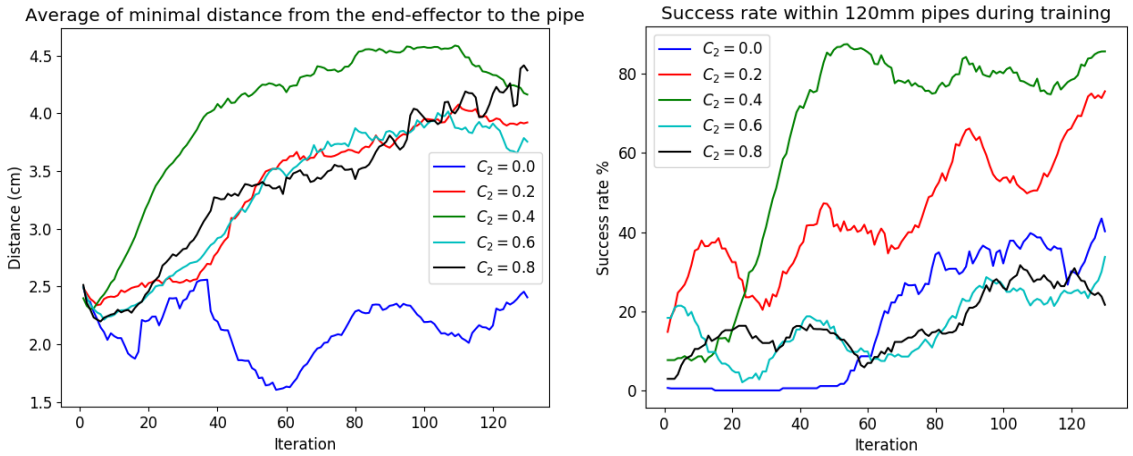
Note the absolute values of the two coefficients do not matter; what plays a difference on the resulting agent is the ratio between different parts of the reward function.

In the training phase, the average distance from the end-effector to the pipe and the success rate of reaching the target in each iteration are recorded, shown in figure 5.1. It should be mentioned that the former one is calculated by averaging the minimal value of the laser data in every time step over the whole iteration; it reflects how far the end-effector is away from the pipe on average.

From the figure 5.1, it can be seen the reward function on the laser data does have an impact on the behaviour of the agent. First, the agent tends to keep away from the pipe when $C_2 \neq 0$, that is to say, the average distance from the end-effector to the pipe is increasing, although different C_2 (except the 0 one) give similar results under a large perspective.

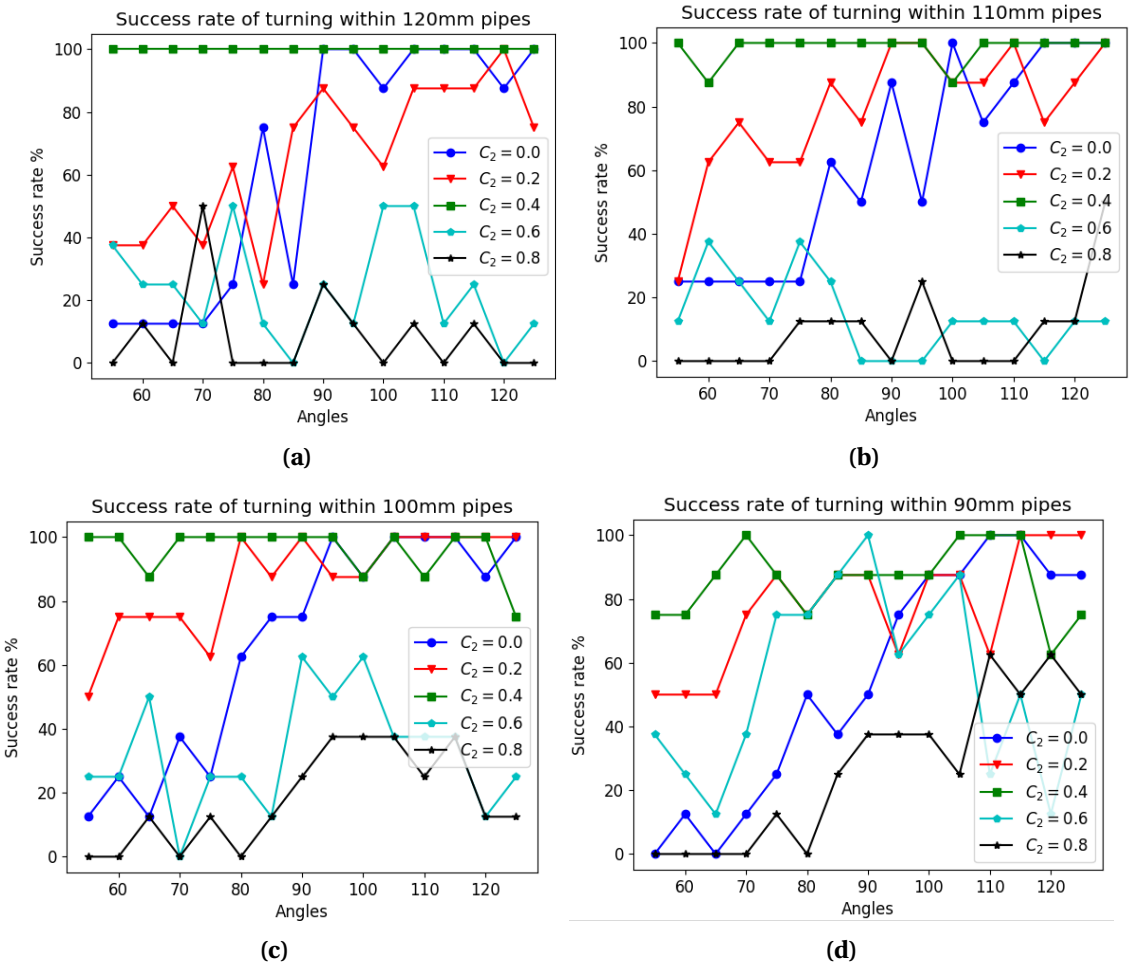
Second, the value of C_2 also influences the success rate of reaching the target. On sub-figure (b), the curves show that as C_2 is increasing from 0.0 to 0.4, the overall success rate also increases; however, when C_2 becomes even larger, the success rate starts decreasing drastically to a result similar with the case when $C_2 = 0.0$. It appears that the best weight on the reward function for the laser data is around 0.4.

Next, **in the evaluation phase** the generalizability of the resulting agents is measured. The success rate for reaching the target is illustrated in figure 5.2 where the x-axis refers to the angle of the pipe corner and the y-axis represents the success rate that the end-effector reaches the target. At the same time, table 5.1 shows the average success rates for each agent inside the pipes with different diameters.



(a) **(b)**

Figure 5.1: Collected data during the training. For both the figures, the x-axis presents to the number of iterations and each curve with a different color refers to the result from an agent with a different value on the coefficient C_2. For the y-axis, in (a) it represents the average distance from the end-effector to the pipe; while in (b), it represents the success rate that the end-effector reaches the target in percentage. Note all the curves have been smoothed by averaging the data over each 20 iterations.



(a) **(b)**

(c) **(d)**

Figure 5.2: The success rate of target-reaching from agents trained with different values of C_2. For each sub-figure, the y-axis indicates the success rate in percentage while the x-axis represents the turning angles of the pipe; the title reveals the diameter of the pipe.

	$120mm$	$110mm$	$100mm$	$90mm$
$C_2 = 0.0$	63%	63%	67%	54%
$C_2 = 0.2$	65%	79%	87%	78%
$C_2 = 0.4$	**100%**	**98%**	**96%**	**86%**
$C_2 = 0.6$	23%	14%	33%	54%
$C_2 = 0.8$	8%	9%	18%	26%

Table 5.1: The success rate of target-reaching for each agent inside pipes with different diameters. Each row corresponds to the success rate of an agent with a specific value of C_2 and each column corresponds to one kind of pipes. This table shows the adaptation of the agent to the change of the diameter of the pipe.

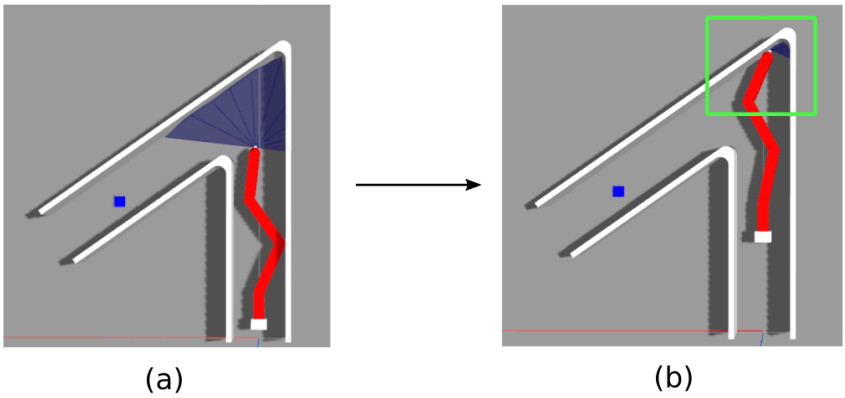


Figure 5.3: The robot gets stuck at the pipe corner with a small, acute turning angle.

As seen on figure 5.2, the agent with $C_2 = 0.4$ (green curve) gives the best generalizability among all, with the success rate dropping down as the diameter of the pipe decreasing. However, even inside pipes with a diameter of $90mm$ this agent still gives a success rate of at least 60%. Meanwhile, the other agents have more difficulty navigating the robot to move through pipes with acute angles than those with obtuse angles, which makes sense since the robot needs to turn its body more and even a little backward in acute pipes so the behaviour is more complicated.

It is noticeable that the agent with $C_2 = 0.8$ (black curve) has the worst performance. According to the visualization of the moving process, this agent appears too sensitive to the distance between the end-effector and the pipe such that it keeps avoiding approaching the pipe. However, it is actually necessary for the robot to get close to the pipe with some extent in order to successfully pass the pipe corner. Hence, C_2 should not be too large.

Another extreme case is the agent with $C_2 = 0.0$ (blue curve), where the environment gives no reaction when the end-effector approaches the pipe during the training. The performance of this agent is significantly worse when the robot is moving inside the pipes with angles smaller than 90°. According to the visualization, it is due to that the end-effector of the robot tends to quickly moves forward and hits the corner of the pipe (the area squared with green in figure 5.3 (b)); then it becomes really hard for the robot to get rid of that area since the corner with an acute angle is blocking its way out. Therefore, the reward function on the laser data does help improve the performance of the agent.

Parts of the observation also get reflected on table 5.1: the average success rate for the agent with $C_2 = 0.4$ is always the highest among the others inside any pipes while that for the agent with $C_2 = 0.8$ the lowest.

However, it is unexpected to notice that for agents with $C_2 = 0.6$ and $C_2 = 0.8$ the average success rate increases when the diameter of the pipes decreases, which was expected to be the

other way around since smaller diameter should give less moving space. According to the visualization from Gazebo, under both agents the robot has a great number of collisions with the pipes but inside one with a large diameter, it gets stuck more easily than in the pipes with a smaller diameter. It is not exactly known why this phenomena happens but note both the two agents fail to learn an optimal policy, so it could happen that the resulting non-optimal policies accidentally fit better in smaller pipes than in larger ones.

5.1.2 Experiment (b)

In this section, the state of the agent and the reward from the environment is changed into the more sophisticated ones as in equation 3.2 and 3.15 to explore the effect of integrating the information from the past.

