

## MASTER

### Learning how to solve the Buzz-wire game with a robot arm

Dorussen, R.J.G.

*Award date:*  
2021

[Link to publication](#)

#### **Disclaimer**

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

#### **General rights**

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

# Learning how to solve the Buzz-wire game with a robot arm

*Graduation report*

R.J.G. Dorussen  
0968849

Master: Systems & Control  
Department: Mechanical Engineering  
Research group: Control Systems Technology

Supervisors: dr.ir. B. Vet (DEMCOM)  
dr.ir. S.H.J. Heijmans (DEMCON)  
dr. D.J. Antunes  
prof.dr.ir. W.P.M.H. Heemels

Reports ID: CST2021.003

Eindhoven, February 5, 2021



## Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct<sup>1</sup>.

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

Date

18-2-2021

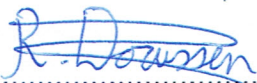
Name

Rob Doumaen

ID-number

227182455

Signature



*Submit the signed declaration to the student administration of your department.*

<sup>1</sup> See: <http://www.tue.nl/en/university/about-the-university/integrity/scientific-integrity/>

The Netherlands Code of Conduct for Academic Practice of the VSNU can be found here also.  
More information about scientific integrity is published on the websites of TU/e and VSNU



## Summary of notation

Both capital letters and lower case letters are used for variables and scalar functions. Quantities that are required to be real-valued vectors are in bold and lower case. Matrices are in bold capitals. Capitals written in calligraphic, indicate a set.

<b>Symbol</b>	<b>Description</b>
$\subset$	subset of, e.g. $\mathcal{A} \subset \mathbb{R}_{\geq 0}^n$
$\in$	is an element of, e.g. $a_k \in \mathcal{A}$
$ \dots $	absolute value
$\#$	cardinality or number of elements of a set, e.g. $\#\mathcal{A}$ is the number of possible actions
$\arg \max_a g(a)$	value of $a$ at which $g(a)$ takes its maximal value
$\alpha$	learning rate, $\alpha \in \mathbb{R}$
$\alpha_e$	rotation of the wire at episode $e$ , $\alpha_e \in [0, 2\pi]$
$\gamma$	discount factor, $\gamma \in [0, 1]$
$\epsilon$	probability of choosing a random action, $\epsilon \in [0, 1]$
$\theta$	rotation of the loop, $\theta \in [0, 2\pi]$
$\theta_{step}$	step-size in the rotation of the loop, $\theta_{step} \in [0, 2\pi]$
$\boldsymbol{\theta}$	feature describing the rotation of the loop, $\boldsymbol{\theta} \in \mathbb{N}_{\leq 1}^{n_\theta}$
$\boldsymbol{\pi}$	policy, matrix of $ \mathcal{S}  \times  \mathcal{A} $ containing the probability of taking action $a$ in state $s$ , $\pi(a s) \in [0, 1]$
$\phi$	feature describing the relative angle between the wire and the loop, $\phi \in \mathbb{N}_{\leq 1}^{n_\phi}$
$a$	action, $a \in \mathcal{A}$
$a_e$	amplitude of the sine wave of the wire at episode $e$ , $a_e \in \mathbb{R}$
$a_k$	action at time $k$ , $a_k \in \mathcal{A}$
$a_{max}$	maximum amplitude of the sine wave of the buzz-wire, $a_{max} \in \mathbb{R}_{\geq 0}$
$D$	all data in the replay buffer, set of experiences, $D_k = \{e_0, \dots, e_k\}$
$\mathbf{d}_{A,k}$	distance from point $A$ to the wire at time-step $k$ , $\mathbf{d}_{A,k} \in \mathbb{R}_{\geq 0}$
$\mathbf{d}_{AI_2}$	feature describing the distance to crosspoint of the wire with interval AC, $\mathbf{d}_{AI_2} \in \mathbb{N}_{\leq 1}^{n_d}$
$\mathbf{d}_{BI_1}$	feature describing the distance to crosspoint of the wire with interval BD, $\mathbf{d}_{BI_1} \in \mathbb{N}_{\leq 1}^{n_d}$
$\mathbf{d}_{goal}$	distance from starting position along the buzz-wire to the finish, $\mathbf{d}_{goal} \in \mathbb{R}_{\geq 0}$
$\mathbf{d}_{loop}$	diameter of the loop, $\mathbf{d}_{loop} \in \mathbb{R}_{\geq 0}$
$\mathbf{d}_{roi}$	distance between loop center and point on the wire used to calculate feature $\phi$ , $\mathbf{d}_{loop} \in \mathbb{R}_{\geq 0}$
$e$	episode, $e \in \mathbb{N}$
$e_k$	experience of the agent, $e_k = (o_k, a_k, r_k, o_{k+1})$
$f_e$	frequency of the sine wave at episode $e$ , $f_e \in [\frac{1}{d_{goal}}, \frac{n_{waves}}{d_{goal}}]$
$h_{obs}$	height of the camera image, $h_{obs} \in \mathbb{R}_{\geq 0}$
$h_{px}$	height of a pixel of the camera image, $h_{px} \in \mathbb{N}_{\geq 1}$
$i_k$	index of the point on the wire to which the loop is projected
$G_k$	discounted cumulative reward
$k$	discrete time step, $k \in \mathbb{N}$
$L$	loss, quantifies the distance between target and real value of a training example, $L \in \mathbb{R}_{\geq 0}$
$n_\theta$	number of segments to in which the loop angle is discretized, $n_\theta \in \mathbb{N}_{\geq 1}$
$n_\phi$	number of segments in which the relative angle between the wire and loop is discretized, $n_\phi \in \mathbb{N}_{\geq 1}$
$n_{check}$	number of checkpoints along the wire, $n_{check} \in \mathbb{N}_{\geq 1}$
$n_d$	number of segments in which line segments BD and AC are discretized, $n_d \in \mathbb{N}_{\geq 1}$
$n_{eval}$	number of episodes in the validation set, $n_{eval} \in \mathbb{N}_{\geq 1}$
$n_{wall}$	number of elements in the walls at the ends of the buzz-wire, $n_{wall} \in \mathbb{N}_{\geq 1}$
$n_{waves}$	maximum number of waves in the sine wire, $n_{waves} \in \mathbb{N}_{\geq 1}$
$n_{wire}$	number of episodes in the buzz-wire, $n_{wire} \in \mathbb{N}_{\geq 1}$
$o$	observation, $o \in \mathcal{O}$
$o_k$	observation at time $k$ , $o_k \in \mathcal{O}$
$p(s_{k+1} s_k, a_k)$	probability of transition to state $s_{k+1}$ given state $s_k$ and action $a_k$
$Q_\pi(s, a)$	value of taking action $a$ in state $s$ under policy $\pi$ , $Q_\pi(s, a) \in \mathbb{R}$
$Q_*(s, a)$	value of taking action $a$ in state $s$ under the optimal policy, $Q_*(s, a) \in \mathbb{R}$

$r$	reward, $r \in \mathcal{R}$
$r_k$	reward at time $k$ , $r_k \in \mathcal{R}$
$r_{wire}$	thickness of the wire, $r_{wire} \in \mathbb{R}_{\geq 0}$
$s$	state, $s \in \mathcal{S}$
$s_k$	state at time $t$ , $s_k \in \mathcal{S}$
$t_{loop}$	thickness of the loop, $t_{loop} \in \mathbb{R}_{\geq 0}$
$T$	time step at the terminal state, $T \in \mathbb{N}_{\geq 1}$
$u$	local coordinate in the 2D environment, longitudinal direction of the loop
$u_{step}$	step-size in the longitudinal direction of the loop, $u_{step} \in \mathbb{R}_{\geq 0}$
$\mathbf{W}$	matrix containing the weights with size $m \times n$ , where $m$ is the number of output neurons and $n$ the number of input neurons
$w$	local coordinate in the 2D environment, lateral direction of the loop
$w_{obs}$	width of the camera image, $w_{obs} \in \mathbb{R}_{\geq 0}$
$w_{px}$	width of a pixel of the camera image, $w_{px} \in \mathbb{N}_{\geq 1}$
$w_{step}$	step-size in the lateral direction of the loop, $w_{step} \in \mathbb{R}_{\geq 0}$
$x$	global coordinate of the 2D environment
$y$	global coordinate of the 2D environment
$\mathbf{y}$	vector containing the calculated outputs of a layer of neurons
$\mathcal{A}$	set of all possible actions, a finite subset of $\mathbb{R}$
$\mathcal{O}$	set of all possible observations, a finite subset of $\mathbb{R}^{2n_a+n_\theta+n_\phi}$
$\mathcal{R}$	set of all possible rewards, a finite subset of $\mathbb{R}$
$\mathcal{S}$	set of all possible states, a finite subset of $\mathbb{R}^{3+2n_{wire}}$
$\mathbb{E}_\pi[X]$	expectation of random variable $X$ under a policy $\pi$
$\mathbb{N}$	set of positive integers $[0, 1, 2, \dots]$
$\mathbb{N}_{\geq 1}$	set of positive integers larger than 0, $[1, 2, \dots]$
$\mathbb{N}_{\leq 1}$	set of positive integers smaller than or equal to 1, $[0, 1]$
$\mathbb{R}$	set of real numbers
$\mathbb{R}_{\geq 0}$	set of real positive numbers

## Abstract

AI and Machine Learning (ML) are expected to play crucial roles in the future world. New applications of AI and ML range from smart irrigation systems in the food industry to systems regulating the power grid in the energy sector. DEMCON Advanced Mechatronics is interested in the solutions that AI can provide. In particular, ML combined with computer vision and robotics can enable applications such as autonomous fruit picking robots.

A ML technique that fits the fruit picking robot example particularly well is Reinforcement Learning (RL). In RL the robot arm (the agent) needs to learn which moves (actions) it should take given a certain situation (observation) to pick fruit (maximize reward). Beforehand, it is unknown to the agent which action for a given observation results in reward. Therefore, random actions are explored. After some exploration, the gained knowledge can be exploited to obtain more reward. At each time step the agent can decide to take actions leading to reward based on current knowledge, or to take different actions to check if these can result in more reward. In RL there is always a trade-off between exploration and exploitation.

As the fruit picking robot example is very complex, DEMCON first wants to develop a demonstrator on the buzz-wire game. This problem is significantly less complex, allowing for a better understanding of how RL problems are solved and lowering the time needed to train the agent. However, the training time on the actual hardware might still be too high and requires human supervision. By creating a model and using transfer learning, these problems can be resolved. This graduation project focusses on implementing an RL algorithm on a 2D model of the buzz-wire problem, to gain the competence needed to solve more complex problems. To this effect, the agent is setup such that the environment can easily be adapted for future work.

The goal of the buzz-wire game is to actuate the loop along the wire, without hitting it. Each time the wire is hit, negative reward is given to the agent. When the finish is reached, the agent receives positive reward. So in other words, the agent should maximize the cumulative reward over an episode through the proper selection of actions based on the observations it receives. The observations are 4 features which can be extracted from images supplied by a camera, that is attached to the wrist of the robot arm.

The RL algorithm that is implemented is the Deep Q-learning algorithm (DQN). This uses Artificial Neural Networks (ANNs) to approximate the so-called optimal  $Q$ -function. A  $Q$ -function is defined as the expected future reward given an action, observation and policy. The optimal  $Q$ -function is defined similarly for the optimal policy. This function is approximated based on data. At each time step the agent receives an observation and selects an action. The method used for the agent to generate actions is an  $\epsilon$ -greedy policy. This means that the agent picks the most promising action so far with probability  $1 - \epsilon$  and with probability  $\epsilon$  it picks a random action. Then the agent performs the selected action in the environment and receives a reward and a new observation. This is all stored in the replay buffer, from which samples are taken to update the  $Q$ -function.

The 2D discrete model of the buzz-wire setup is represented as a Partially Observable Markov Decision Process (POMDP), because the environment is only partially observable as the camera does not capture the complete wire. In this framework the state consists of the location of both the loop and the wire and its shape. The action consists of translation and rotation of the loop. Based on whether the action results in hitting the wire, reaching the finish or neither, the agent gets reward. The observation consists of: the distance between the wire and the loop at both the front and back of the loop, the orientation of the loop and the relative angle between the loop and the wire.

Using the agent and the 2D model, several policies have been trained. From training the policies, it became clear that the orientation of the loop is useless as it does not correlate to any of the actions. Next to that, comparing several policies shows that implementing continuous reward has a negative influence on the final policy, even though the policy converges faster during training. When the policy is trained on continuous reward, the minimum reward of the validation set is negative. This means that the loop has hit at least one wire and got stuck. The policy trained without continuous reward does not have this negative minimum reward. Therefore, it does not hit any wire of the validation set. Moreover, when



tested on a wire outside the training set, the policy is still able to reach the finish. This shows that the policy has generalized well.