Firstly, C_3 is chosen to be 0.0 to ignore the effect of the new reward function. As mentioned in the description at section 4.4.2.1, C_1 then remains fixed and C_2 is chosen as the value which gives the best performance in experiment (a), in other words: $C_1 = 1.5$, $C_2 = 0.4$.

However, soon after the training starts, it is observed that the robot quickly develops the “avoiding" behaviour described in experiment (a) for the case when $C_2 = 0.8$, which means **the current value for C_2 is too large under the new state for the agent**; the information from the previous time step lets the agent become more sensitive to the distance between the end-effector and the pipe. Hence, the value for C_2 must be decreased. After trials and errors, $C_2 = 0.1$ appears to be a suitable value where the “avoiding" behaviour of the robot disappears and the resulting agent performs decently on the target-reaching task during the training.

Secondly, another two agents are trained with C_3 chosen as 1.0 and 1.5 respectively in order to introduce the effect of the new reward function; meanwhile, C_1 and C_2 remain as 1.5 and 0.1.

Figure 5.4: Success rate of target-reaching during the training. The blue curve represents the results from the best agent in experiment (a) where the basic state vector is used and $C_2 = 0.4$; the other three agents are all trained under $C_2 = 0.1$ and the state vector with history information. **All the results are under torque control**

The results from the training phase are shown in figure 5.4. The agent which gives the best performance from experiment (a) is used as a baseline for comparison. It can be seen that the success rate of all the agents converges to a similar level, but the ones employed state history reach the convergence at a later moment. It should be mentioned that there is an unusual “peak" between 20 to 50 iterations on the light blue curve(the result from the agent when $C_3 = 1.5$); the success rate first raises to around 60% then quickly drops down to 20%. This can be explained by the facts that the robot learns how to reach the target faster in the pipes with obtuse angles than those with acute angles and the angle of the pipe in each iteration is randomly selected. What possibly happened around that peak could be that in the iterations

before that peak, most of the pipes got chosen have a obtuse angle and the robot quickly learns how to reach the target, so the curve raises; but after that peak and before the success rate increases again, most the pipes got chosen have an acute angle and the robot has not learned how to reach the target in these pipes yet, so the success rate drops. Therefore, this “peak" is not a common phenomenon and it could only happen at the early stage of the training.

Next, the generalizability of the resulting agents is assessed **in the evaluation phase** and the results are shown in figure 5.5 and table 5.2. As seen, the three agents with state history give more or less the same performance inside $120mm$ pipes, but as the diameter of the pipes decreases the performance of the agents starts dropping, with the one with $C_3 = 0.0$ the most and the one with $C_3 = 1.5$ the least. Especially, the success rate of the agent with $C_3 = 0.0$ (red curve) is significantly lower than the others inside the pipes with $100mm$, $90mm$ diameter and acute turning angles.

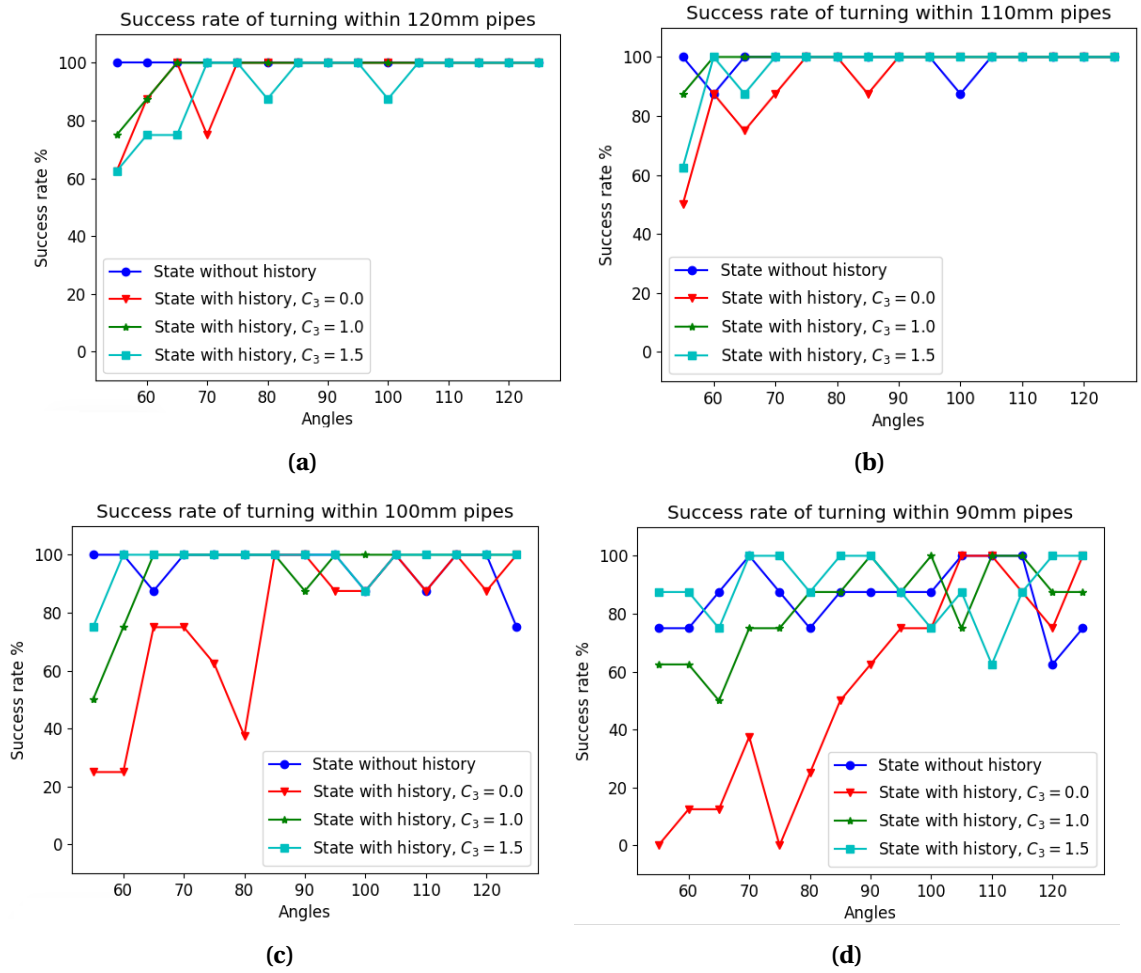


Figure 5.5: A comparison of the generalizability of the agents when the state history and the advanced version of reward function are introduced into the training. For each sub-figure, the y-axis indicates the success rate in percentage while the x-axis represents the turning angles of the pipe; the title reveals the diameter of the pipe. All the results are obtained under **torque control**.

This observation refers that the advanced reward function in equation 3.15, which includes the penalty on the change of the joint positions as in equation 3.14, improves the generalizability of the agent when the state history is included. A possible explanation for this improvement is that this extra reward help truncate those joint configurations where the robot gets stuck in the pipe, since at these moments the change of the joint positions is extremely small, resulting a very low reward.

	$120mm$	$110mm$	$100mm$	$90mm$
State without history	100%	98%	96%	86%
State with history, $C_3 = 0.0$	95%	93%	77%	54%
State with history, $C_3 = 1.0$	97%	99%	94%	82%
State with history, $C_3 = 1.5$	92%	97%	**98%**	**89%**

Table 5.2: The success rate of target-reaching for each agent inside pipes with different diameters. The first row corresponds to the results from the agent which gave the best performance from the previous experiment, while the other three rows show the success rate from the agents incorporated with the advanced state and reward function. This table shows the adaptation of the agent to the change of the diameter of the pipe. All the results are obtained under **torque control**.

On the other hand, the agent which employs the basic state function and a suitably selected value for C_2 (in this case, 0.4) gives a similar generalizability with the agent which incorporates the state history and a suitable C_3 (in this case, 1.5), with the former one giving a faster convergence during the training but the latter one a better generalizability inside $100mm$, $90mm$ pipes according to table 5.2. It is still possible that if C_3 gets increased even more, the performance of the resulting agent can be accordingly improved. However, due to time limits issue, this part of the experiments will not be explored further.

5.2 Position control experiments

Recall that during the position control, the set-points for the controllers of the joints are calculated with equation 3.5 as below:

$$q_{goal} = q + \alpha \cdot a_{pos}$$

The value α is better to be small because at the beginning of the training phase, the RL agent has not reached convergence and will output random signals. If α is not small enough, the joints position of the robot will be drastically changed every time step and the robot will have a great number of contacts with the pipes, making the simulation become unstable. By first doing several trials, α is chosen to be 0.2 such that a stable simulation can be guaranteed while the robot does not move too slowly.

5.2.1 Experiment (c)

In this experiment, the robot is controlled with position controllers and the resulting RL agents are compared with their counterparts under torque control. Four agents are trained with the same configurations on the state and the reward function as those of the agents in experiment (b) to form a better comparison.

The first agent is trained with the basic state 3.1 and reward function 3.12. C_1 and C_2 are chosen as 1.5 and 0.4 accordingly. **The training and evaluation results** are then compared with their counterparts under torque control, as shown in figure 5.6 and 5.7.

As seen on the training curve, the agent under position control converges slightly faster but gives a worse final performance than the one under torque control. In the evaluation phase, the generalizability of the position control agent is worse in general than that of the torque control agent.

One of the possible reasons that causes this difference is that the joints of the robot are more “stiff" under the control of position controllers (PD controllers in this case) than those under torque control. When the robot is having contacts with the pipe, it is receiving reaction force from the environment and the position of the joints will be perturbed consequently. The torque (force) being applied on the joints from the agent will not be changed under torque control scheme because of this perturbation since the agent is directly sending torque commands

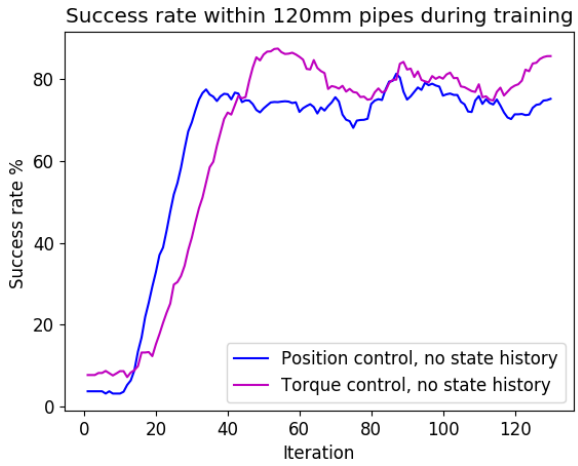


Figure 5.6: Success rate of target-reaching during training. The training process and RL parameters for the two agents are exactly the same.