Since the policy trained without continuous reward performs well in the simulations, it was implemented on an experimental setup comprised of the Niryo One robotic arm and an ad-hoc wire. Here several problems were encountered. A shape of the wire is modelled for the observation because the features from the camera images can not yet be obtained. As the actual wire is shaped differently from this modelled wire, the observations are not always accurate. Next to that, the robot arm drifts in other directions than in which it moves. This results in the wire being hit, when the agent does not expect it to.

# Contents

<b>Summary of notation</b>	<b>i</b>
<b>Abstract</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Problem statement</b>	<b>5</b>
2.1 Research questions . . . . .	6
<b>3 The learning algorithm</b>	<b>7</b>
3.1 Deep Q-Learning (DQN) . . . . .	7
3.2 Data collection . . . . .	8
<b>4 Modelling the 2D environment</b>	<b>9</b>
4.1 State . . . . .	9
4.2 Action . . . . .	10
4.3 Transition . . . . .	10
4.4 Reward . . . . .	11
4.5 Observation . . . . .	11
<b>5 Performance and validation of the trained policies</b>	<b>13</b>
5.1 Training policy A on an environment with continuous reward . . . . .	13
5.2 Training policy B on an environment without continuous reward . . . . .	14
5.3 Performance on wire with four corners to the left . . . . .	16
5.4 Performance on the real setup . . . . .	17
<b>6 Conclusions and recommendations</b>	<b>21</b>
6.1 Conclusions . . . . .	21
6.2 Future work . . . . .	21
<b>References</b>	<b>23</b>
<b>A Implemented agent</b>	<b>25</b>
<b>B Sets of 2D wires</b>	<b>27</b>
<b>C Implementation of the 2D model</b>	<b>29</b>
<b>D Settings of the training sessions</b>	<b>33</b>
<b>E Problems during training</b>	<b>35</b>
<b>F Results of the hysteresis experiments on the Niryo One robot arm</b>	<b>37</b>



# 1 Introduction

Over the past decades, many research fields and industries have been interested in Artificial Intelligence (AI) to increase productivity. For instance in the food industry, it is expected that AI and computer vision are part of the necessary improvements that will increase the food supply [1]. This increase is needed to sustain the future world population, which is expected to reach around 9.1 billion by 2050 [2]. At the same time other industries can benefit from infusing their systems and products with AI [3]. Examples include management systems in the energy sector which can handle variable loads on the power grid created by renewable energy sources [4], controllers for welding robots in the manufacturing industry that improve weld quality [5] or even AI models in the medical sector that can help physicians in detecting infection regions for the coronavirus 2019 (COVID-19) [6].

As one of the players in many of these industries, DEMCON Advanced Mechatronics is interested in the solutions provided by AI, especially in combination with computer vision and robotics. For example, AI can be key for creating an autonomous fruit picking robot, see Figure 1.1. Controlling a robot arm with AI comes with many challenges. First of all, the robot should be able to recognize the fruit and determine whether it is ripe or not. Additionally, twigs, leaves, or other fruit can obstruct both the view of the ripe fruit and the path towards it. To enable the robot arm to detect the fruit and move towards it while avoiding obstructions, information about the surroundings is needed. The surroundings can change as leaves, twigs, and fruit all move with the growth of the plants. Furthermore, external forces, such as wind, might move the leaves and twigs. All in all, there is a large amount of uncertainty in the environment due to the vast amount of variation, which requires a general and robust solution. Fortunately, models created with Machine Learning (ML) have shown to generalize a variety of tasks well [7], from pouring liquids into containers of various sizes [8] to pushing randomly shaped object [9].



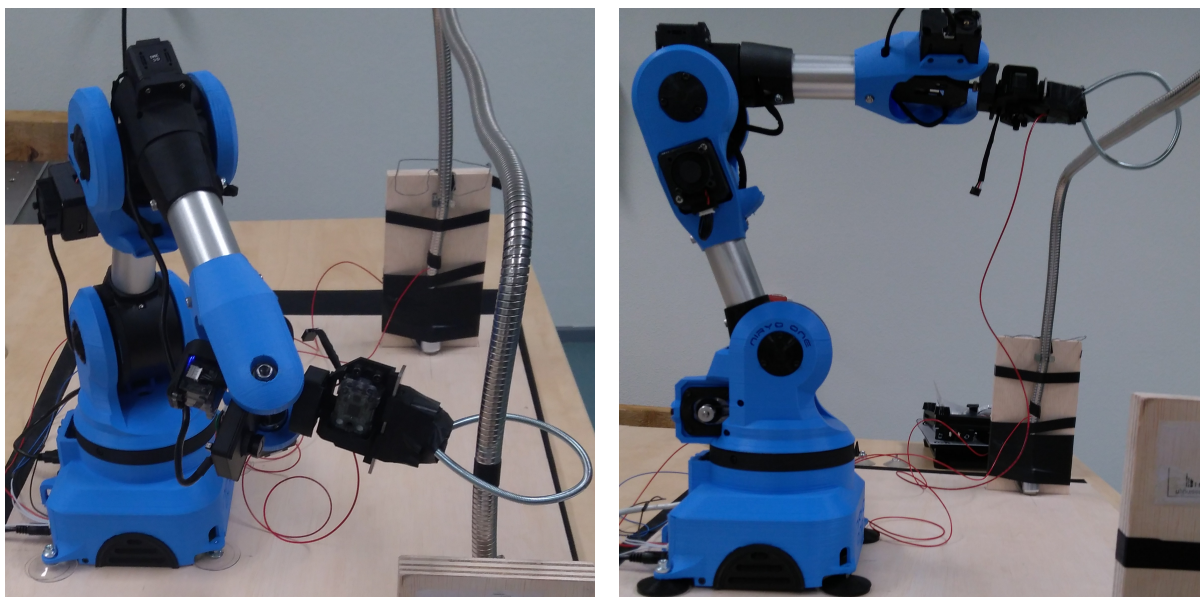
Figure 1.1: *Example of a fruit picking robot.*

An ML technique that fits the fruit picking robot example particularly well is Reinforcement Learning (RL). In RL the agent needs to learn what actions it should take to maximize its reward [10]. This fits the fruit picking problem since the controller of the robot arm (agent) does not know which moves result in hitting twigs or leaves (environment) or reaching the fruit and picking it. To find out which moves lead to picking the fruit and getting a reward the agent needs to try several moves (explore). When the agent has discovered a sequence of actions which result in fruit being picked, it can use this knowledge

to get reward (exploit). It knows that for certain observations a specific set of actions results in reaching the observed fruit. Alternatively, the agent can keep exploring to find a sequence of actions that possibly results in a larger reward in the future. RL is the only ML technique that considers the trade-off between exploration and exploitation [11]. Next to that, the impact of a given action is not immediately clear. Thus rewards are delayed and RL can handle this in contrast to other ML techniques such as Supervised Learning.

RL uses several complex high dimensional and nonlinear functions, for example the policy (which is the mapping from observations to the actions). Estimating high dimensional complex functions is difficult. However, Artificial Neural Networks (ANNs) can approximate arbitrary nonlinear functions, as proved by the universal approximation theorem [12]. For this reason, ANNs are utilized during this project.

Even though RL fits the fruit picking example, extensive knowledge on solving RL problems is needed for the development of an algorithm due to the complexity of the problem. To obtain the required competence, a simplified problem is studied first. This problem still has to include essential aspects of the fruit picking example (path planning and avoiding obstacles) while omitting some of the complexity (extra task of recognizing the fruit and changing surroundings). As a result, DEMCON has decided to develop a demonstrator on the buzz-wire game, see Figure 1.2. The setup of this demonstrator consists of the Niryo One robotic arm [13] with a camera placed on its wrist, which holds a metal loop, and a flexible wire. The loop and the wire are both connected to a power source and a buzzer. This is done in such a way that when the loop touches the wire, a closed electrical circuit is formed and the buzzer will sound. The controller of the robotic arm has camera images as input and the joint angles of the robot as output. The objective is to guide the loop along the wire without touching it.



(a) Robot arm at the starting position.

(b) Robot arm halfway along the wire.

Figure 1.2: Buzz-wire setup in the lab.

There are several other benefits to using the buzz-wire setup. First of all, by reducing the complexity of the task, the needed observation space is reduced and the agent is able to find reward quicker. Both phenomena lower the training time. Secondly, the demonstrator is suitable for exhibitions, which helps DEMCON to showcase the technology and bring it to the market. Lastly, it is possible to set intermediate target positions inbetween the start and final position to decrease the sparsity of the reward and therefore decrease training time. This might be needed if the reward is too sparse and the training time takes several days.

To develop an RL algorithm for the buzz-wire, it needs to train on accurate data. While training on the real setup will result in the most accurate training data, someone will need to supervise the system during the entire training. Next to that, the training time is longer than compared to training on a simulation or model as the simulated data can be acquired faster. Therefore, creating a simulator, hereafter referred to as a digital twin, for transfer learning will be very useful.

A 3D model is still quite complex and requires a long time to train. Therefore, this graduation project, focusses on creating a 2D discrete model of the buzz-wire problem. This reduces complexity further and gives more insight into the buzz-wire problem as a whole. Then the agent is trained on this simulated environment. In the 2D model, the number of inputs is reduced by replacing the camera images with several features such as the orientation of the loop with respect to the camera. These features are directly calculated from the environment, but in future work a model that converts the images to features can replace this step [14]. Furthermore, the code will be compartmentalized such that both the agent and environment can easily be changed. This helps facilitating the switch to a 3D environment or real world setup in the future and possible changes in learning algorithms.

On the 2D discrete model a Deep Q-learning algorithm (DQN) [15] is trained. This algorithm uses ANNs to approximate the optimal  $Q$ -function. A  $Q$ -function returns the expected discounted future reward when starting in a given state, taking a given action, and thereafter following a given policy. The given policy in the the optimal  $Q$ -function is an optimal policy. The approximation of the optimal  $Q$ -function is based on data. The agent receives an observation, selects an action using a given method (which balances exploration and exploitation), performs this action in the environment and receives a reward and a new observation. Together, this is stored in the replay buffer, from which samples are taken to update the  $Q$ -function. By randomly sampling data from the replay buffer, correlations within sequences of observations are removed and therefore the variance of the updates is reduced. This helps the policy to converge [15].

To summarize, the overall goal of the buzz-wire project is to develop a demonstrator, which shows the capability of combining computer vision, RL and robotics on a buzz-wire setup. To gain the competence needed to apply RL on the demonstrator, a 2D discrete model of the buzz-wire setup is created. This graduation project, a collaboration between the University of Technology Eindhoven (TU/e) and DEMCON, focusses on training an agent on this 2D model utilizing DQN. The policy of the trained agent is evaluated to check if the policy generalizes to wires it has not seen during training.

This thesis documents the project phase of the graduation project *Learning how to solve the Buzz-wire game with a robot arm*. The outline of the remainder of this thesis is as follows. In Chapter 2 the main challenges of the buzz-wire problem are identified and the main research questions are posed. Chapter 3 describes the general form of the DQN algorithm used to solve the buzz-wire problem. After that, a 2D model of the buzz-wire demonstrator is presented in Chapter 4. The results of training the agent on the 2D model are shown in Chapter 5. Lastly, conclusions are drawn in Chapter 6 and recommendations are done for future research.



## 2 Problem statement

As mentioned in Chapter 1, DEMCON is interested in the combination of RL, computer vision and robotics. To gain competence within DEMCON on RL and computer vision, a buzz-wire demonstrator is created. The setup is used for experimental research and consists of:

- a buzzer
- a camera
- a metal loop
- a plywood frame
- a power source
- a flexible conducting wire representing the buzz-wire
- a stiff conducting wire representing the target position
- LED lights
- the Niryo One robotic arm [13]

The metal loop is attached to the end effector of the robotic arm, which also has a camera fixed to its wrist. The images of this camera are the input for the controller. The controller of the robotic arm controls the position of the metal loop and should move such that the metal loop does not touch the flexible wire which goes through it. Both the metal loop and the flexible wire are attached to a power source via a buzzer and one of the LED lights, see the electrical scheme shown in Figure 2.1.

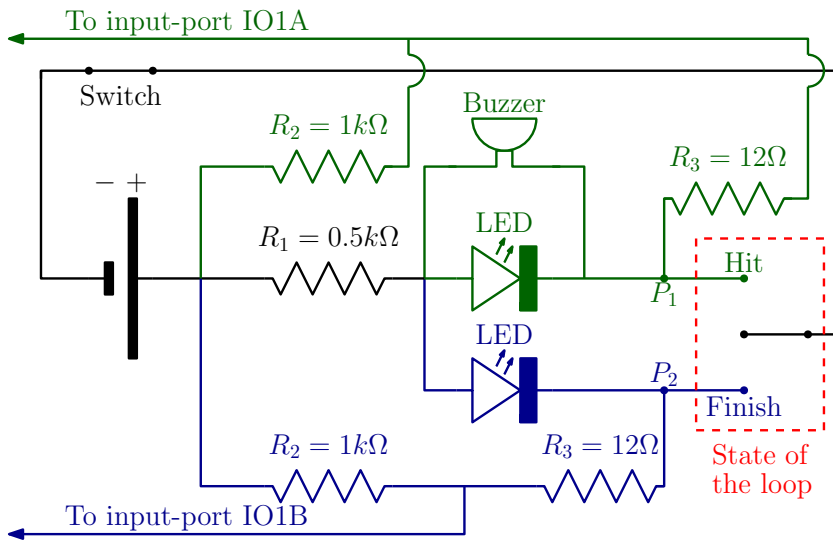


Figure 2.1: Electrical scheme of the setup.

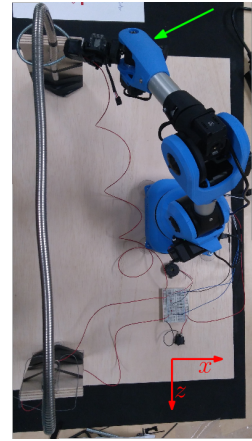


Figure 2.2: Top view of the buzz-wire setup in the lab. The camera is placed at the wrist (green arrow).

When the metal loop touches the flexible wire, the (green) circuit is closed. As a result the buzzer sounds, the LED is illuminated and the state of input port IOA1 changes (the voltage at  $P_1$  drops). This indicates that the wire has been hit. Every time the robot sees this signal, a negative reward is given to the agent. When the loop touches the stiff wire at the finish, the second (blue) circuit is closed and the other LED is illuminated. As a result input port IOA2 state changes as the voltage at  $P_2$  drops. This means that the loop has reached the finish and thus a positive reward is given to the agent. The camera is attached to the wrist of the robot arm. A photo of the setup is shown in Figure 2.2.

The goal of the buzz-wire demonstrator is to actuate the loop along the buzz-wire to complete its track based on the interactions with the environment, while avoiding to hit the wire. At each time step  $k \in \mathbb{N}$ , a camera image is given to the controller of the robot. Based on this observation, the robot needs to decide which action (a translational/rotational step) to take. If an action results in the loop hitting the buzz-wire, negative reward is given, and the robot moves to its previous position. If the loop completes the buzz-wire, a positive reward is given and the game (episode) ends. If the loop does not hit or complete the buzz-wire, the robot just moves to a new position and observes a new image. In Chapter 4 the observations, actions, rewards and their dynamics are explained in more detail. By maximizing the reward gathered over an episode, the goal of the buzz-wire demonstrator is achieved. Therefore, the goal



can be reformulated as:

**Goal:** *Find a motion policy for the agent that maximizes the cumulative reward through the proper selection of actions based on observations.*

To achieve this goal, the agent needs to create a mapping from current observations  $o \in \mathcal{O}$  to actions  $a \in \mathcal{A}$ , in other words a policy  $\pi$ . This policy should maximize the cumulative reward over each episode. However, reward is not dependent on observations, but on the positions of the robot and the wire, which are captured in the state  $s \in \mathcal{S}$ . This state and its dynamics are described in Chapter 4. Using this state, the optimal policy  $\pi_*$  (the policy which maximizes the cumulative reward over each episode) can be written as

$$\pi_* = \arg \max_{\pi'} \sum_{i=0}^{\infty} r(s_i, \pi[o_i(s_i)]), \quad \text{for all } s \in \mathcal{S}, \quad (2.1)$$

where  $s_i$  denotes the state at time  $i$ ,  $o_i$  the observation at time  $i$  (which depends on the state),  $r(s_i, a_i)$  the reward given the state  $s_i$  and the action  $a_i$  at time  $i$  and  $\pi : \mathcal{O} \rightarrow \mathcal{A}$  is the policy. We assume that there is a terminal state  $s_T$ , for which  $r(s_T, \cdot) \geq 0$ , making this a shortest path problem. When the goal state is achieved, a transition to  $s_T$  occurs. Because it is possible that the agent never reaches the finish, the simulation environment is given an upper bound of 400 time steps.

## 2.1 Research questions

There are several challenges in finding such a policy. These challenges are summarized in terms of research questions. First of all, a 2D discrete model of the buzz-wire problem is created to reduce the complexity of the buzz-wire problem and gain a better understanding of the problem. This leads to the first research question:

**Research question 1:** *What are the interactions of the 2D model with the agent (what is the action/ observation/ state of the environment/ dynamics of the environment/ reward)?*

Knowing the inputs and outputs of the environment, the agent can be designed. However, there are several known RL algorithms that can be implemented, such as Deep Q-learning [15] (DQN) or Soft Actor Critic (SAC) [16]. Next to that, parameters such as the learning rate need to be set. The resulting agent should be able to learn a policy that achieves the goal. Therefore, the learned policy should converge. Thus, the second research question is:

**Research question 2:** *How to define an RL architecture and choose its parameters such that the policy converges?*

Even if the policy of the agent converges, problems may still occur when the reward function is not defined properly. First of all, when there are few rewards and the reward function is sparse, it may take a long time before the policy converges. This can be solved by defining subtasks for which reward can be received or by using a heuristic such as the distance to the finish. However, this might lead to a bad policy that does not reach the finish, if the subtask or heuristic is not chosen adequately [17]. For example, using the distance to the finish as a heuristic can be problematic when the path of the wire moves away from the finish. Therefore, the third research question is as follows:

**Research question 3:** *How to choose the reward function to speed up learning?*

### 3 The learning algorithm

As stated in Chapter 2, the overall goal of this graduation project is to implement an RL algorithm on the 2D model described in Chapter 4, such that the cumulative reward of each episode is maximized for any random wire. Next to that, the following requirements are set for the agent:

- (i) The agent should be able to guide the loop to the finish within 60 seconds using the learned policy.
- (ii) The path of the buzz-wire is unknown to the agent and changes between episodes, which means the policy needs to be general enough that it can complete any path.
- (iii) The loop is not allowed to hit the buzz-wire.
- (iv) The agent has to make decisions based on camera images (or features derived from the images).
- (v) The training time of a new policy in the simulation environment should be within the time frame of a few days on a laptop.
- (vi) The agent should be able to work with interchangeable environments such as 2D models, 3D simulations, or the experimental setup.

Since one of the requirements is to guide the loop to the finish within 60 seconds, the optimal policy should try to make moves that lead to the finish as quickly as possible. This is achieved by maximizing the sum of the discounted future reward, for which we have to introduce the discount factor  $\gamma \in [0, 1]$ . The idea is that a reward received  $i$  time steps in the future is only worth  $\gamma^i$  times what it would be worth if it was received immediately [10]. So, when the agent tries to maximize the discounted reward, it is incentivized to take those actions that lead to the finish the quickest. The sum of the discounted future reward  $G_k$  at time step  $k \in \mathbb{N}$  is written as

$$G_k = \sum_{i=0}^{\infty} \gamma^i r_{k+i+1}, \quad (3.1)$$

where  $r_{k+i+1}$  is the reward returned by the environment  $i + 1$  time steps after time step  $k$ . As long as the discount factor is smaller than one, this sum will be bounded. When the discount factor is set to zero, then only the immediate reward is important.

#### 3.1 Deep Q-Learning (DQN)

The applied RL algorithm is a Deep Q-learning algorithm (DQN) [15]. This algorithm is an extension of tabular Q-learning algorithms [11] and applies to systems described by (Partially Observable) Markov Decision Processes. As mentioned in Chapter 3, the problem tackled here can be formulated as a POMDP. The goal of the algorithm is to find the mapping from states to actions that maximizes Equation (3.1). However, the problem is that many actions have no immediate reward, while they might be instrumental in getting to the reward eventually. As a result, it is hard to quantify how good any given action in a given state is. So instead, we consider the discounted future reward that can be expected by taking an action in a given state and then following a certain policy  $\pi$ . This leads to a so-called action-value function or  $Q$ -function, which is defined as [18]

$$Q_{\pi}(s, a) := \mathbb{E}_{\pi} \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right] \quad \text{for all } s \in \mathcal{S}, \text{ and for all } a \in \mathcal{A}. \quad (3.2)$$

Using Equation (3.2) different policies can be compared. A policy  $\pi$  is better than a policy  $\pi'$  if and only if the action-value function for all states and actions is larger or equal. There is always at least one policy better than or equal to all other policies. Optimal policies are denoted by  $\pi_*$ . There might be more than one optimal policy, however they all have the same action-value function  $Q_*$ , which is defined as [18]

$$Q_*(s, a) = \max_{\pi} Q_{\pi}(s, a) \quad \text{for all } s \in \mathcal{S}, \text{ and for all } a \in \mathcal{A}. \quad (3.3)$$

This optimal action-value function follows an important identity known as the Bellman equation [19]. The reasoning is that if the  $Q$ -value at the next time step is known for all possible actions  $a'$ , then the optimal strategy is to select the action that maximizes the expected value of  $r + \gamma Q_*(s', a')$ .

$$Q_*(s, a) = \mathbb{E}_{\pi} \left[ r + \gamma \cdot \max_{a'} Q_*(s', a') \mid s, a \right]. \quad (3.4)$$

In Q-learning algorithms an iterative update based on Equation (3.4) is used [15]. This results in an estimated action-value function that converges to the optimal action-value function under the assumption

that the policy interacting with the environment has a nonzero probability of selecting all actions. The estimated action-value function at iteration  $i$  is written as

$$Q_{i+1}(s_k, a_k) = Q_i(s_k, a_k) + \alpha \left( r_{k+1} + \gamma \cdot \max_a Q_i(s_{k+1}, a) - Q_i(s_k, a_k) \right), \quad (3.5)$$

where  $\alpha \in \mathbb{R}_{\geq 0}$  is the learning rate. When  $i$  goes to  $\infty$ ,  $Q_i$  converges to  $Q_*$ . The difference between normal Q-learning and DQN is that in DQN an Artificial Neural Network, also called a  $Q$ -network, is used to approximate the optimal action-value function  $Q_i$ . This  $Q$ -network,  $Q(s, a; \mathbf{W}_i)$  with  $\mathbf{W}_i$  as the weights of the network, is trained by adjusting the weights at each iteration  $i$  to such that the mean squared error of the Bellman equation, Equation (3.5) with  $Q_i$  replaced by  $Q(s, a; \mathbf{W}_i)$ , is reduced. This results in the following loss function for the  $Q$ -network

$$L_i(\mathbf{W}_i) = \left( r + \gamma \max_{a'} Q(o', a'; \mathbf{W}_i^-) - Q(o, a; \mathbf{W}_i) \right)^2, \quad (3.6)$$

where  $\mathbf{W}_i^-$  are the weights of  $Q$ -network for some previous iteration. Using a  $Q$ -network can result in the policy not converging. This can be caused by the correlations in the sequence of observations. Next to that, changes in the action-value function effect the policy and therefore the data distribution can shift drastically. This is resolved by implementing a replay buffer for experience replay [15]. Storing sequences in a replay buffer and then sampling data randomly from this buffer, has three advantages. First, experiences sampled from the sequences are potentially used in many updates, which allows for greater data efficiency. Secondly, the correlation between consecutive samples is broken reducing the variance of the updates. Third, the data of the next samples is gathered using the current policy. If the maximizing action changes, the data distribution also switches. This can lead to feedback loops where the parameters of the network  $\mathbf{W}_i$  get stuck in a poor local minimum or even diverge [20]. By using experience replay the policy changes slower resulting in a smoother change of the data distribution and avoiding oscillations or divergence. As a result the policy converges.

### 3.2 Data collection

The replay buffer needs to be filled with experience tuples. An experience tuple consists of an observation, an action, a reward and the next observation,  $e_k = (o_k, a_k, r_k, o_{k+1})$ . To gather experiences, the agent selects and executes actions in the environment. Various methods to select an action are possible. The first option is to select the action that maximizes the  $Q$ -value. This action is called a *greedy* action and is written as [10]

$$a = \arg \max_{a'} Q_\pi(s, a'). \quad (3.7)$$

By taking this greedy action each time, the current knowledge is exploited to maximize the immediate reward. No time is spent sampling actions, which seem to be inferior, to see if they might be better. An alternative option is to take a random action with a probability  $\epsilon \in [0, 1]$  and otherwise behaves greedily. This action selection method is called the  $\epsilon$ -greedy method and results in

$$a = \begin{cases} \arg \max_{a'} Q_\pi(a') & \text{with probability } 1 - \epsilon + \frac{\epsilon}{\#\mathcal{A}} \\ \text{any other action } a' & \text{with probability } \frac{\epsilon}{\#\mathcal{A}} \end{cases} \quad (3.8)$$

where  $\#\mathcal{A}$  is the number of possible actions. Using the  $\epsilon$ -greedy method, the agent will sometimes take a random action and explore. However, it will also take greedy actions and exploit its knowledge, which leads to reward. Having a good balance between exploration and exploitation is important for the convergence rate of the learning algorithm. Especially at the start, the agent should explore more. As a relatively good policy is found, exploration becomes less important. Therefore, the DQN algorithm utilizes the  $\epsilon$ -method where the  $\epsilon$  is decaying.