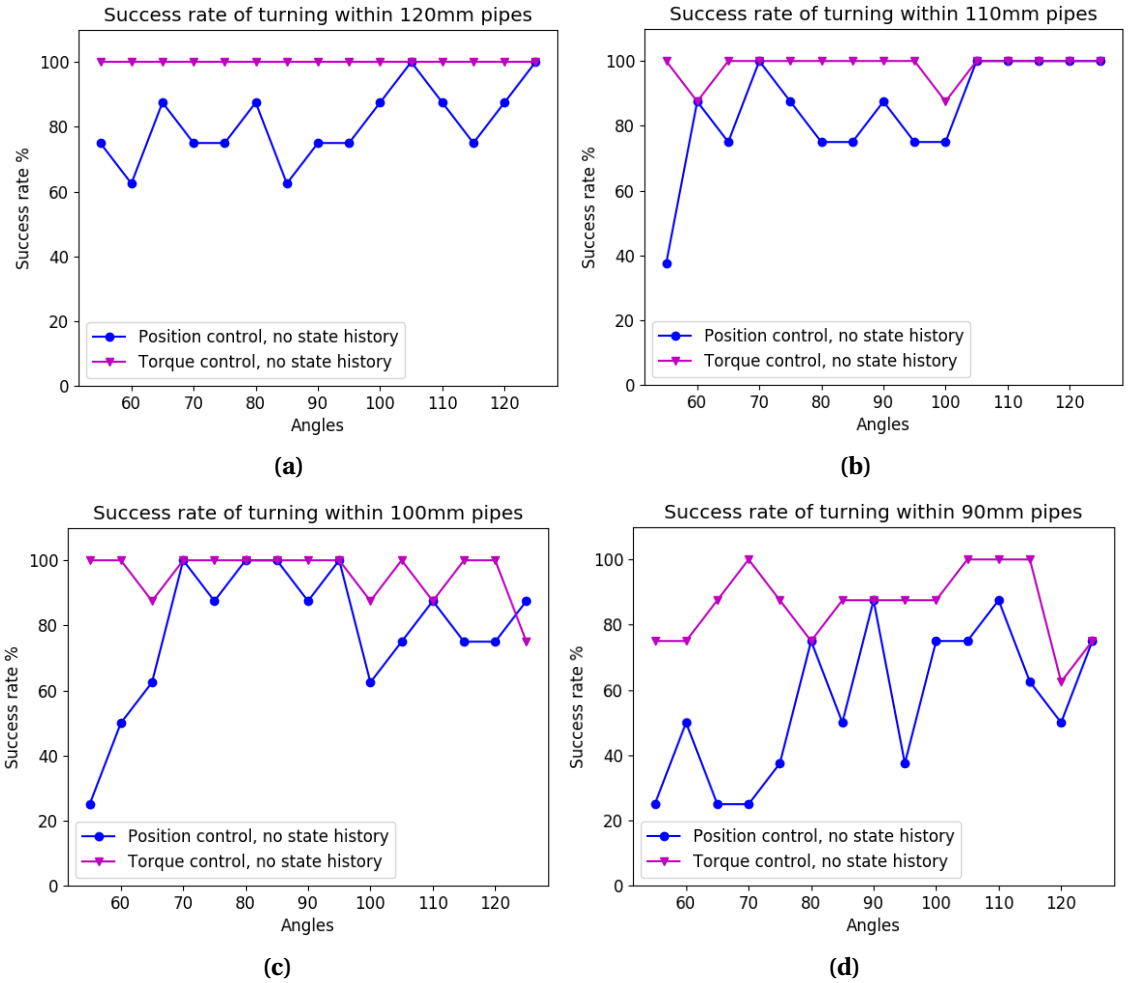


Figure 5.7: A comparison between the generalizability of the position control agent and torque control agent. Both the agents are trained under the basic state and reward function.

to the robot; however, under position control scheme the controllers will increase the torque commands it sends to the joints whose positions are deviating the set-points.

Therefore, the robot is easier to be stuck in the pipe under position control scheme when contacts happen and it leads to the decreasing of the success rate for the target-reaching which get reflected in both the training phase and the evaluation phase.

Next, another three agents are trained with the advanced state vector and reward function. The coefficients for the reward function is chosen as $C_1 = 1.5$, $C_2 = 0.1$, $C_3 = 0.0, 1.0, 1.5$ just as what have done in experiment (b), so that the effect of this different RL configuration can be evaluated under position control scheme. **The results from the training phase** are shown in figure 5.8; to make a better comparison, the agent which does not include the history information is used as a baseline.

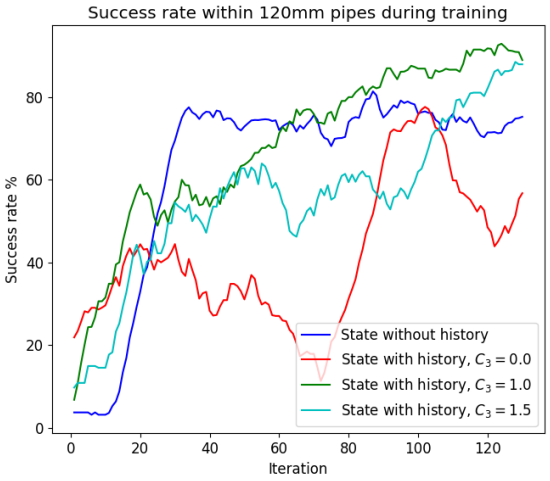


Figure 5.8: Success rate of target-reaching during the training. All the results are under **position control**

From the training results it can be observed that the agents incorporating with state history and non-zero C_3 (green and light blue) converges slower but reaches a better final performance than the one without (blue). In addition, it is noticeable that there is an unusual drop between the 20th iteration and the 70th iteration of the curve for the agent with state history but a zero C_3 (red), and right after this drop a drastic increasing of the success rate appears but then a decreasing comes back again. This bizarre behaviour the training curve reflects that the learning of the agent is very unstable and probably fails to converge to the optimum.

	$120mm$	$110mm$	$100mm$	$90mm$
State without history	81%	85%	78%	56%
State with history, $C_3 = 0.0$	85%	71%	59%	42%
State with history, $C_3 = 1.0$	80%	88%	92%	78%
State with history, $C_3 = 1.5$	**99%**	**98%**	**98%**	**92%**

Table 5.3: The success rate of target-reaching for each agent inside pipes with different diameters. The first row corresponds to the results from the agent which does not include state history in the training, while the other three rows show the success rate from the agents incorporated with the advanced versions of the state and reward function. This table shows the adaptation of the agent to the change of the diameter of the pipe. All the results are obtained under **position control**.

The observations obtained from the training phase also gets reflected on **the evaluation phase**. As shown in figure 5.9 and table 5.3, the agent trained with $C_3 = 1.5$ (light blue) surpasses all the other agents inside the pipes with every chosen diameter, although the success rate still keep decreasing when the diameter becomes smaller. Meanwhile, the agent incorporating with state history and zero C_3 (red) also gives the worst generalizability among all which means it did not find an optimal policy.

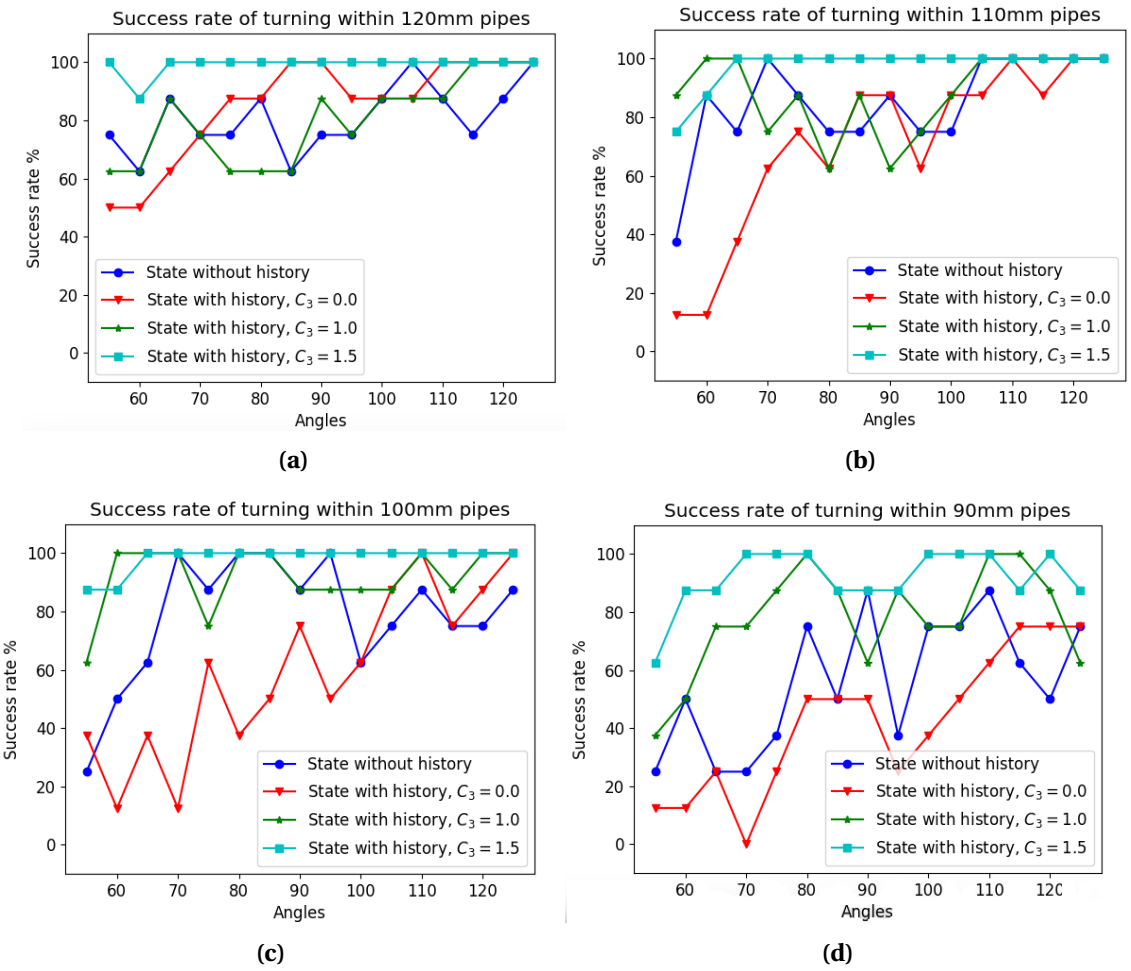


Figure 5.9: A comparison of the generalizability of the agents when the state history and the advanced version of reward function are introduced into the training. All the results are obtained under **position control**.

Similarly as in torque control, the possible reason that the agent trained under $C_3 = 1.5$ gives the best performance is that the reward function on the change of joints position adds extra penalty on those configurations where the robot is stuck in the pipe. As a result, the visiting rate to those joint configurations becomes smaller, leading to better trajectories for the robot. Therefore, including state history together with a reward function on the change of joints position helps improve the performance of the agent when the robot is controlled with position controllers.

Finally, a comparison is made between the two agents which are both trained with $C_3 = 1.5$ but one under torque control and the other under position control . **Both of their training and evaluation results** are compared and shown in figure 5.10 and 5.11. It can be seen that the two agents have similar learning curves and generalizability.

The original motivation of employing this sophisticated RL configuration where the state history and the extra reward function on the change of the joints position are integrated is to let the agent be able to detect whether the robot gets stuck in the pipe and also learn how to free itself after being stuck. However, according to the observations from Gazebo, both the agents in figure 5.10 fail to learn these expected behaviours, this is why their performance still drops in $90mm$ pipes since the robot gets stuck more easily.

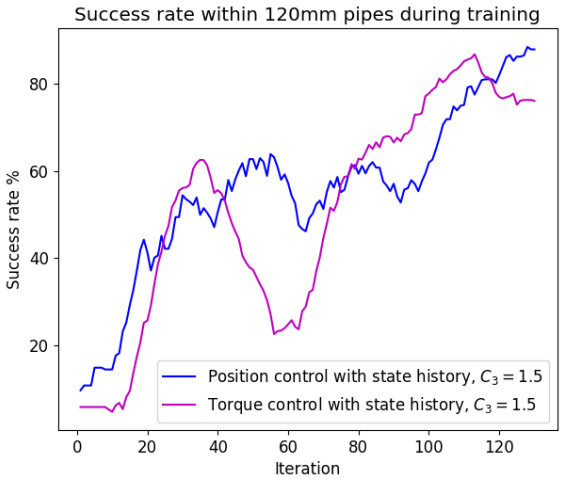


Figure 5.10: Success rate of target-reaching during the training. Both agents are trained under the same RL configurations.