## 4 Modelling the 2D environment

As mentioned in Chapter 1, this graduation project focusses on training an RL algorithm for the 2 dimensional model of this buzz-wire setup. RL algorithms use the framework of Markov Decision Processes (MDPs) to describe the relation between the agent and the environment [10]. However, the controller of the robot arm receives camera images showing only a part of the wire. Thus, the environment is only partially observable and for that reason the framework of a Partially Observable Markov Decision Process (POMDP), see Figure 4.1, applies.

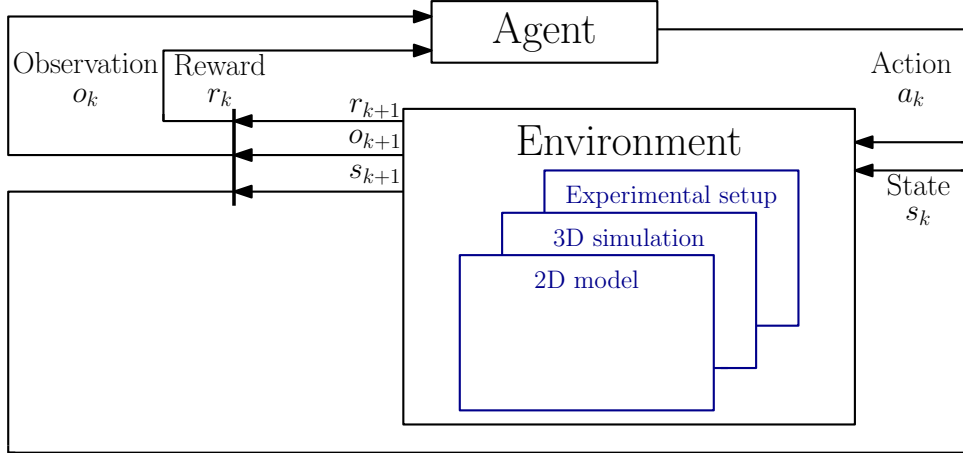


Figure 4.1: *The agent-environment interaction in a Partially Observable Markov Decision Process [10]. The vertical line after the environment represents the end of the time step.*

The general framework of a POMDP is as follows. At each time step  $k \in \mathbb{N}$  the agent receives an observation  $o_k \in \mathcal{O}$  of the environment. Based on this observation, the agent has to select an action  $a_k \in \mathcal{A}$ . The action causes the environment to transition from the state  $s_k \in \mathcal{S}$  to a new state  $s_{k+1}$  with probability  $p(s_{k+1}|s_k, a_k)$  (in our case this will be a delta function). The new state results in a reward  $r_k \in \mathcal{R}$  (based on whether the wire is hit/ end of the wire is reached) and a new observation  $o_{k+1}$ . In the next sections the state, action, transition, observation and reward of the 2D environment are explained.

### 4.1 State

In a 2D model the flexible buzz-wire needs to be contained within a single plane. The translation and rotation of the loop are restricted to this plane as well. Therefore, the position and the orientation of the loop at time step  $k \in \mathbb{N}$  can be described with 2 coordinates and 1 angle  $[x_{loop,k}, y_{loop,k}, \theta_{loop,k}]^\top$ . Knowing the position and rotation of the loop would be enough to determine if the loop hits the wire, if the shape of the loop and the shape and position of the wire are given. However, the shape of the wire is changed between games (episodes  $e \in \mathbb{N}$ ). Thus, the state should contain information about the wire shape. Therefore,  $x$  and  $y$  coordinates of  $n_{wire}$  points along the wire are included in the state resulting in

$$\mathbf{s}_k = [x_{loop,k} \quad y_{loop,k} \quad \theta_{loop,k} \quad \mathbf{x}_{wire,e} \quad \mathbf{y}_{wire,e}]^\top, \quad (4.1)$$

where  $\mathbf{x}_{wire,e}$  and  $\mathbf{y}_{wire,e}$  are the global coordinates of  $n_{wire}$  points along the wire in the order in which they are encountered. Thus, the points start from the starting position increase towards the target position. The target position is defined as the first point on the wire that is at least a distance  $d_{goal}$  removed from the start (during the training in Chapter 5  $d_{goal}$  was set to 100). The state  $\mathbf{s}_k$  is chosen for three reasons. First of all, the dynamics of this state are relatively simple. Secondly, this state contains enough information to calculate the reward (and observation). Lastly, using this state the 2D environment can be visualized, as shown in Figure 4.2.

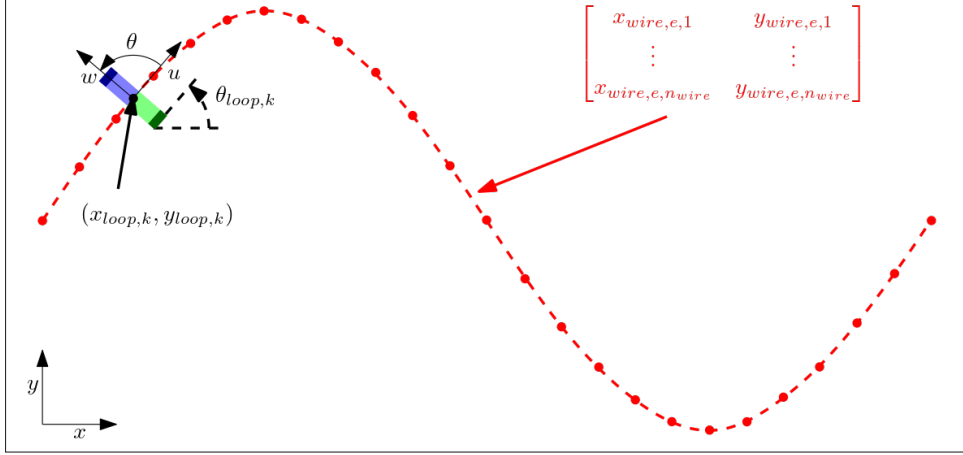


Figure 4.2: The state of the loop is described by the global coordinates ( $x$  and  $y$ ) of the center of the loop (black point) and the angle  $\theta$ . The layout of the wire is given by the global coordinates of the points on the wire (red points). A local coordinate system ( $u$  and  $w$ ) is attached to the loop.

## 4.2 Action

The robot arm can move in three ways: translate along the longitudinal axis of the loop  $u$ , translate along the lateral axis of the loop  $w$  or rotate the loop around its centre  $\theta$ . The positive directions for each move are shown in Figure 4.3. With the assumption that the speed is low, the moves are discretized to  $+1$ ,  $0$  or  $-1$  times the step-size. The different moves can also be combined resulting in 27 possible actions. The velocity is normalized, to prevent the combinations of translations from going faster than any single translation.

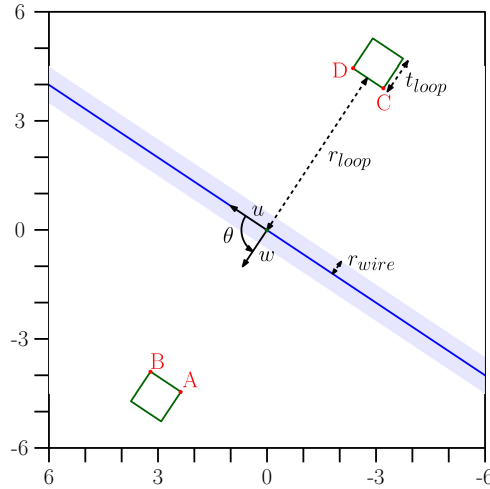


Figure 4.3: The direction of the actions, given a orientation of the loop.

## 4.3 Transition

Given an action, the new state  $s_{k+1}$  can be calculated. The layout of the wire does not change during an episode but is resampled from a subset of wires (see Appendix B) between episodes. Thus, the actions only change the position and rotation of the loop. This is done by adding the move, after converting it to the global coordinate system  $(x, y)$ , to the previous state. This results in

$$\begin{aligned}
 x_{loop,k+1} &= x_{loop,k} + \cos(\theta_{loop,k}) \cdot \Delta u_k - \sin(\theta_{loop,k}) \cdot \Delta w_k \\
 y_{loop,k+1} &= y_{loop,k} + \sin(\theta_{loop,k}) \cdot \Delta u_k + \cos(\theta_{loop,k}) \cdot \Delta w_k, \\
 \theta_{loop,k+1} &= \theta_{loop,k} + \Delta \theta_k
 \end{aligned} \tag{4.2}$$

where  $\Delta u_k$  is the longitudinal translation,  $\Delta w_k$  is the lateral translation and  $\theta_{step,k}$  is the rotation. However, when the loop hits the wire in this new position, then the loop is returned to the position before the action. How to calculate whether the loop hits the wire, is explained in the next section.

#### 4.4 Reward

In the buzz-wire setup the diameter and thickness of the loop,  $d_{loop}$  and  $t_{loop}$ , respectively, the thickness of the wire  $r_{wire}$  and the distance to the target position  $d_{goal}$  are fixed. Using these parameters and the current state, it is determined whether the wire is hit or not. Given the position and orientation of the loop the global coordinates of the inside corners of the loop  $A, B, C, D$  can be calculated. Then the distances from these points to the discretized points on the wire are calculated. This can be expressed as

$$\begin{aligned} \mathbf{d}_{A,k} &= \sqrt{(x_{loop,k} + d_{loop}/2 - \mathbf{x}_{wire,e})^2 + (y_{loop,k} - t_{loop}/2 - \mathbf{y}_{wire,e})^2} \\ \mathbf{d}_{B,k} &= \sqrt{(x_{loop,k} + d_{loop}/2 - \mathbf{x}_{wire,e})^2 + (y_{loop,k} + t_{loop}/2 - \mathbf{y}_{wire,e})^2} \\ \mathbf{d}_{C,k} &= \sqrt{(x_{loop,k} - d_{loop}/2 - \mathbf{x}_{wire,e})^2 + (y_{loop,k} + t_{loop}/2 - \mathbf{y}_{wire,e})^2} \\ \mathbf{d}_{D,k} &= \sqrt{(x_{loop,k} - d_{loop}/2 - \mathbf{x}_{wire,e})^2 + (y_{loop,k} - t_{loop}/2 - \mathbf{y}_{wire,e})^2} \end{aligned} \quad (4.3)$$

Then the minimum of these distances is taken to get the minimal distance between the wire and the loop  $d_{wire,k}$ . If this distance is smaller than the radius of the wire  $r_{wire}$ , then the loop is hitting the wire and negative reward is given to the agent. This is the red case in Equation (4.6).

To determine whether the loop has reached the finish, the point on the wire that is the closest to the center of the loop is found. The index of this point at time step  $k$  is calculated as

$$i_k = \operatorname{argmin}_{j \in \mathbb{N}} \left( \sqrt{(x_{loop,k} - x_{wire,e,j})^2 + (y_{loop,k} - y_{wire,e,j})^2} \right). \quad (4.4)$$

As mentioned in Section 4.1, the index of the points along the wire increase from the start towards the finish. The finish is defined as the first point along the wire for which the distance to the start (at coordinate 0, 0) is greater than or equal to  $d_{goal}$ . This results in

$$i_{goal,e} := \left\{ \min(j) \mid \sqrt{(x_{wire,e,j})^2 + (y_{wire,e,j})^2} \geq d_{goal} \right\} \quad (4.5)$$

If the index of the loop  $i_k$  at time  $k$  is greater than or equal to the index of the finish  $i_{goal,e}$  at the current episode  $e$ , then the loop has reached the finish. This means a positive reward is given to the agent, as shown by the green part in Equation (4.6), and the episode ends.

Next to the rewards for reaching the finish and hitting the wire, a small reward can be given for progressing along the wire. The progress is calculated by comparing the current and previous projection of the loop on the wire. This is the blue part of Equation (4.6). Each reward ( $r_{progress}$ ,  $r_{finish}$  or  $r_{hit}$ ) can be turned off by setting its value to 0. The total reward function is written as

$$r_k = \frac{r_{progress}}{n_{wire}}(i_k - i_{k-1}) + \begin{cases} r_{hit} & \text{if } d_{wire,k} \leq r_{wire} \\ r_{finish} & \text{if } d_{wire,k} > r_{wire} \text{ and } i_k \geq i_{goal,e} \\ 0, & \text{else} \end{cases} \quad (4.6)$$

#### 4.5 Observation

As mentioned in Chapter 1, the controller in the real system observes images from the camera and the joint angles. To lower the amount of inputs (every pixel is an input) and reduce computation time for the 2D model, the observation is reduced to features, which can be extracted from the images. The 4 features included are shown in Figure 4.4 and consist of

- (a) 2 features ( $\mathbf{d}_{BD}, \mathbf{d}_{AC}$ ) that describe the distance between the intersection of the wire with the front (in interval BD)  $I_1$  and back (in interval AC)  $I_2$  of the loop and the left side of the loop (points A and B). These features are shown in Figure 4.4a.
- (b) 1 feature which describes the orientation of the loop  $\theta$  with respect to the global coordinate system, see Figure 4.4b.
- (c) 1 feature describing the relative angle between the loop and the wire  $\phi$ . This is parametrized by the crossing point  $I_4$  of the wire with a half-circle in front of the loop as shown in Figure 4.4c.

The representation of the features are one-hot encoded, meaning the feature space is discretized. This is done to avoid scaling problems (related to activation functions in the neural network of the agent) as well as recurrence problems (for example  $-\pi$  is close to  $0.99\pi$ ).

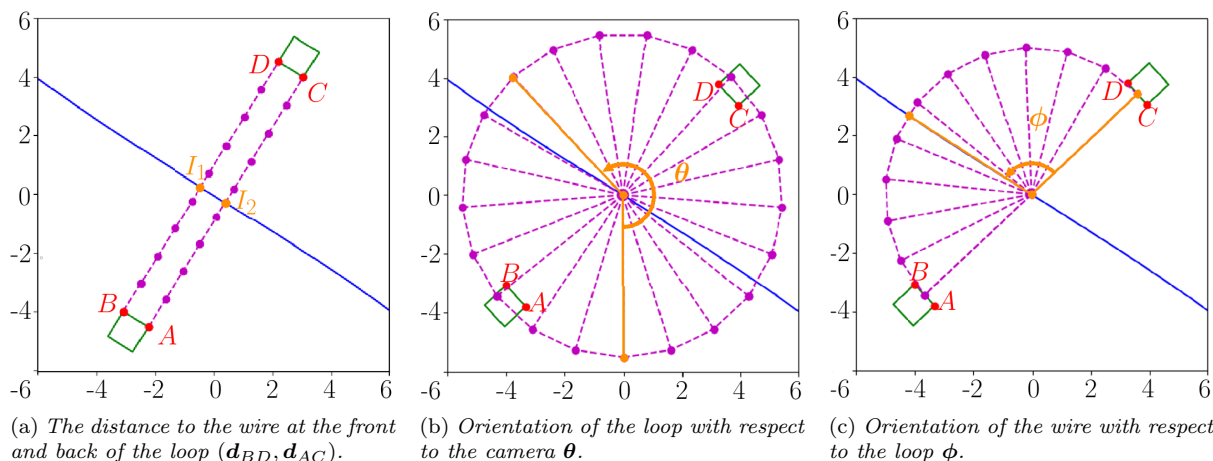


Figure 4.4: The features which form the observation of the 2D model.

To summarize, the agent (the robot arm) interacts with the environment (in this case the 2D model), via a sequence of actions, observations and rewards. At each time step  $k \in \mathbb{N}$  the agent selects an action  $a_k$  from the set of possible actions  $\mathcal{A} = 0, \dots, 26$ . The actions is performed in the 2D model, modifies its internal state  $s_k \rightarrow s_{k+1}$ . This change in the environment is observed via the features  $(\mathbf{d}_{BD}, \mathbf{d}_{AC}, \theta, \phi)$  in the observation  $o_{k+1}$ . In addition the agent receives a reward  $r_k$ . The goal of the agent is to select the actions such that the future discounted return, which is calculated by Equation (3.1), is maximized.

## 5 Performance and validation of the trained policies

Now that both the agent and the environment are defined, policies can be trained and their performance can be evaluated. The first policies did not perform well on the environment, due to several bugs and additional problems explained in Appendix E. These problems were mainly caused by how the actions were defined. After this was solved, two policies were trained. Policy A was trained on an environment with continuous rewards and policy B was trained on an environment without continuous reward. By comparing the performance of these policies, the impact of the continuous reward has been analysed. The performance and validation of both policy A and B can be found in Section 5.1 and Section 5.2 respectively. The generalization of both policies was also tested by applying them to a different wire, see Section 5.3. Lastly, policy B was implemented on the buzz-wire setup. The results of this simulation are shown in Section 5.4.

In the environment used to train both policy A and B, the loop moves in local coordinates, see Figure 4.2. The environment sends the ground truth features as the observations. Training on the ground truth features is useful because if the agent cannot find a policy based on the ground truth, it will not be able to learn a policy on an estimation of these features. In that case, the features themselves should be changed. More settings of the environment on which policy A and B were trained, can be found in Appendix D. There is only one difference between the environment and that is the reward for making progress  $r_{progress}$ . While training policy A, this reward was set to 2000. During the training of policy B, it was set to 0.

### 5.1 Training policy A on an environment with continuous reward

In Figure 5.1 the performance of policy A during training is plotted. The performance of the policy is evaluated with four categories: the average length of an episode (Figure 5.1.A), the average reward per episode (Figure 5.1.B), the maximum reward achieved in one of the episodes (Figure 5.1.C) and the minimum reward achieved in one of the episodes (Figure 5.1.D). The average length of an episode should be minimized, since the policy should solve the wire as quickly as possible. On the other hand, the average reward per episode, the maximum reward and the minimum reward should all be maximized.

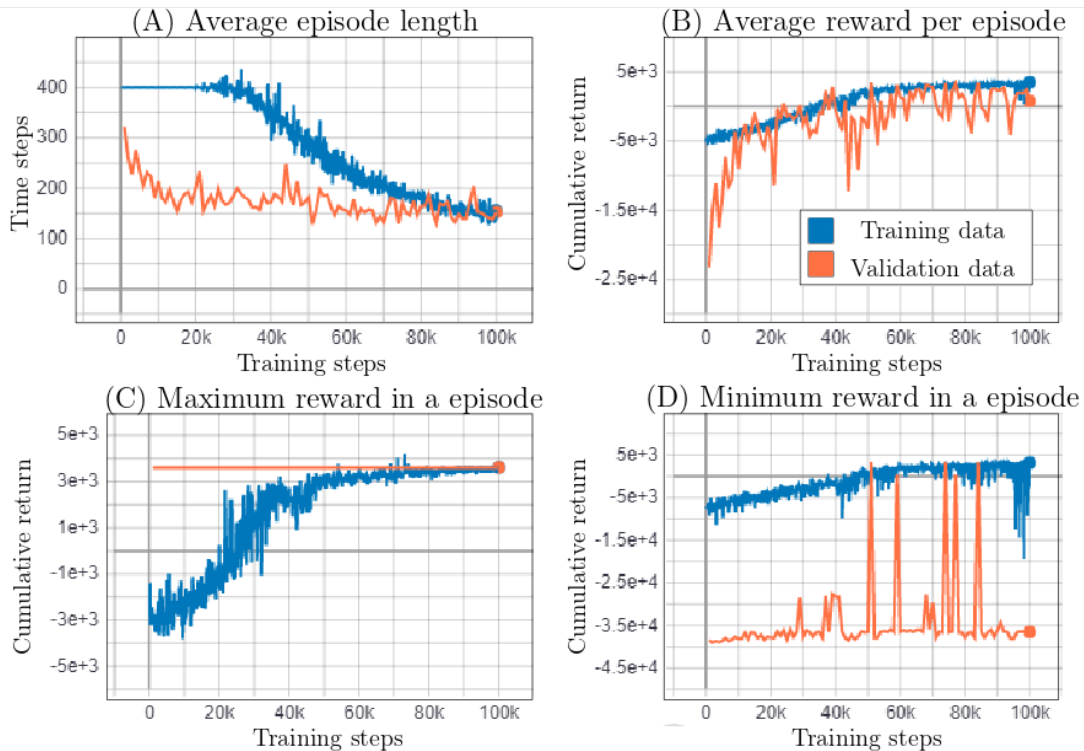


Figure 5.1: *The performance of policy A during training. In each plot two lines are shown: the training data and the validation data. During training actions are chosen using the  $\epsilon$ -greedy method and the wires are random. During validation the greedy action is always taken and the set of wires is fixed.*



As shown by the maximum reward (Figure 5.1.C), the agent immediately learns to reach the finish for a few validation wires (in orange). This is because going in the right direction is already rewarded. So, if a wire has no corners and the agent knows it has to move forward for reward, it can already reach the finish. Since the agent quickly learns the direction to move in, the average episode length (Figure 5.1.A) of the validation data lowers quickly and seems to be near its minimum after 37000 training steps. In a similar way, the average reward per episode (Figure 5.1.B) has increased to 2255.

The validation data shows that the agent learns quickly, but the average reward per episode (Figure 5.1.B) during training lags behind. This is explained by the fact that the  $\epsilon$ -greedy action selection method is used instead of the greedy method that is used for the validation set. It also explains why the minimum cumulative reward (Figure 5.1.D) of the validation data is much lower than the minimum reward of the training data. After the wire is hit, the policy during the validation will repeat this move 400 time steps as there is no exploration. The training policy on the other hand will explore other moves, due to the  $\epsilon$ -greedy method. These might not hit the wire and thus the minimum reward will not decrease as much. However, the training policy will become greedier each epoch as the  $\epsilon$  gets lowered. For this reason, the lowest minimum reward is found at the end of the training session.

After 65000 training steps the average reward of the validation (shown in orange in Figure 5.1.B) is still not stabilized at the maximum reward. Instead it fluctuates below it. This can be explained by the projection of the loop on the wire near corners. To be able to make the corner, there needs to be enough space between the loop and the wire at the outside of the corner to be able to turn around  $\theta$ . In Figure 5.2a there is not enough space to make the corner so in this case the loop should be moved in the negative  $w$  direction. However, doing so leads to a negative continuous reward as the projection on the wire  $P$  is moved backwards, see Figure 5.2b.

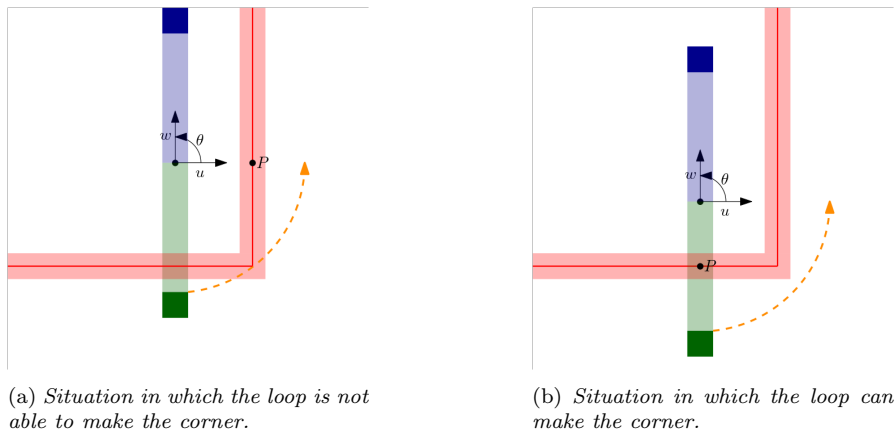


Figure 5.2: The projection of the loop on the wire at a corner

To summarize, while the reward for making progress helps the agent to quickly figure out which direction to move in, it is counterproductive in the long term as it prevents the agent from learning the sequence of moves needed to turn through a 90 degree corner. This is caused by actions that, while necessary, move the projection back along the wire. As a result the agent receives immediate negative reward, and learns to not do these moves anymore.

## 5.2 Training policy B on an environment without continuous reward

The performance of policy B during training is plotted in Figure 5.3. For the first 40000 training steps the agent barely knows what moves to make. The reward collected during training is only negative, since the maximum reward is negative (blue line in Figure 5.3.C). As a result the agent only learns how to avoid the wire during these first 45000 training steps, but not how to reach the finish. This is shown by the fact that both the maximum (orange line in Figure 5.3.C) and minimum return (orange line in Figure 5.3.D) of the validation both go to 0. From the moment the loop has reached the finish once in the training data, the agent quickly learns what moves it should take. The average episode length (Figure 5.3.A) converges to around 150 and the average return (Figure 5.3.B) to 2000. This means that the learned policy has maximized the average reward as the maximum reward (Figure 5.3.C) is also 2000.