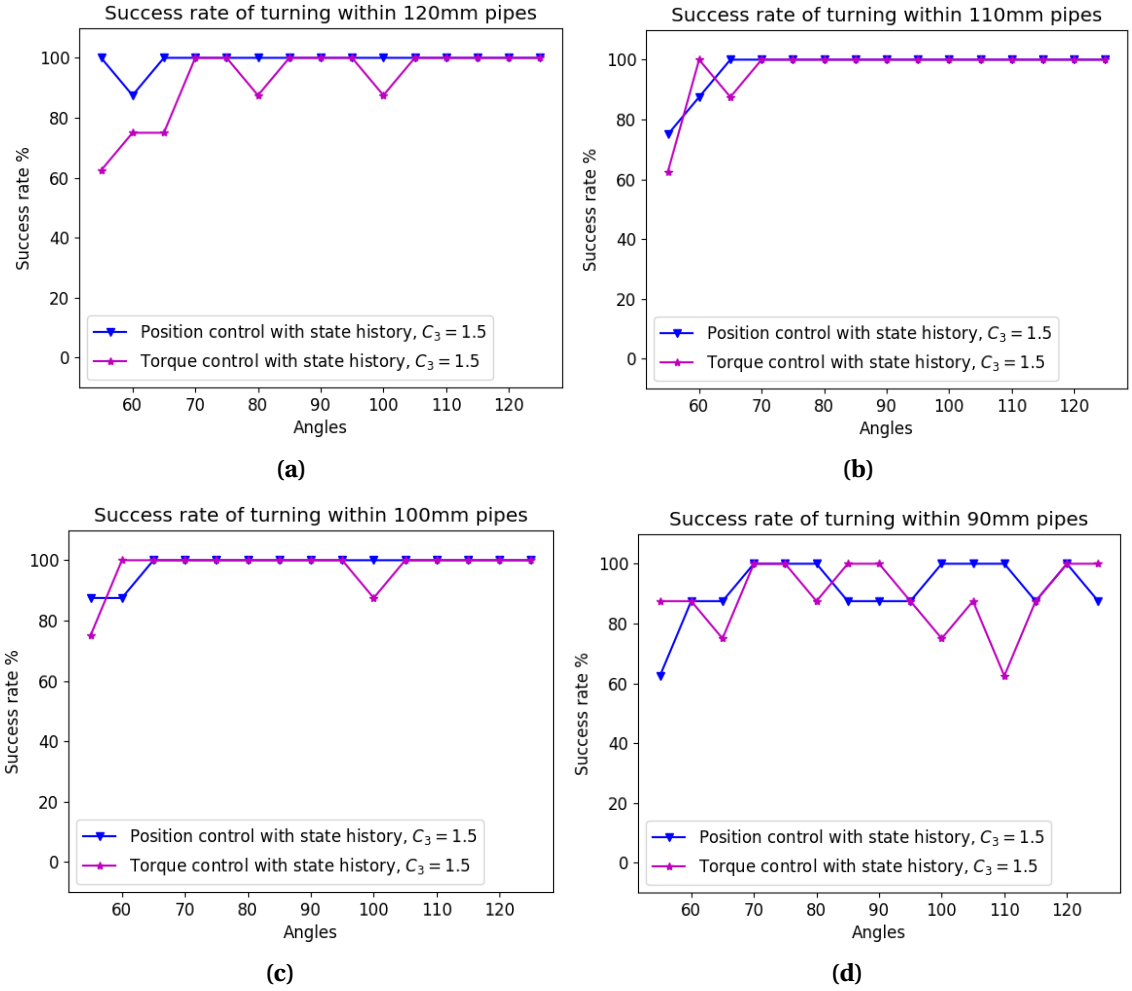


Figure 5.11: A comparison of the generalizability of the agents under torque control and position control. Both agents are trained under the same RL configurations.

5.2.2 Experiment (d)

In this experiment, the supplementary approach (II) described in section 3.5 is evaluated under position control. The approach uses the trajectories collected during the motion of the front

	$120mm$	$110mm$	$100mm$	$90mm$
Position control with state history, $C_3 = 1.5$	99%	98%	98%	92%
Torque control with state history, $C_3 = 1.5$	93%	97%	98%	89%

Table 5.4: The success rate of target-reaching for each agent inside pipes with different diameters. Both the agents have the same RL configuration but one under position control and the other under torque control.

part of the PIRATE through the corner from the agent trained with state history and with state history, the advanced reward function and $C_3 = 1.5$; since this agent gives the best performance under position control scheme.

Assume the moving process for the front part of PIRATE is the first phase and the process for the rear part is the second phase. One episode for the evaluation in this experiment is conducted as the following steps:

1. Start the first phase of the movement.
2. If the robot reaches the target, re-spawn it at the initial position for the second phase of the movement; If not, end the episode.
3. Start the second phase (if the episode is not over); end the episode until either the robot moves from the pipe corner completely or the time is out.

Figure 5.12 shows the initial position and the ending position which indicates the robot passes the pipe corner completely in the second phase of the movement.

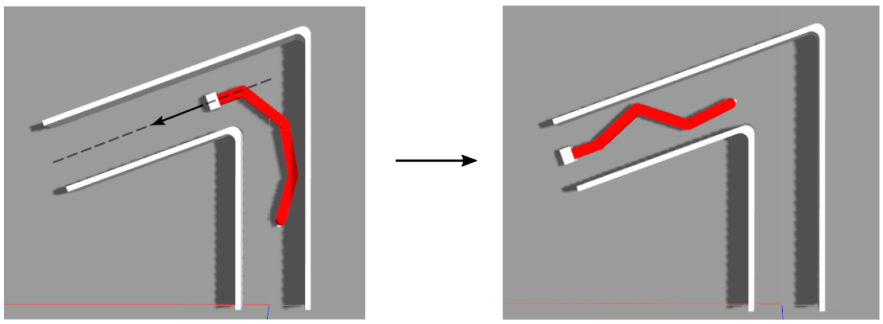
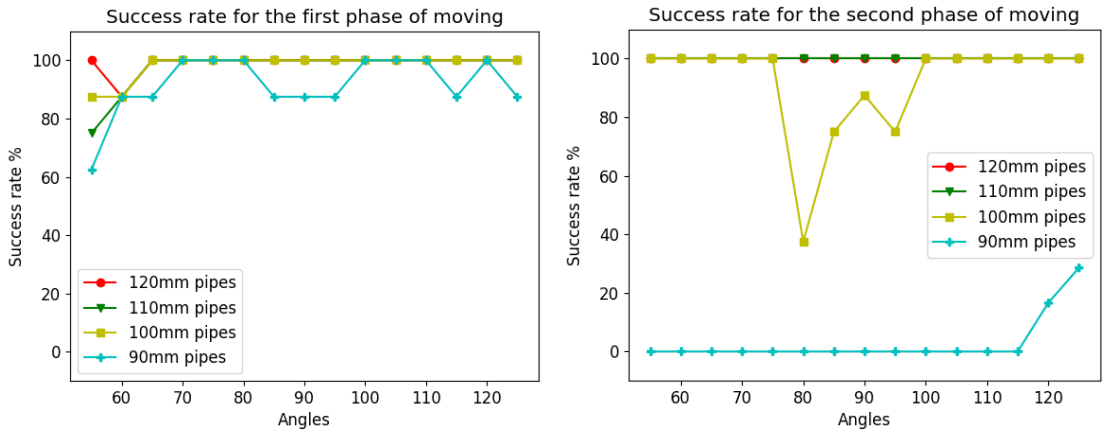


Figure 5.12: (a) The robot is re-spawned at an initial position which represents the starting point where the rear part of PIRATE just unclamps and begins to move. The white block moves along the dash line to mimic the front part of PIRATE pulling the rear one. (b) The moment when the robot is considered to move out of the pipe corner completely

The pipes used for the validation remain the same as those in the previous experiments—with the diameter chosen from [90, 100, 110, 120]mm and the turning angle between 55° and 125°, every other 5°. The results are shown in figure 5.13, where the success rate of the first phase and the second phase inside different pipes are both presented. It can be seen that the robot manages to move through the pipe corner with a success rate of 100% inside $120mm$ and $110mm$ pipes. However, note that the second phase can only be triggered if the robot reaches the target in the first phase, hence a 100% success rate should indicate that if the front part of PIRATE can move through the pipe corner, the rear part can do so as well.

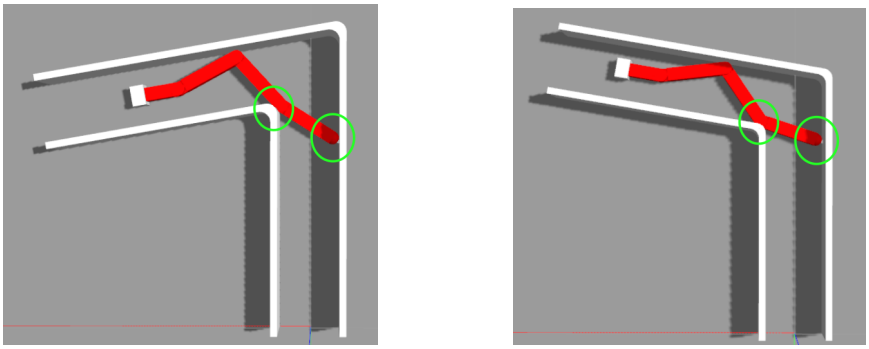
On the other hand, the second phase barely succeeds inside $90mm$ pipes. According to the observations from Gazebo, it is because the end-effector link gets stuck while passing the corner when the diameter of the pipe is too small. Figure 5.14 shows two examples of these moments:



(a) **(b)**

Figure 5.13: (a) The success rate of target-reaching when the front part of PIRATE is moving through pipe corners; (b) The success rate for the rear part to move out of the corner.

the end-effector is stuck at the areas circled with green because it is too long to rotate further to the other side; the robot stops moving anymore.



(a) **(b)**

Figure 5.14: (a) The robot stops moving inside a pipe with $90mm$ diameter and 80° turning angle. (b) the robot is stuck inside a pipe with $90mm$ diameter and 100° turning angle. Both moments happened because the end-effector gets stuck at the green-circled areas.

5.3 High fidelity experiments

In this experiment, the fidelity of the simulation is improved by making the following three changes:

1. Replace the meshes of the links on the robotic arm by those from the PIRATE model.
2. Decrease the horizontal range of the laser scanner in simulation from 180° to 86°. (86° is the range of one of the running lasers in the RAM lab)
3. Integrate the supplementary approach (I) (section 3.4) so that the robot can determine a target point by itself.

for the aim of making the situations in the simulation more realistic and closer to those for PIRATE. Meanwhile, the robot is still controlled with position controllers.

To conduct the experiments described in section 4.4.2.3, an agent is first trained with the best RL configurations explored in the position control experiments, which means the state vec-

tor includes the history information and the reward function incorporates the penalty on the change of joints position with C_3 chosen as 1.5.

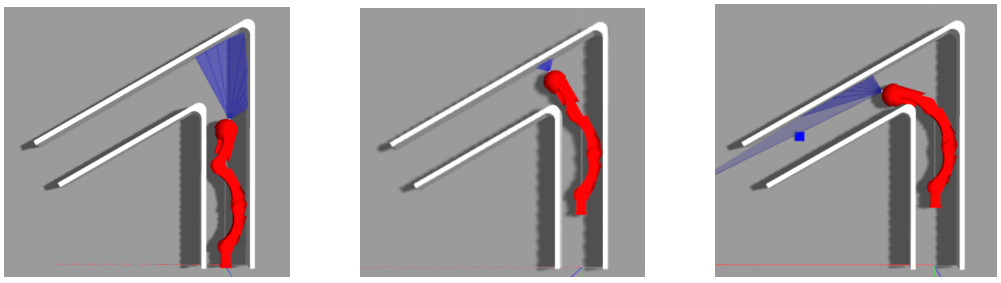
5.3.1 Experiment (e)

In this experiment, the suitable values for the parameters in equation 3.17 and 3.18 described in the supplementary approach (I) is determined.