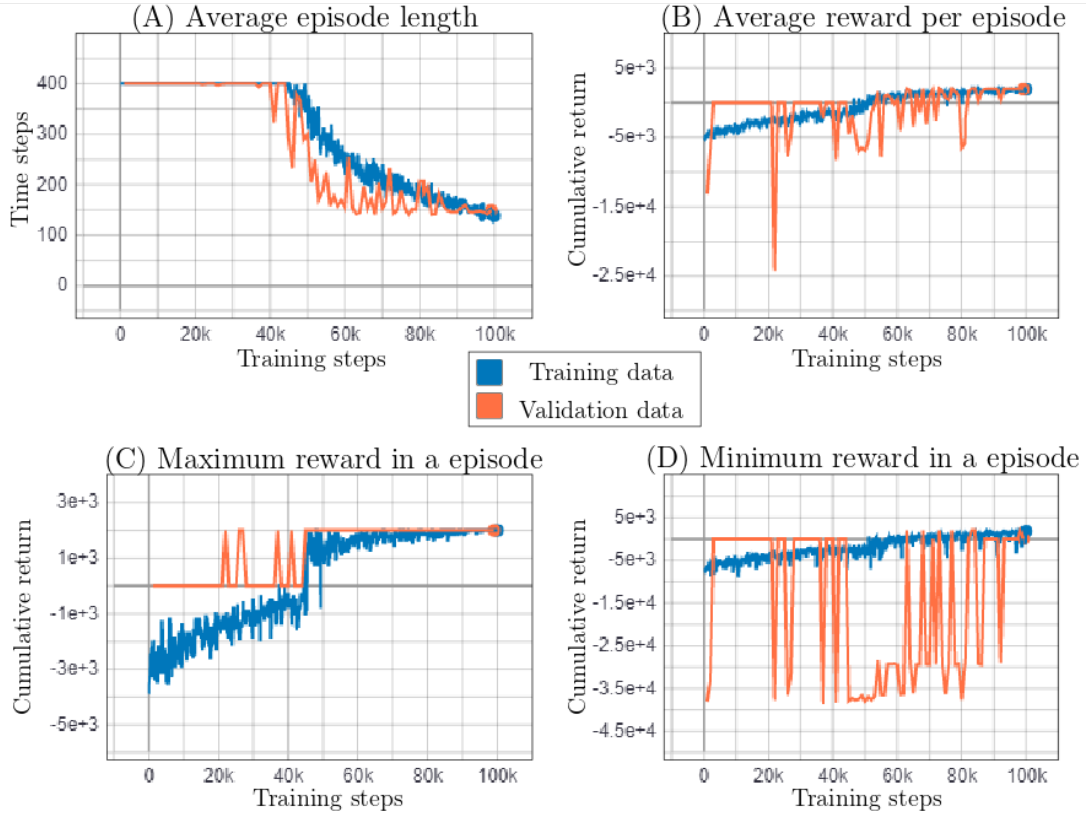


Figure 5.3: *The performance of policy B during training. In each plot two lines are shown: the training data and the validation data. During training actions are chosen using the  $\epsilon$ -greedy method and the wires are random. During validation the greedy action is always taken and the set of wires is fixed.*

The policies learned in training session 1 and 2 are shown in Figure 5.4 and Figure 5.5 respectively. Both figures show that there is no relation between the agent angle feature  $\theta$  and the actions. This means that this feature is obsolete as the policy does not rely on it. Furthermore, these figures show that the move forward in the  $u$  direction and turning rely mostly on feature  $\phi$ . The difference between the policies can be found in how the lateral move  $w$  responds to  $\phi$ . The policy trained on the continuous reward has a clear preference for lateral movements. This can be explained by using Figure 5.2b again. Given this situation the agent get positive continuous reward when moving in the positive  $w$  direction and negative reward for the other direction. The policy of the second training session does not have this incentive and therefore is not orientated as such.

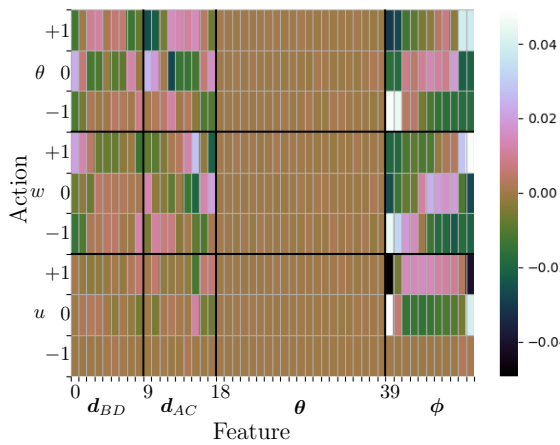


Figure 5.4: *Covariance between the features and actions under the policy A.*

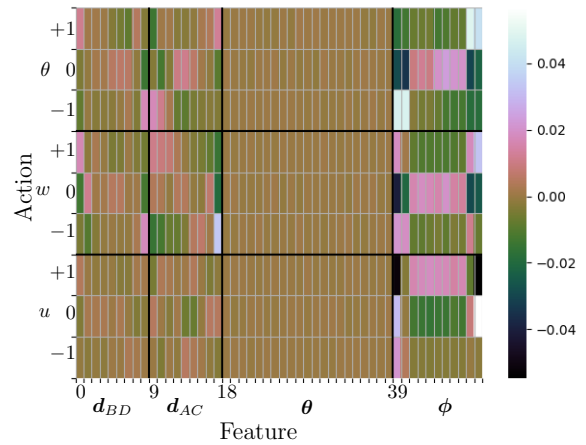


Figure 5.5: *Covariance between the features and actions under the policy B.*

### 5.3 Performance on wire with four corners to the left

The performance of both policies has been evaluated on a wire with four consecutive 90 degree corners to the left, to see whether the policies are able to generalize to a different wire as well. This wire with four corners ( $C_1, C_2, C_3, C_4$ ), shown in Figure 5.6, is not part of the training set. On the other hand, the main parts of the wire, such as the 90 degree corners and the straight parts, are still there.

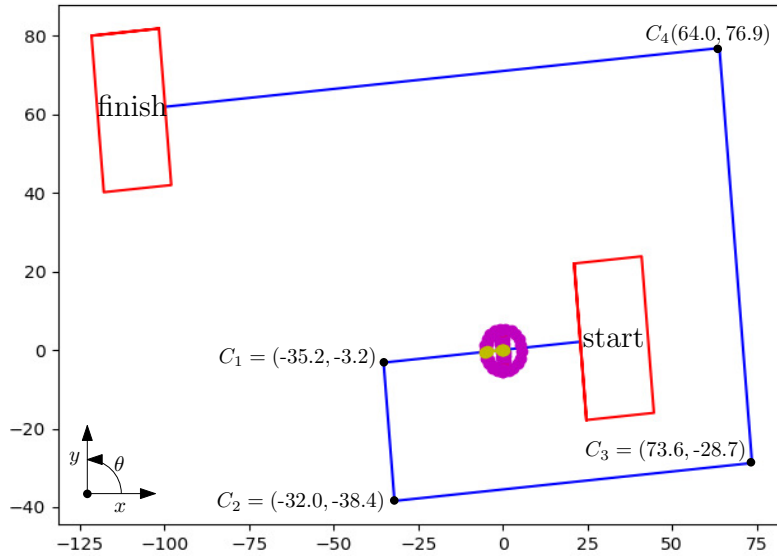


Figure 5.6: Wire with four 90 degree corners in the same direction.

The policy of training session 1 is not able to traverse this wire, hitting the wire at the first corner to the left, as shown by Figure 5.7. The policy of the second training session performs much better. This policy is able to make all corners (final corner is shown in Figure 5.8) and reach the finish. This indicates that the policy of the second training session generalizes much better than the policy of the first training session. This is most likely caused by the continuous reward discouraging actions needed to make the corner, as was illustrated by Figure 5.2a and Figure 5.2b.

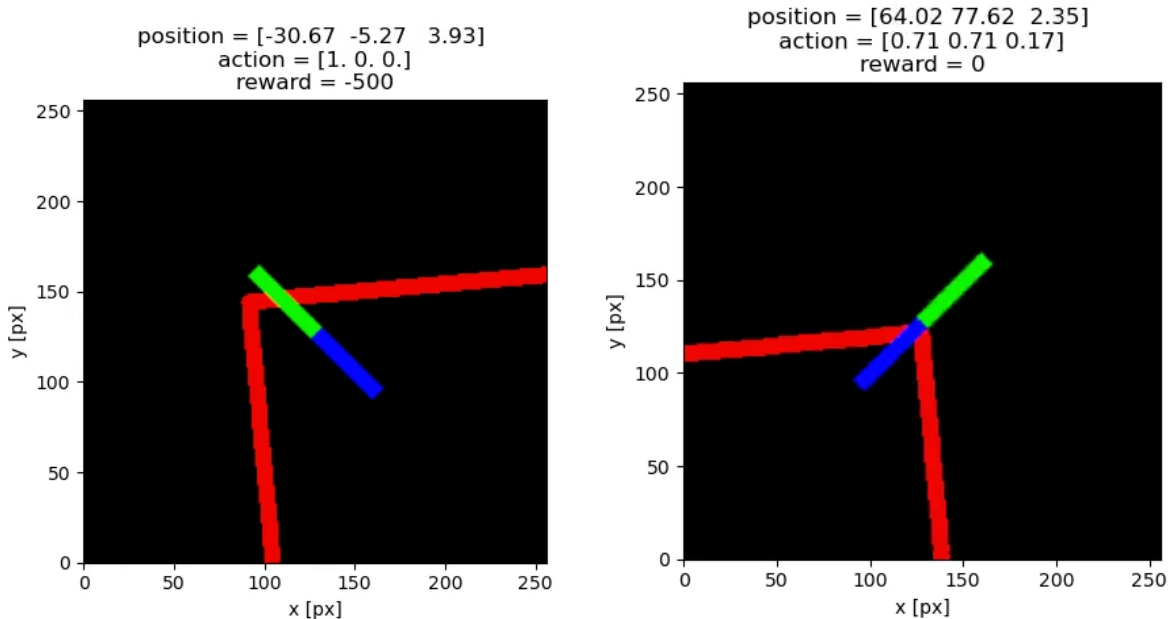


Figure 5.7: Policy A getting stuck at the first corner. Figure 5.8: Policy B turning through the final corner.

To summarize, the results show that policy B performs better than the policy A. Policy B is able to complete almost all the wires in the validation set as the average reward per episode (orange line in Figure 5.3.B) is almost equal to the maximum reward gained in a episode. On top of that, the policy is able to complete the wire with four corners. Based on these results, policy B seems to be able to complete any wire that is smooth or has 90 degree corners.

#### 5.4 Performance on the real setup

Since policy B performed well on the 2D model, it is tested on the real setup as well. The policy requires features as inputs. However, there was no model to obtain the features from camera images. Instead, a wire is hard-coded into the agent, which also measures the position of the end effector of the robot. Using these measurements and the wire, the features are calculated, see Figure 5.9.

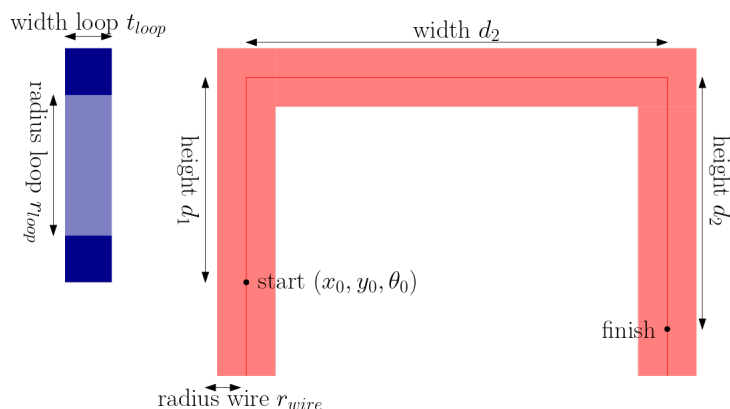


Figure 5.9: The hard coded wire and the dimensions of the loop used to calculate the observation.

Using the calculated features, the policy decides which action to takes. Then, the robot takes this action and moves to a new position. This new position is send back to the agent to calculate new features and decide on the next action. The robot keeps doing this cycle until it has done 100 steps, hit the wire or reached the finish.

With policy B the loop was actuated through the first corner, along the straight part and hit the wire at the second corner, as shown Figure 5.10. This is partly due to the calculation of the observation, which assumes the wire is shaped according to Figure 5.9. However, the real shape of the wire does not match the hard-coded wire exactly, see Figure 5.11, leading to the observations being slightly off. As a result the agent thinks there is room to move in  $x$ -direction, while in reality there is not.

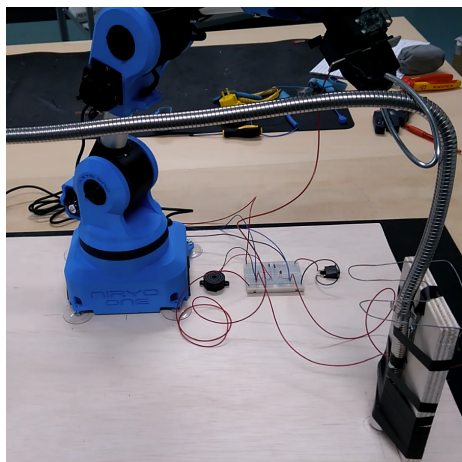


Figure 5.10: The loop hitting the wire at the second corner.