The rules for finding these values have been briefly introduced in section 3.4 and more specific tuning is done by executing the approach in the simulation and adjusting the values so that the robot can successfully locate target point inside different pipes. After several trials and errors, the following values for the parameters in equation 3.18 and 3.19 are decided:

$$\begin{gathered} \omega_0 = 0.15 \text{ / time step} \\ v_0 = 0.02m \text{ / time step} \\ N = 8,\ \delta = 0.05m,\ L = 0.2m \\ L_o = 0.3m \end{gathered} \tag{5.2}$$

Note the length of one time step is $t_s = 0.1s$ as mentioned before. With these parameters, the robot can determine a suitable target point inside the pipes with a success rate of **100%**, which means the robot never gets stuck in the pipe during this process and the target always locates at the other side of the pipe with respect to the corner. An example where the robot successfully finds a target point inside a pipe with $90mm$ diameter and 60° turning angle is illustrated in figure 5.15.



(a) Start at the Initial position

(b) Move forward and turn the end-effector to the left

(c) Find the target point

Figure 5.15: The sequence of the robot locating a target point inside a pipe with a diameter of $90mm$ and turning angle of 60°.

5.3.2 Experiment (f)

In this experiment, the robot is controlled by the RL agent together with the supplementary approach (I) and (II). The whole process for the PIRATE to move through pipe corners is then simulated: first, the front part locates a suitable target point with the laser ; next, trigger the RL agent to let the front part reach the target and pass the pipe corner; finally, navigate the rear part to move through the pipe corner.

For a better observation on the effect of the supplementary approach (I), two separate sub-experiments are conducted where in the first one the target point is given in advance by pre-computing its location while in the second sub-experiment the target is located by the robot by using supplementary approach (I). The flow charts for one episode in the two sub-experiments are shown in figure 5.16.

The results from the first sub-experiment are presented in figure 5.17 and table 5.5. It can be seen that the success rates for both phases of the moving process significantly drops compared

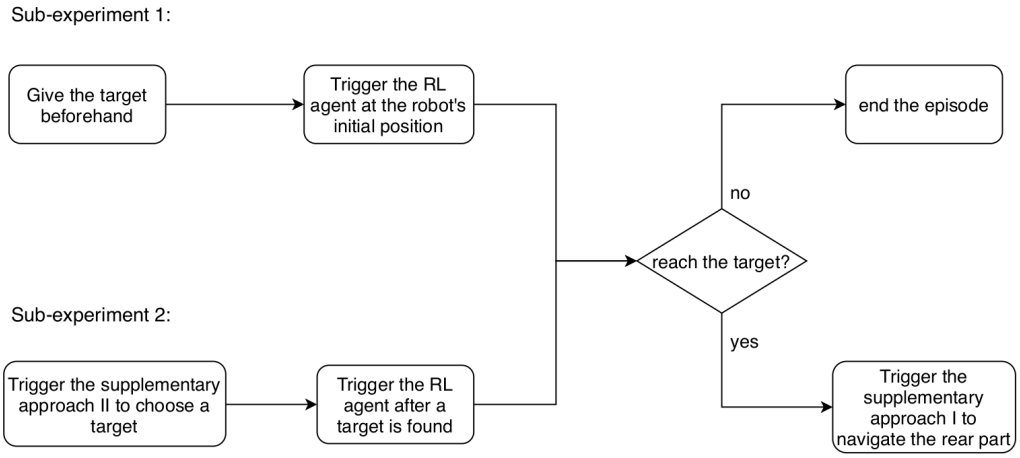
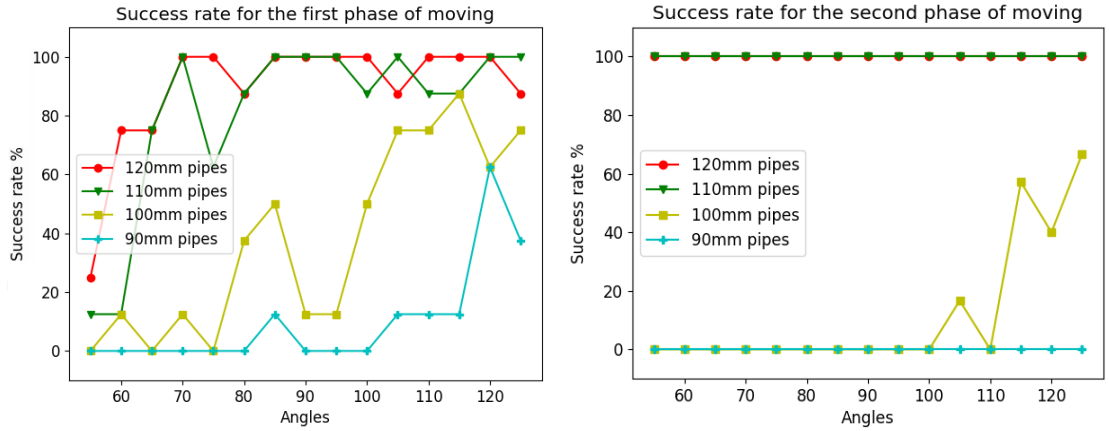


Figure 5.16: Flow chart for one episode in the two sub-experiments where in the first one the RL agent is triggered at the beginning and the target is given beforehand while in the second one the supplementary approach (I) is triggered in advance to choose a suitable target.



(a) **(b)**

Figure 5.17: The success rate of moving through the pipe corners for the front part and the rear part of PIRATE. The data for both plots are obtained **without supplementary approach (I)**, which means the target point in the first phase of moving is given beforehand.

to their counterparts shown in figure 5.13 when the mesh files from the PIRATE have not substituted the original robotic arm, especially in $100mm$ and $90mm$ pipes. One of the possible reasons for this decreasing in success rate is that the meshes from the links of PIRATE are wider and also have more complicated structures than those on the robotic arm, which makes the robot easier to be stuck in the pipe when the diameter becomes smaller. In addition, it can be also due to the change of the horizontal range of the laser: the range is decreased from 180° to 86° so the area that the robot can detect in each time step becomes smaller; at the same time, the weight C_2 for the reward on the laser data remains as the same value as it was in experiment (d), which may not be a good selection for the current situation any more.

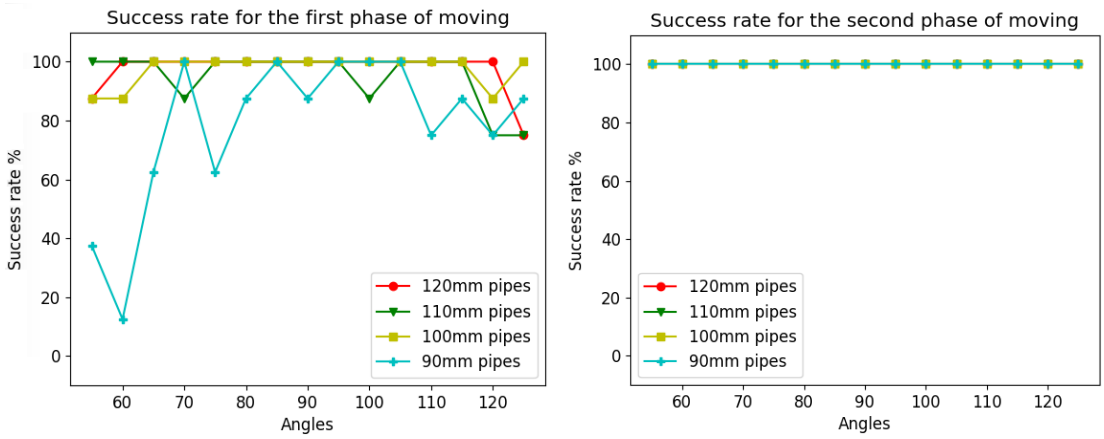
Furthermore, on sub-figure (b) in figure 5.17 the success rates for the second phase of the moving process inside $120mm$ and $110mm$ pipes reach 100%, meaning as long as the front part of the PIRATE moves though the corner, the rear part can also do so. However, the success rate significantly decreases inside $100mm$ and $90mm$ pipes with a larger degree compared to its counterpart in figure 5.13. From the observations from Gazebo, it is due to the same reason as illustrated in figure 5.14: the end-effector link is stuck when it is passing the corner; and since

the dimension of the link becomes larger after its mesh is changed, the success rate drops even more.

	$120mm$	$110mm$	$100mm$	$90mm$
First phase	89%	81%	36%	10%
Second phase	100%	100%	12%	0%

Table 5.5: The average success rate for the first phase of the moving process **without supplementary approach (I)**, which means the target point is given before the moving starts.

Next, the results of the second sub-experiment in this section where the supplementary approach (I) is integrated are presented in figure 5.18 and table 5.6 below. The robot in this sub-experiment needs to determine a target point itself with the help of the laser, making the whole process more realistic and reasonable.



(a) **(b)**

Figure 5.18: The success rate of moving through the pipe corners for the front part and the rear part of PIRATE. The data for both plots are obtained **with supplementary approach (I)**, which means the target point in the first phase of moving is determined by the robot itself.

	$120mm$	$110mm$	$100mm$	$90mm$
First phase	**98%**	**95%**	**98%**	**78%**
Second phase	100%	100%	**100%**	**100%**

Table 5.6: The average success rate for the second phase of the moving process **with supplementary approach (I)**, which means robot needs to locate a suitable target itself before the RL agent is triggered.

Surprisingly, the results show that integrating this target-finding approach considerably improves the success rate for both the first and the second phase of the moving process inside all types of pipes. This can be explained from the observation of the simulation: with supplementary approach (I) the overall trajectory of the robot is changed and the RL agent is triggered at different start points, which reduces the possibility for the robot of being stuck inside the pipes.

Figure 5.19 and figure 5.20 show the sequence of movement for the front part of PIRATE inside a pipe which has a diameter of $90mm$ and an angle of $70°$ without and with the supplementary approach (I) respectively. In figure 5.19, the RL agent is triggered at the initial position of the robot, then the robot gets stuck at the corner of the pipe because the moment when it starts turning left is too late. Note this kind of behaviour can lead the robot to reach the target in $120mm$ and $110mm$ pipes which have more space at the corner. On the other hand, in figure 5.20 the robot manages to pass the pipe corner because it starts turning the end-effector to the left at the initial position shown in (a) and has already “faced" to the other side of the pipe

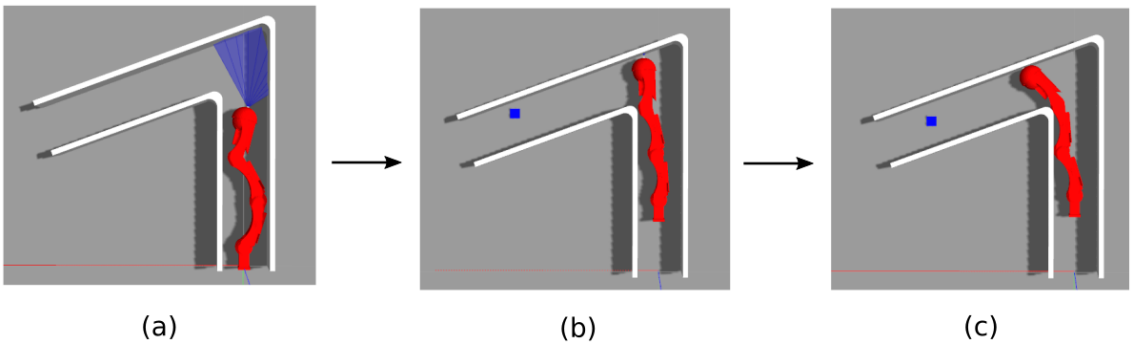


Figure 5.19: The sequence of movement for the front part of PIRATE in a 90mm pipe **without supplementary approach (I)**. The RL agent is triggered in sub-figure (a) and the moment when the end-effector link starts turning (sub-figure (b)) is too late such that the robot gets stuck and fails to pass the corner.