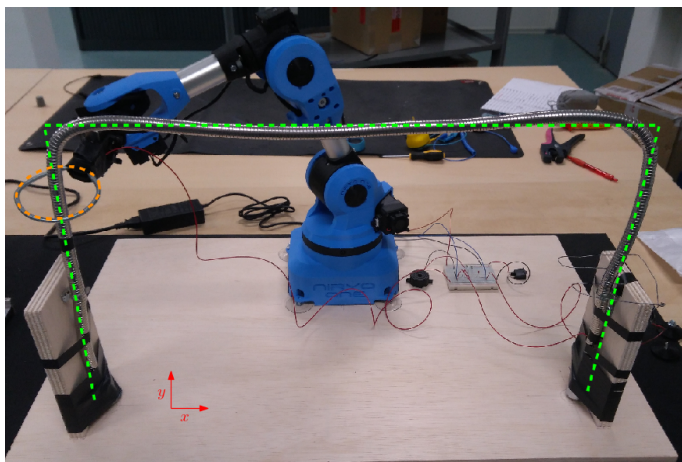


Figure 5.11: The shape of the hard coded wire (drawn in green) versus the actual wire.

Another reason for hitting the wire is the drift of the robot. During the movement of the robot, the end effector drifts in several directions (such as the yaw and  $z$  direction, see Figure 5.13 and Figure 5.12 respectively). This is even worse when the robot is operating near the limits, which is the case during the second turn. Due to these drifts, the loop hits the wire before the robot expects it.

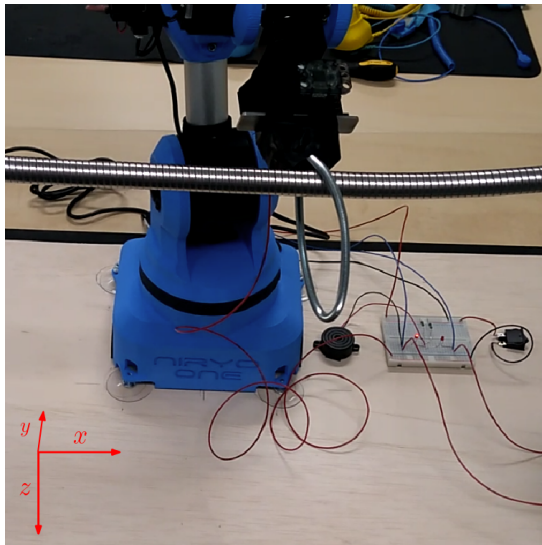


Figure 5.12: The loop has drifted in  $z$  direction, resulting in hitting the wire.

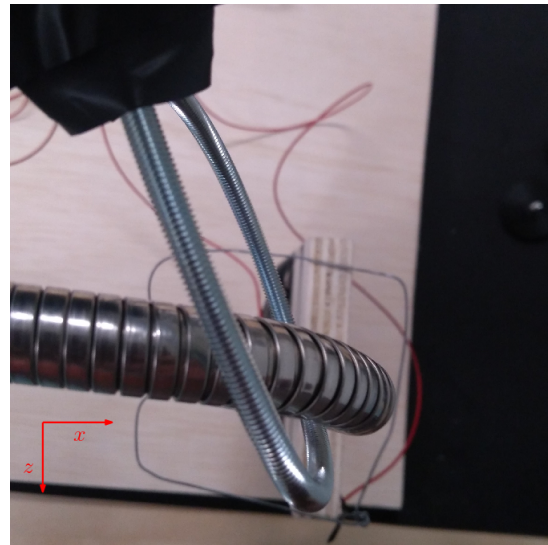


Figure 5.13: The loop has rotated around the  $y$  direction due to drift, leading to hitting the wire.

To test how much the arm would drift, hysteresis experiments have been performed. In Figure 5.14 the position and orientation of the loop are plotted. At each time step the robot arm receives the command to change the  $x$  position according to the reference signal shown in Figure 5.14a and keep the other dimensions equal to their previously measured values. During the experiment the  $x$  and  $y$  positions both drift, see Figure 5.14a. The height  $y$  slowly decreases over time and the  $x$  position lags when going in the negative direction. As a result the position of the loop is slightly different than expected. However, the biggest difference can be seen in the rotations. According to page 18 the yaw after 20 steps is almost 0.4 radians higher. This drift matches the drift seen in Figure 5.13.

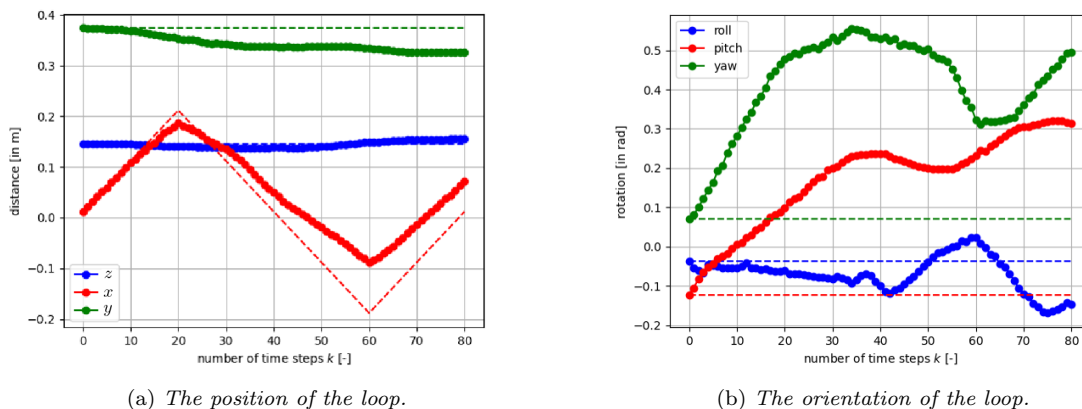
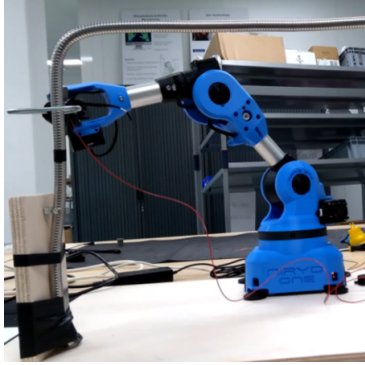
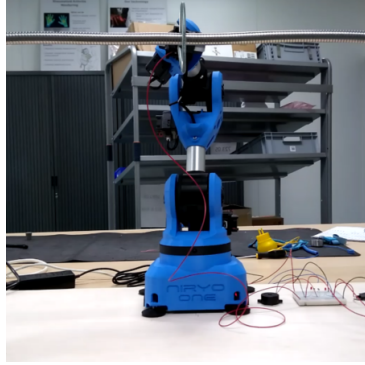


Figure 5.14: The position of the loop while taking 20 steps in the positive  $x$  direction, followed by 40 steps in the negative  $x$  direction and then another 20 steps in the positive  $x$  direction. The dotted lines are the reference signals.

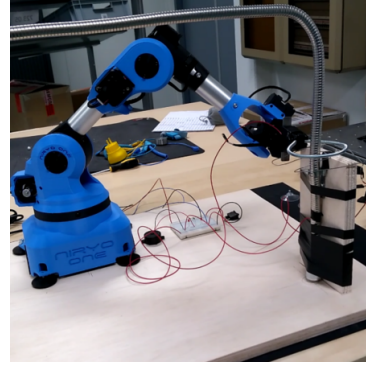
After the hysteresis experiment, policy B was tested again. To prevent the end effector from drifting in the  $z$  position, pitch and yaw, they were all fixed to their initial value. By controlling the drift in this way, policy B was able to successfully complete the buzz-wire, see Figure 5.15.



(a) *Starting position*



(b) *Intermediate position*



(c) *Final position*

Figure 5.15: *The robot arm in different positions during a successful run of policy B on the real-life setup.*



## 6 Conclusions and recommendations

### 6.1 Conclusions

The objective of this project was to implement an RL algorithm on the 2D model described in Chapter 4, such that a motion policy is generated which reaches the finish without having the loop hit the wire. Policy B, which is trained on an environment without continuous reward, is able to do this since it reaches the maximum reward of 2000 for nearly every validation episode. In only one case the finish is not reached, but the wire is not hit either. The policy also works on a wire with four consecutive left corners, indicating the policy is generally applicable and not restricted to the specific wires it has trained on. Next to that, policy B has been tested on the actual setup and was able to complete it once. This shows that policy B is robust enough to handle the noise coming from the sensors in the real-life setup. Given these results, the objective of this project has been achieved for the 2D model. Policy B is able to maximize the cumulative reward through the proper selection of actions based on the observations. The features in the observation only give local information, while the reward (of reaching the finish) is global. Notably, the learned policy is still able to use the local features to find the global reward.

In contrast to policy B, policy A only achieves the objective for wires that are shaped similarly to the wires it has trained on. The average cumulative reward of this policy is significantly lower than the maximum reward that can be achieved in a single episode. Moreover, for some wires the agent is repeatedly hitting the wire resulting in the minimum cumulative reward of around -35000 for some episodes. Since the only difference between the training of policy A and B is the presence or absence of the continuous reward, it can be concluded that the continuous reward is hindering the long term learning process. As a result, the average cumulative reward likely never reaches its maximum, even after training on 100000 mini-batches.

The analysis between of policy A and B leads to several insights. First of all, the loop angle feature seems to be obsolete. After training the agent, the actions are not correlated with this feature at all. This makes sense as the actual movement in  $u$  and  $w$  direction already rotates with the loop. Furthermore, the actions are mostly dependent on the relative angle  $\phi$  between the wire and the loop. In both policies the plane of the loop is rotated in such a way that it is perpendicular to the wire. In addition, both policies almost exclusively want to move forward in the  $u$  direction. The difference between the policies can be found in the lateral movement  $w$ . While the lateral movement of policy A is orientated to one side due to the continuous reward, policy B wants to move in either direction. In specific situations moving to the other side is needed to turn around a corner. As a result, policy A does not reach the finish for every wire.

### 6.2 Future work

While the goal of this project is reached with the development of the RL agent, there are many improvements to be made before the agent can be implemented on the real-life buzz-wire setup and deal with actual 3D wires. Therefore, the following recommendations are made:

1. An observer should be created that extracts the necessary features from images received from the 2D environment. Then this observer should be coupled with the policy and tested. The coupled observer and policy should be validated to check whether the coupling is done correctly and the extracted features match the ground truth features. If there are problems with the coupled system, some retraining might be required. By implementing this observer, vision in the loop will be achieved on the 2D model.
2. While the policy found in this graduation project is able to complete the objective with the given numerically computed features, they might be restrictive on the performance. Thus, a better policy might be found if the inputs to the policy are images of the 2D environment instead of the four chosen features. In this case, the agent needs to learn the mapping from the images to the action directly, instead of taking two separate steps (first to features, then to actions) and then coupling them. The performance of this agent should be validated and compared to the performance of the agent that uses the coupled networks.
3. To get the controller working on wires that are not constrained to a 2D plane, the 2D environment should be replaced with a 3D environment. This 3D simulation environment can be used to develop a policy, which can also work on the real buzz-wire setup. Even though observations in the 3D



simulation will contain less noise than the observations supplied by real-world sensors, the trained policy will be a relatively good initial policy.

4. Continuing from the previous recommendation, the learned policy should be implemented on the setup using transfer. This might require some retraining again to deal with the noise introduced by the sensors and camera. However, generating a policy using transfer learning will be quicker than starting from scratch. Using wires the agent has not seen during training, the policy should be evaluated.
5. The next recommendation is to look at some other learning algorithms than DQN. Algorithms, such as Double Deep Q-learning (DDQN) [21], Proximal Policy Optimization (PPO) [22] or Soft Actor Critic (SAC) [16], might be preferred in the 3D simulations or on the real setup. The performance of the different learning algorithms should be compared, to see which learning algorithm actually performs the best.
6. The final recommendation is to change the action from fixed translation/rotation steps in any direction to variable translation/rotations. This complicates the policy the agent has to learn as there are more actions possible. However, the new policy might perform much better as moving via states that were not reachable before might allow the robot to complete certain wires. Going even further, the output of the policy could be changed to the joints angles, speed of the joint angles, the forces at the joints, or even the voltages given to each motor. While this might complicate the policy even further, it could also improve the policy as its output can be directly send to the motors of the robot arm. This allows for more control since not only the position, but the speed and acceleration of each part of the robot are controlled as well.