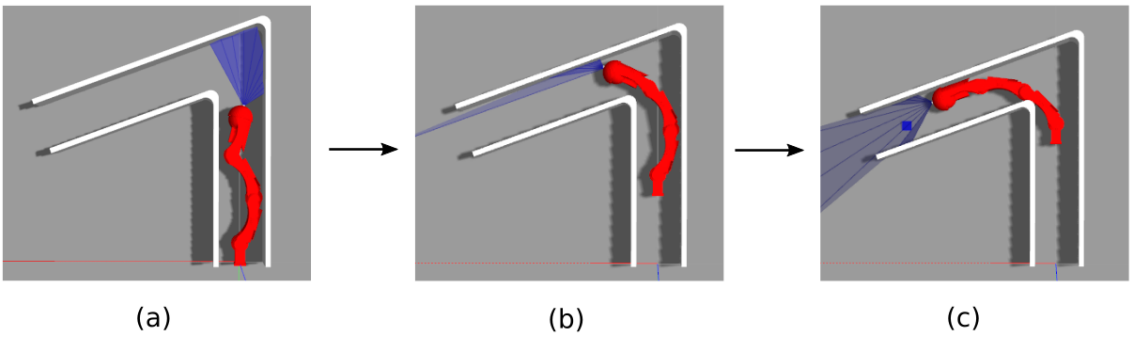


Figure 5.20: The sequence of movement for the front part of PIRATE in a 90mm pipe **with supplementary approach (I)**. Since the robot first needs to turn the end-effector link to the left to locate a target point, when the RL agent is triggered, the robot has already been at a good starting point (sub-figure (b)) from which the target is easier to be reached.

before the RL agent is triggered (sub-figure (b)). This kind of behaviour brings two benefits: first, the end-effector will not get stuck at the corner as in figure 5.19; second, when a target point is determined and the RL agent is triggered, the robot has already been at a configuration which makes it easier to reach the target point. As a result, the supplementary approach (I) increases the success rate for the first phase of the moving process.

Furthermore, it can be seen in figure 5.18 (b) that the success rate for the second phase reaches 100% not only in 120mm and 110mm pipes, but also 100mm and 90mm pipes. This is not a trivial improvement and is due to that there are fewer collisions happening between the robot and the pipe during the first phase of the movement; and since the trajectories for the second phase is determined by those from the first one, fewer collisions happen in the second phase as well. As a result, the success rate also increases.

5.4 Experiments onto the real setup

5.4.1 Experiment (g)

In this experiment, the proposed RL-based navigation approach together with the supplementary approach (I) and (II) are evaluated in the real robotic environment introduced in section 4.3.

The experiment is also divided into two phases as before: a training phase and an evaluation phase. During the training phase, a simulation model which has approximately the same physical properties (dimension, inertia, friction, etc.) as that of the real setup will be built and an RL agent will be trained with this model in the simulation; while in the evaluation phase the

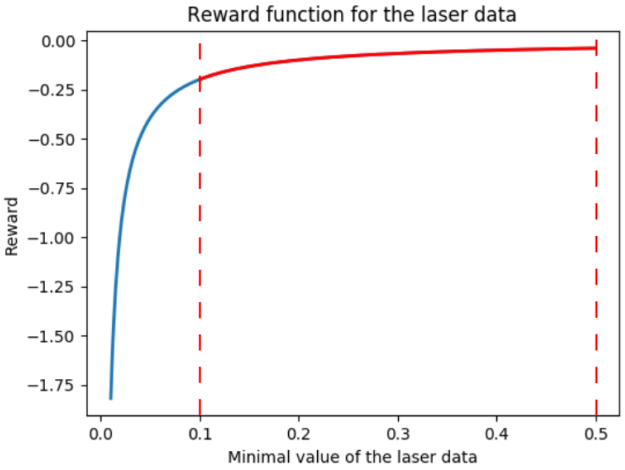


Figure 5.21: The plot of the reward function for the laser data. It can be seen that the function value does not change much when the minimal laser data is between $0.1m$ and $0.5m$ (red part).

resulting agent will be directly deployed onto the real setup and work together with the two supplementary approaches to navigate the robotic arm to move with the presence of pipe-like obstacles.

Training phase in simulation

Before the training of the RL agent, another Gazebo model is built such that it has the same physical characteristics as the real 4-DOF robotic arm. Specifically, the links of the new model are relatively wider than those of the previous ones; the laser scanner has a horizontal range of 86° and a vertical range of $0.1m \sim 0.5m$. It should be mentioned that on the high fidelity Gazebo model deployed in experiment (e) and (f), the minimal range of the laser is $0.01m$. Meanwhile, the pipes used for the training remain the same—with a diameter of $120mm$ and turning angle from 60° to 120°, every 10°. In addition, it is chosen to employ position control scheme since the real robotic arm is controlled with position commands.

The RL configuration is first chosen as the same in the high fidelity experiments, which means the state vector should include the history information and the reward function should incorporate the penalty on the change of joints position; meanwhile, $C_1 = 1.5, C_2 = 0.1, C_3 = 1.5$.

However, during the training it is quickly found that the agent fails to navigate the robotic arm to pass the corner which has an acute turning angle. This observation is actually not very unexpected: considering the reward function on the laser data expressed in equation 3.10:

$$r_{LD} = -\frac{1.0}{\beta * \min(D) + d_0}$$

where $\beta = 5.0$, $d_0 = 0.005$, the value of the function is rather flat in the range of $0.1m \sim 0.5m$ according to the function plot shown in figure 5.21. This means the reward for the laser data almost serves as a constant value so there would be no difference for the agent whether the end-effector of the robot is close to or far away from the pipe. In other words, the reward contributes nothing to the training process and causes the failure of navigating the robotic arm to move through pipes with acute angles.

Therefore, the parameter β and d_0 in equation 3.10 need to be adjusted to increase the slope of the reward function when the range of the laser data is between $0.1m$ and $0.5m$. In the end, **it is decided to choose $\beta = 1.0$, $d_0 = 0.001$** as the new parameters. With the new configuration of the reward function, the agent is able to find a fairly good policy in which it can control the robotic arm to move through every pipe during the training.

Evaluation phase in real world

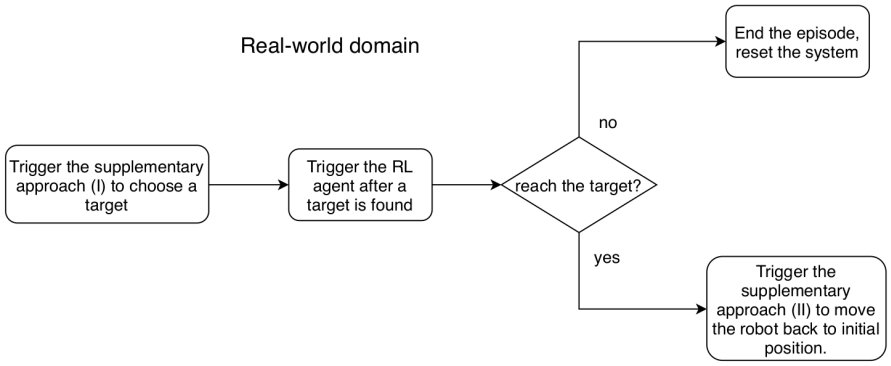


Figure 5.22: Flow chart of the evaluation phase on the real robotic system

Next, the obtained RL agent together with the supplementary approach (I) and (II) are evaluated onto the real robotic system which has been introduced in section 4.3. As mentioned before, the diameter of the pipe-like obstacles is at a minimum of $130mm$ restricted by the connection between the robotic arm and the wooden board below it. Additionally, three different angles for the pipe corner are selected: 60°, 90° and 120°, which include an acute angle, a right angle and an obtuse angle.

The RL agent is directly deployed onto the real system, which means the state vector is formulated by using the sensory data (joint states and the laser data) from the real robotic arm and the agent will directly send the action (position commands) to the motors which actuate the joints of the robot. The flow chart of the evaluation is illustrated in figure 5.22. Since it is difficult to re-spawn the robot in the real world to the configuration which represents the initial position for the rear part of the PIRATE, as illustrated in figure 5.12, it is decided to simply move the robotic arm back to its initial position after the target is reached in order to evaluate the supplementary approach (II).

Soon after the evaluation starts, multiple problems appear and deteriorate the success rate of the overall navigation strategy in the real world. The main observations made during the evaluation phase are as follows:

1. As the robot is looking for a suitable target point, the data from the laser scanner is too noisy and often leads to the failure of detecting an open area on the other side of the pipe. Specifically, it is decided in the program that when any of the data point from the laser is larger than $0.3m$, an open area is considered to be found. However, since the data is too noisy in the real world, the laser readings often return $0.1m$ (the minimal vertical range of the laser) even the end-effector is just facing to an open space inside the pipe. This inaccuracy adds considerable difficulty in locating a target point for the robot. In average, the success rate of determining a suitable target is less than 60%.

2. The second problem is that when the RL agent is triggered, the robot does not behave exactly the same as it does in the simulation, due to the modeling difference between the real world and Gazebo simulator as well as the noisy data from the laser scanner. This leads to the observation that there are many collisions happening between the pipe and the robot and most of the collisions are significantly strong. This frequently stops the movement of the robot in the pipe and the success rate of reaching the target drastically drops.

3. Despite the problems mentioned above, there are still cases where the robot successfully move through the pipe corner, reaching the target point, and go back to its initial position by following the control of the supplementary approach (II). Appendix A illustrates three moving sequences in which the robot manages to move through the 60°, 90° and 120°

pipe corner respectively. It is observed that the robot can always move back to its initial configuration after reaching the target. The supplementary approach (II) does not have differences in the real world with it does in the simulation.

To sum up, the strategy does not have a good performance on the real setup and most of the time in the experiment was spent on the tuning of the robotic system. Due to the limit amount of time, there is no quantitative data which has been recorded.

5.5 Discussions

In the research of the RL-based path planner for the front part of PIRATE, the resulting agents can navigate the robot to reach the target point inside various pipe corners with different diameters and turning angles, although the thinner the pipe is, the harder for the robot to achieve the task.

In torque control experiment, it is found that the penalty on the distance between the end-effector and the pipe influences the performance of the RL agents: if it is too small, the end-effector tends to collide with the pipe frequently so the robot is easier to be stuck; if it is too large, the end-effector will always keep a relatively long distance with the pipe and can never reach the goal. A medium penalty then holds a good balance and leads to an agent which gives the best performance.

In position control experiments, it is found that the RL-based approach performs poorer when the robot is controlled under position control scheme than under torque control scheme. The probable reason is that under torque control the robot has softer joints so the collisions between the pipe and the robot is less strong, leading to a smaller possibility for the robot to stop moving. However, integrating the state history of the environment and an extra penalty on the change of the joint positions into the training help improve the performance of the RL agents under position control.

In addition, the supplementary approach (II) generates the paths for the rear part of PIRATE to move through the pipes by reversing and mirroring the joint trajectories collected from the front part of the robot. This is due to the assumption that the moving process for the two parts of PIRATE are actually mirrored in space and reversed in time. Testing the approach with the robotic arm turns out that it works quite effectively when the robot is inside pipes with a relatively large diameters such as $120, 110, 100mm$, but fails when the diameter becomes even smaller ($90mm$). This is due to the fact that the end-effector tends to be stuck when it is passing the corner of a pipe with a small diameter.

In the high fidelity experiments, the success rate for both the front part and the rear part of PIRATE to pass the corner of the pipe significantly drops when the meshes of the robotic arm are replaced with those from the PIRATE model. The new meshes have wider dimensions and more complex structures so the robot gets stuck in the pipes more often than before. However, the supplementary approach (I) in which the robot can locate and determine a suitable target itself helps increase the success rate for both parts of PIRATE to move through the pipe corners. The reason for this improvement is that by first looking for a target point, the trajectory of the robot gets changed and the possibility for the robot to get stuck in the pipe is reduced, leading to an increase of the success rates. In the end, the reinforcement learning agent together with the two supplementary approaches form a complete path planning strategy for the PIRATE robot to move through different pipe corners.

Finally, the overall approach is tested onto a real robotic system adopted from an old setup in the RAM lab. It is quickly found that the performance of the approach is deteriorated by the noisy laser data, leading to the frequent failure of locating a suitable target. Besides, even after a target point is found, the robot does not always succeed in reaching it since the policy of the RL agent is trained on simulation and cannot perfectly fit the real world situations. Therefore,

the performance of the overall strategy should be improved if a better laser scanner with a more accurate output and a smaller vertical range replaces the current one and a more accurate simulation environment during the training can be built. However, due to time issues, this part will be left to the future work.

6 Conclusions and recommendations

6.1 Conclusions

As mentioned in chapter 1, the objective of this master project is to develop a robust navigation strategy for PIRATE so that it can move through sharp pipe corners by leveraging reinforcement learning. until now, the approaches developed within the thesis have not completely achieved this ultimate objective but can be served as one of the possible ways of approaching it.

For a better critical appraisal of the work done in this project, the strength and weakness of the proposed navigation approach should be carefully discussed.

Strength

- **No mathematical model of the environment required**

  The RL algorithm used in this project does not need a model of the environment, which means the policy is learned by only using the data from the interaction between the agent and the environment. This reduces the effort of building a accurate mathematical model and deriving the policy out of it as in traditional approaches.

- **Cover the strategies for both the front and rear part of PIRATE**

  The proposed approach covers the navigation problems for both the front and the rear part of PIRATE. In contrast, in [4] it only mimics the situation for the front part of the robot with a 3-DOF robotic arm.

- **Generalizability**

  As shown in the results chapter (chapter 5), the proposed RL-based approach together with the supplementary approach (I) and (II) are able to navigate the robot to move through various types of pipe corners which have different diameters and turning angles. Especially, the training for the RL agent is done inside pipes with $120mm$ diameter, but during the evaluation phase the overall approaches can be generalized to navigate the robot inside pipes with $110mm$, $100mm$ and $90mm$ diameters and also with turning angles that have never appeared during the training. This generalizability is crucial for the PIRATE robot because it can encounter many different pipes when being applied in the real industry, so the navigation strategy the robot follows must fit in multiple different environments.

Weakness

- **The tests in the real world is incomplete**

  Although an experiment in the real world has been conducted, the results is not very optimistic mainly because the data from the laser scanner is too noisy and the simulation model used in the training is only a rough approximation of the robot in the real world. These issues cannot be fixed due to the limited amount of time left, which makes the experiments in real world not complete and reliable enough.

- **Development and evaluations on a simplified model**

  As mentioned earlier, a 4-DOF robotic arm is used as a substitute for the PIRATE robot because there are no running models of PIRATE existed in either simulation or the real world. Therefore, whether the proposed approach in this thesis is workable onto the PIRATE remains unknown.

6.2 Recommendations for future work

Although the overall approach presented in this thesis achieves a fairly good performance when the robot is moving inside multiple different pipes, there are still several aspects that can be improved.

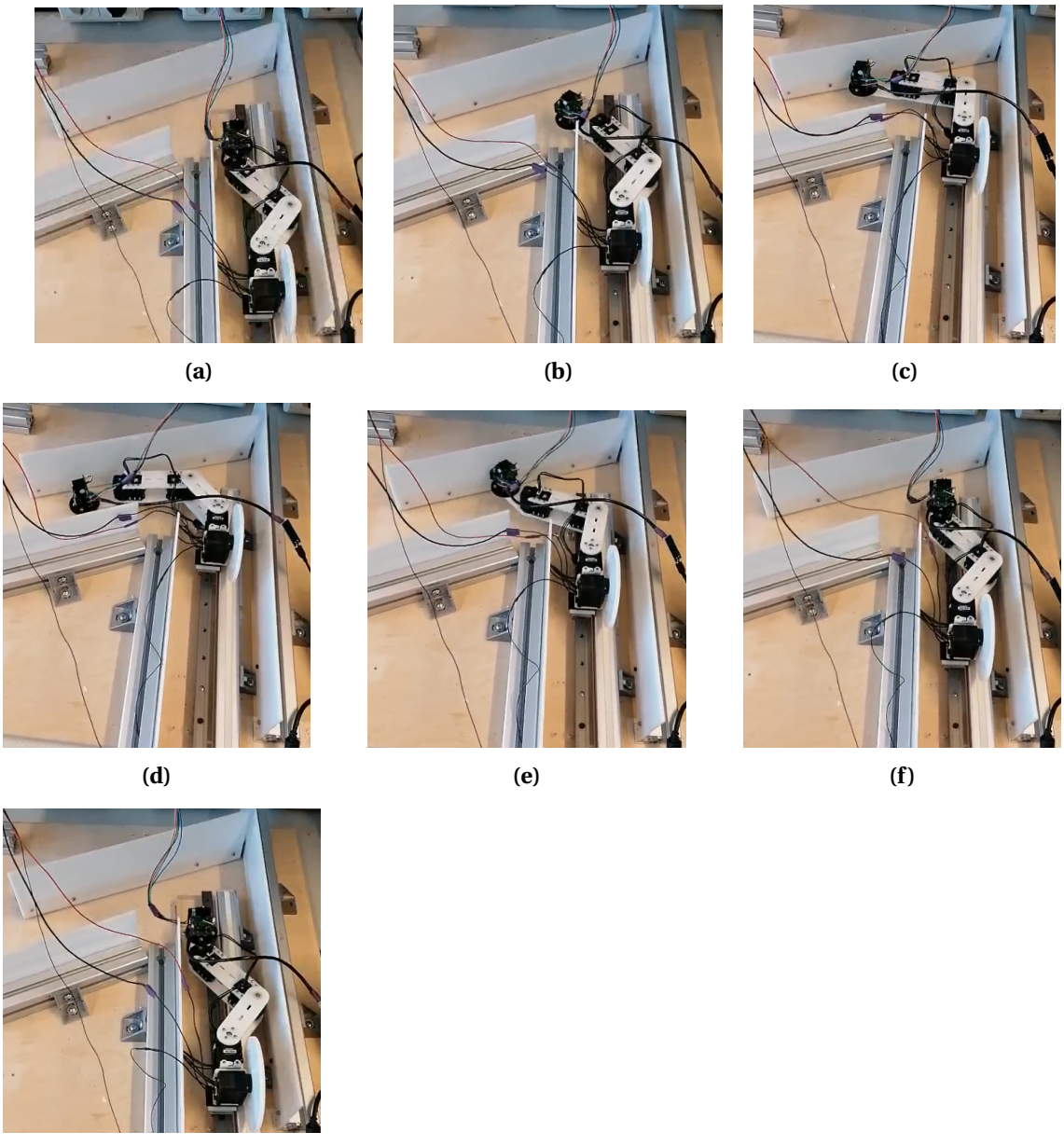
First, the success rate for the front part of PIRATE moving through the pipes with a diameter of $90mm$ is comparably low according to figure 5.18 in the final experiment. This is because the agent gets trained when the robot is inside $120mm$ pipes so the behaviour it develops is not suitable for the navigation in much narrower pipes. However, the common working environment for PIRATE is inside pipes with a diameter from $70mm$ to $120m$, the results obtained in this thesis is obviously not good enough for tackling with the real situations.

In order to improve the performance of the overall approach, one possible way is to let the RL agent learn behaviours which are more sophisticated. For instance, when the robot gets stuck in the pipes, the agent should be aware of it and can drive the robot away from this “bad" configuration to those in which the robot is free to move. This requires, in my opinion, to employ more advanced RL methods such as hierarchical reinforcement learning, where multiple layers of agents work together from top to bottom to execute complicated behaviours; or model-based reinforcement learning, where the agent can possess an inner dynamics model of the robot so that it can detect whether the robot is having collisions with obstacles or not.

Second, it should be pointed out that the 4-DOF robotic arm deployed in this thesis does not fully capture the behaviour of the PIRATE, even after the meshes for the links have been replaced. For instance, the transition between the first phase and the second phase of the moving process is not properly modeled during the simulation; there is a “jump" between the final position for the front part and the initial position for the rear part. In addition, the model does not have wheels on it as the real PIRATE does, which may lead to inaccurate behaviour when the robot have contacts with the pipe. Therefore, a comparable simulation model of the whole PIRATE is required for conducting better research.

Finally, more experiments in the real world should be conducted. The proposed approach in this thesis has been massively evaluated in the simulation but lacks approvals from the real robotic environment. To complete the experiment, a better laser scanner is definitely needed, which should have less noisy readings, a smaller minimal vertical range ($0.01m$ ideally) and a larger horizontal range ($180°$ should be perfect). In addition, approaches that can make the RL-agent robust to the differences between the real world and the simulation should also be explored, so that the performance of the navigation strategy in the reality is as close as that in a simulator.

A Appendix: moving sequence of the real robot

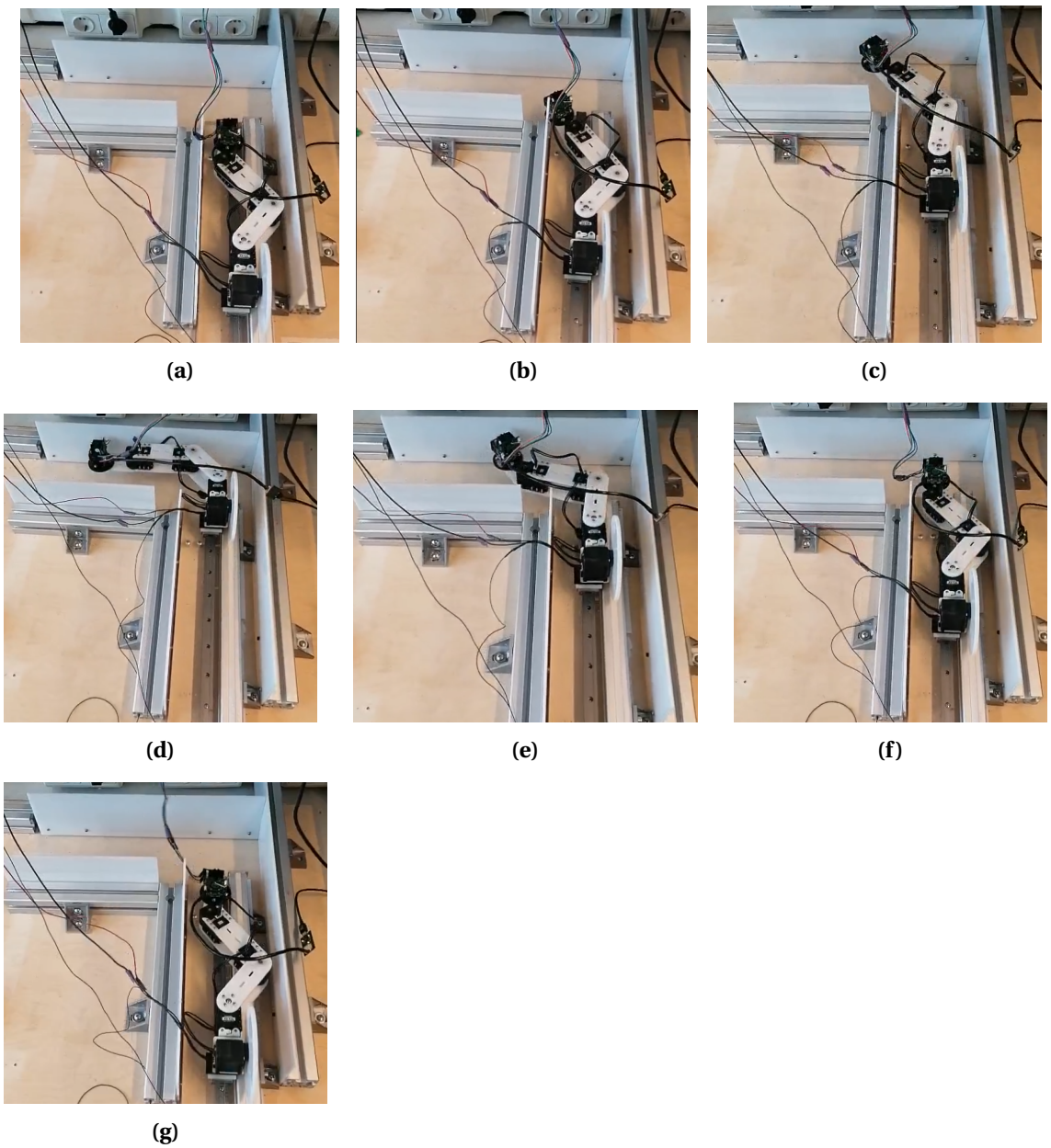


(a) **(b)** **(c)**

(d) **(e)** **(f)**

(g)

Figure A.1: The sequence of movement that the robot go through a $\mathbf{60°, 130}\boldsymbol{mm}$ pipe corner and move back to initial position.

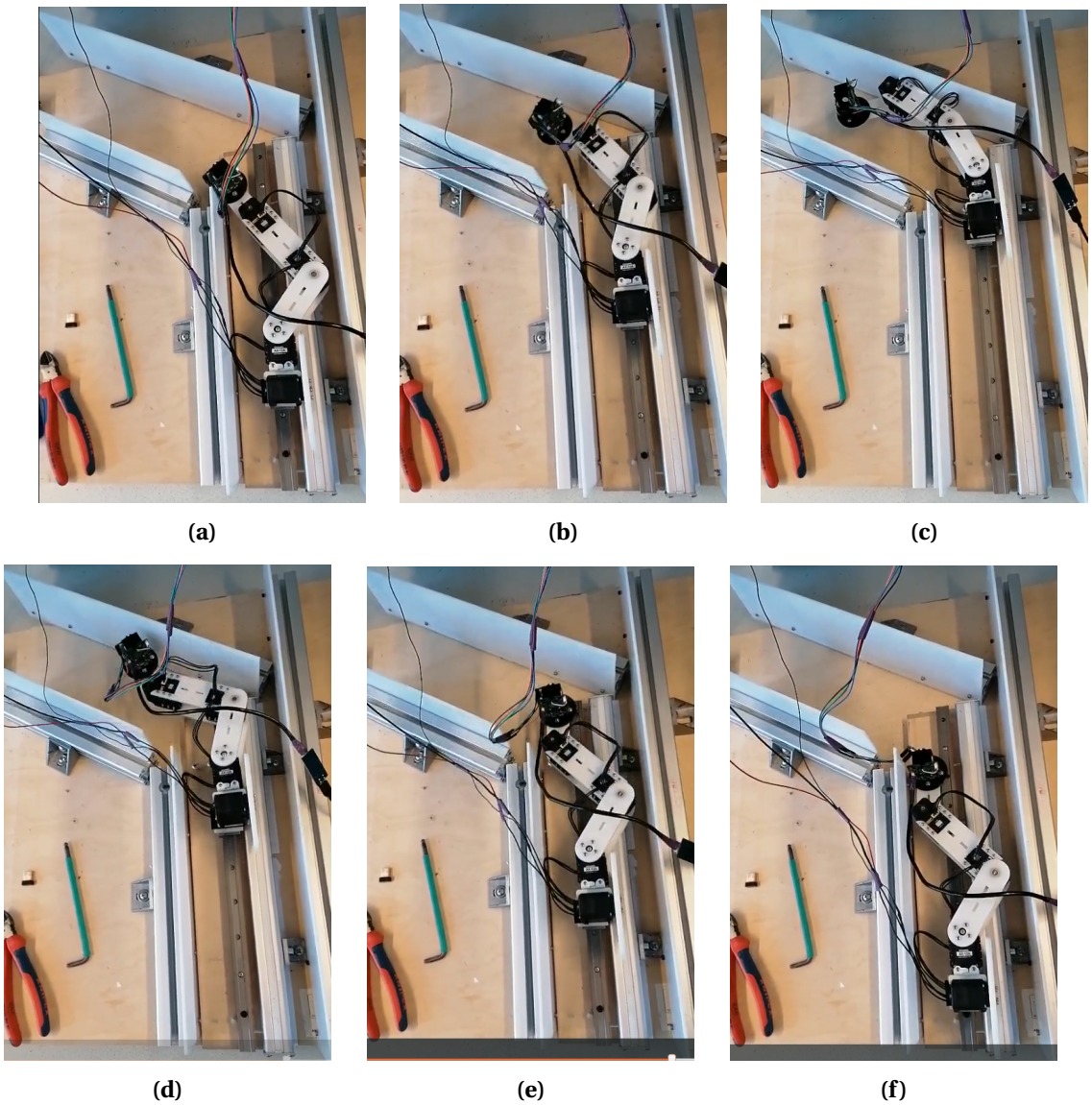


(a) **(b)** **(c)**

(d) **(e)** **(f)**

(g)

Figure A.2: The sequence of movement that the robot go through a $\mathbf{90°, 130}\boldsymbol{mm}$ pipe corner and move back to initial position.



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figure A.3: The sequence of movement that the robot go through a $\mathbf{120°, 130}\boldsymbol{mm}$ pipe corner and move back to initial position.

Bibliography

[1] Dertien, E. C., *Design of an inspection robot for small diameter gas distribution mains*, PhD thesis, University of Twente, Enschede.

[2] G.A. Garza Morales, *Increasing the Autonomy of the Pipe Inspection Robot PIRATE*, Master thesis, University of Twente, 2016

[3] N.M. Geerlings, *Design of a high-level control layer and wheel contact estimation and compensation for the pipe inspection robot PIRATE*, Master thesis, Delft University of Technology & University of Twente, 2019

[4] Marta Barbero, *Reinforcement Learning for Robot Navigation in Constrained Environments*, Master thesis, University of Twente, Enschede, Netherlands, 2018

[5] Richard S Sutton and Andrew G Barto, *Reinforcement learning: An introduction*, MIT press Cambridge, 1998.

[6] Sergios Karagiannakos, *The idea behind Actor-Critics and how A2C and A3C improve them*, https://sergioskar.github.io/Actor_critics/, 2019

[7] John Schulman, Sergey Levine, Philipp Moritz, *Trust Region Policy Optimization*, Proceedings of the 31 st International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.

[8] Ziyu Wang, Volodymyr Mnih, Victor Bapst, *Sample efficiency actor-critic with experience replay*, 5th International Conference on Learning Representations, 2017

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, *Proximal Policy Optimization Algorithms*, arXiv.org, 2017

[10] OpenAI, *Trust Region Policy Optimization*, https://spinningup.openai.com/en/latest/algorithms/trpo.html, 2018

[11] OpenAI, *Proximal Policy Optimization*, https://openai.com/blog/openai-baselines-ppo/, 2017

[12] OpenAI Spinning Up, *Proximal Policy Optimization*, https://spinningup.openai.com/en/latest/algorithms/ppo.html, 2018

[13] Nicolas Heess, Dhruva TB, Srinivasan Sriram, *Emergence of Locomotion Behaviours in Rich Environments*, arXiv:1707.02286v2, 2017

[14] Yuke Zhu, Ziyu Wang, Josh Merel, *Reinforcement and Imitation Learning for Diverse Visuomotor Skills*, arXiv:1802.09564v2, 2018

[15] Andrea Franceschetti, Elisa Tosello, Nicola Castaman, and Stefano Ghidoni, *Robotic Arm Control and Task Training through Deep Reinforcement Learning*

[16] Cybenko, G., *Approximations by superpositions of sigmoidal functions*, Mathematics of Control, Signals, and Systems, 2(4), 303–314, 1989

[17] Kurt Hornik, *Approximation Capabilities of Multilayer Feedforward Networks*, Neural Networks, 4(2), 251–257, 1991

[18] Hanin, B, *Universal function approximation by deep neural nets with bounded width and ReLU activations*, arXiv, 2017

[19] Tambet Matiisen, Guest Post (Part I): Demystifying Deep Reinforcement Learning, https://ai.intel.com/demystifying-deep-reinforcement-learning/

[20] J. Kober, J.A. Bagnell, and J. Peters, *Reinforcement Learning in Robotics: A Survey*, The International Journal of Robotics Research, Vol. 32, pp. 1238-1274, 2013.

[21] Andrea Franceschetti, Elisa Tosello, Nicola Castaman, and Stefano Ghidoni, *Robotic Arm Control and Task Training through Deep Reinforcement Learning*

[22] Shixiang Gu, Ethan Holly, Timothy Lillicrap, Sergey Levine, *Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates*, 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, May 29 - June 3, 2017

[23] John Schulman, Philipp Moritz, Sergey Levine, *HIGH-DIMENSIONAL CONTINUOUS CONTROL USING GENERALIZED ADVANTAGE ESTIMATION*, International Conference on Learning Representations, 2016.

[24] Jemin Hwangbo1, Joonho Lee1, Alexey Dosovitskiy, *Learning agile and dynamic motor skills for legged robots*, Science Robotics, 2019.

[25] RAM, University of Twente, *PIRATE - Overview*, https://www.youtube.com/watch?v=QKWz-J91Icw, 2017

[26] C. Pulles, E.Dertien, H.J. van de Pol, *Pirate, the development of an autonomous gas distribution system inspection robot*, International Gas Union Research Conference, Paris, 2008

[27] *ROS/Introduction*, http://wiki.ros.org/ROS/Introduction, ROS wiki.

[28] Robotis, *USB2DYNAMIXEL e-Manual*, in http://support.robotis.com/en/product/auxdevice/interface/usb2dxl_manual.htm.

[29] MahanFathi, TRPO-TensorFlow, 2018, GitHub repository, https://github.com/MahanFathi/TRPO-TensorFlow

[30] yjhong89, TRPO with GAE, 2017, GitHub repository, https://github.com/yjhong89/TRPO-GAE

[31] A. Ezquerro, M. Angel Rodriguez and R. Tellez, *openai_ros*, http://wiki.ros.org/openai_ros

[32] Jie Tan, Tingnan Zhang, Erwin Coumans, *Sim-to-Real: Learning Agile Locomotion For Quadruped Robots*, arXiv:1804.10332v2, 2018