## References

- [1] V. Kakani, V.H. Nguyen, B.P. Kumar, H. Kim, and V.R. Pasipuleti. A critical review on computer vision and artificial intelligence in food industrie. *Journal of Agriculture and food research*, 2:1–12, 2020.
- [2] H.C.J. Godfray, J.R. Beddington, I.R. Crute, L. Haddad, D. Lawrence, J.F. Muir, J. Pretty, S. Robinson, S.M. Thomas, and C. Toulmin. Food security: the challenge of feeding 9 billions people. *Science*, 327:812–818, 2010.
- [3] Y.K. Dwivedi, L. Hughes, E. Ismagilova, G. Aarts, T. Coombs, C. amd Crick, Y. Duan, R. Dwivedi, J. Edwards, A. Eirug, V. Galanos, P.V. Ilavarasan, M. Janssen, P. Jones, A.K. Kar, H. Kizgin, B. Kronemann, B. Lal, B. Lucini, R. Medaglia, K. Le Meunier-FitzHugh, L.C. Le Meunier-FitzHugh, S Misra, E. Mogaji, S.K. Sharma, J.B. Singh, V. Raghavan, R. Raman, N.P. Rana, S. Samothrakis, J. Spencer, K. Tamilmani, P. Tubadji, A. amd Walton, and M.D. Williams. Artificial intelligence (ai): Multidisciplinary perspectives on emerging challenges, opportunities, and agenda for research, practice and policy. *International Journal of Information Management*, 1:1–47, 2019.
- [4] T. Ahmad, D. Zhang, C. Huang, H. Zhang, N. Dai, Y. Song, and H. Chen. Artificial intelligence in sustainable energy industry: Status quo, challenges and opportunities. *Journal of Cleaner Production*, 289:1–31, 2021.
- [5] E. A. Gyasi, H. Handroos, and P. Kah. Survey on artificial intelligence (ai) applied in welding: A future scenario of the influence of ai on technological, economic, educational and social changes. *Procedia Manufacturing*, 38:702–714, 2019.
- [6] B. Wang, S. Jin, Q. Yan, H. Xu, C. Luo, L. Wei, W. Zhao, X. Hou, W. Ma, Z. Xu, Z. Zheng, W. Sun, L. Lan, W. Zhang, X. Mu, C. Shi, Z. Wang, J. Lee, Z. Jin, M. Lin, H. Jin, L. Zhang, J. Guo, B. Zhao, Z. Ren, S. Wang, W. Xu, X. Wang, J. Wang, Z. You, and J. Dong. Ai-assisted ct imaging analysis for covid-19 screening: Building and deploying a medical ai system. *Applied Soft Computing Journal*, 98:1–11, 2021.
- [7] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *International Conference for Learning Representations*, 2, 2017.
- [8] Y. Huang, J. Wilches, and Y. Sun. Robot gaining accurate pouring skills through self-supervised learning and generalization. *Robotics and Autonomous Systems*, 136:1–15, 2021.
- [9] C. Finn and S. Levine. Deep visual foresight for planning robot motion. *IEEE International Conference on Robotics and Automation.*, pages 2786–2793, 2017.
- [10] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2 edition, 2018.
- [11] E. Brunskill. Course on reinforcement learning. Online available from <http://web.stanford.edu/class/cs234/index.html>, 2019.
- [12] L. Zhou, H. Pu, F. Wang, Z. Hu, and L. Wang. The expressive power of neural networks: A view from the width. *Advances in Neural Information Processing Systems 30*, pages 4214–4223, 2017.
- [13] Niryo. Documentation of niryo one. Online available from <https://niryo.com/product/niryo-one/>.
- [14] T.B. Sriram. Machine learning-based vision-in-the-loop system for automated buzz wire demonstrator.
- [15] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrosik, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

- [16] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Han, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. *Computing Research Repository*, 2:1–17, 2018.
- [17] D. Schwab, T. Springenberg, M.F. Martins, T. Lampe, M. Neunert, A. Abdolmaleki, T. Hertweck, R. Hafner, F. Nori, and M. Riedmiller. Simultaneously learning vision and feature-based control policies for real-world ball-in-a-cup. *Robotics: Science and Systems.*, 2019.
- [18] L. Busoniu, T. de Bruin, D. Tolic, J. Kober, and I. Palunko. Reinforcement learning for control: Performance, stability, and deep approximators. *Annual Reviews in Control.*, 46:8–28, 2018.
- [19] D. Bertsekas. *Dynamic Programming and Optimal Control Vol 2*. Athena Scientific, 3 edition, 2007.
- [20] J. Tsitsiklis and B. V. Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42:674–690, 1997.
- [21] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *AAAI Conference on Artificial Intelligence*, 16:2094–2100, 2015.
- [22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *Computing Research Repository*, 2:1–12, 2017.
- [23] Sergio Guadarrama and Anoop Korattikara and Oscar Ramirez and Pablo Castro and Ethan Holly and Sam Fishman and Ke Wang and Ekaterina Gonina and Neal Wu and Efi Kokiopoulou and Luciano Sbaiz and Jamie Smith and Gábor Bartók and Jesse Berent and Chris Harris and Vincent Vanhoucke and Eugene Brevdo. TF-Agents: A library for reinforcement learning in tensorflow. Online available from <https://github.com/tensorflow/agents>.
- [24] OpenAI. Documentation of openai gym library. Online available from <https://gym.openai.com/docs/>.

## Appendix A Implemented agent

Just like the environment, the agent is set up in Python using the same libraries of Table C.1. Several training parameters can be set, which determine the number of times interactions between the agent, environment and replay buffer take place. These parameters effect the simulation time as well. They are shown in Table A.1.

Name	Setting	Type
EPOCHS	number of epochs	integer
COLLECT_STEPS_PER_EPOCH	number of experiences gathered per epoch	integer
TRAIN_STEPS_PER_EPOCH	number of mini-batches on which the agent is trained per epoch	integer
BATCH_SIZE	size of the mini-batch	integer

Table A.1: *Training settings*

The implemented agent is shown in Figure A.1. A training loop of the agent, also called an epoch, goes as follows. First, the agent uses its initial policy to take a number of steps in the environment. The amount of steps is set by the parameter COLLECT\_STEPS\_PER\_EPOCH. The data of each step is stored in the replay buffer. Then a mini-batch of experiences is sampled from the replay buffer. The size of the mini-batch is set with BATCH\_SIZE. Then the action-value function is updated and a new mini-batch is sampled. The amount of times this repeated is set with TRAIN\_STEPS\_PER\_EPOCH. After this, the target action-value function is updated. This completes the epoch. The number of epochs is set with EPOCHS.

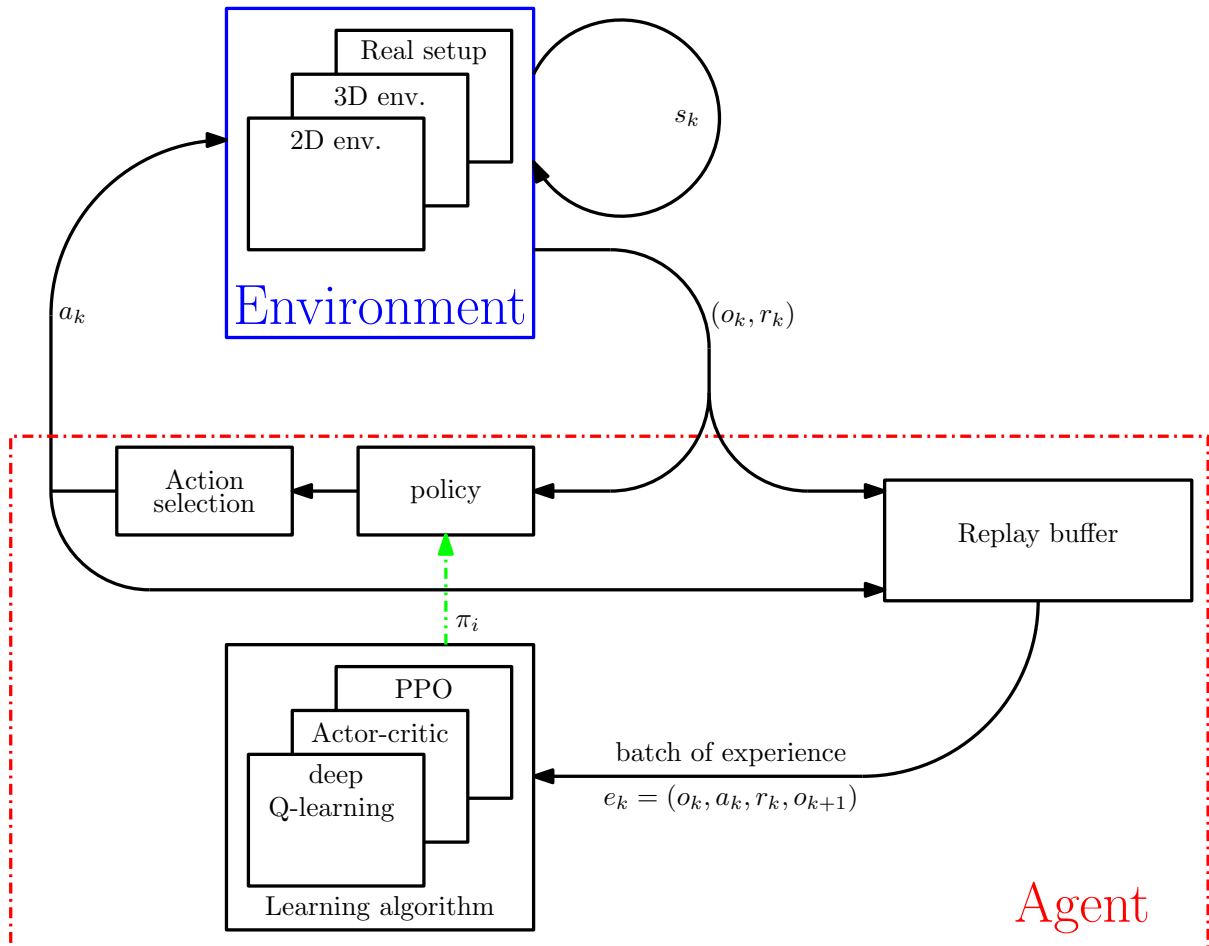


Figure A.1: *The interaction of the implemented agent and environment.*

The full algorithm of the agent is shown in Algorithm 1. This algorithm is implemented in TF-Agents [23], because this allows the agent to switch between different learning algorithms such as DQN [15] and DDQN [21].

**Algorithm 1: deep Q-learning with experience replay.**

Initialize replay memory  $D$  to capacity `RB_CAPACITY`

Initialize action-value function  $Q$  with random weights  $\mathbf{W}$

Initialize target action-value function  $\hat{Q}$  with weights  $\mathbf{W}^- = \mathbf{W}$

**For** epoch = 1, `EPOCHS` **do**

Initialize the environment and obtain the initial observation  $o_1$

**For** t = 1, `COLLECT_STEPS_PER_EPOCH` **do**

With probability  $\epsilon$  select a random action  $a_t$ , otherwise select  $a_t = \operatorname{argmax}_a Q(o_t, a; \mathbf{W})$

Execute action  $a_t$  in the environment and observe reward  $r_t$  and observation  $o_{t+1}$

Store transition  $(o_t, a_t, r_t, o_{t+1})$  in the replay buffer  $D$

**End For**

**For** t = 1, `TRAIN_STEPS_PER_EPOCH` **do**

Create a minibatch of size `BATCH_SIZE` by sampling transitions  $(o_j, a_j, r_j, o_{j+1})$  from  $D$

Set  $y_i = \begin{cases} r_j & \text{if episode terminates at step } j + 1 \\ r_j + \gamma \max_{a'} \hat{Q}(o_{j+1}, a'; \mathbf{W}^-) & \text{otherwise} \end{cases}$

Perform a gradient descent step on  $(y_i - Q(o_j, a_j; \mathbf{W}))^2$  with respect to network parameters  $\mathbf{W}$

**End For**

Set  $\hat{Q} = Q$

Update the learning rate  $\alpha$  and  $\epsilon$ .

**End For**

## Appendix B Sets of 2D wires

Two different subsets of wires are explored. One subset of wires consists of sinusoids, while in the other subset each wire has two perpendicular corners in opposite direction. Both wires are discretized in  $n_{wire}$  points, which are equidistantly taken along the  $x$  axis. The  $x$  and  $y$  coordinates of these points on the sine wires are

$$\mathbf{x}_{wire,e,0} = [0, \dots, d_{goal}] \quad (\text{B.1})$$

$$\mathbf{y}_{wire,e,0} = a_e \cdot \sin(\mathbf{x}_{wire,e,sine} \cdot 2\pi \cdot f_e) \quad (\text{B.2})$$

where  $a_e$  is the amplitude of the sine wave sampled from the range  $[-a_{max}, a_{max}]$  (with  $a_{max}$  being the maximum amplitude of the sine),  $f_e$  is the frequency of the sine wave sampled from the range  $[\frac{1}{d_{goal}}, \frac{n_{waves}}{d_{goal}}]$  (with  $n_{waves}$  being the maximum number of waves allowed in the wire) and  $d_{goal}$  is the distance between the target position and the start.

The second wire set consist of two perpendicular corners and is defined as

$$\mathbf{x}_{wire,e,0} = [0, \dots, d_{goal}] \quad (\text{B.3})$$

$$\mathbf{y}_{wire,e,0} = b_e \cdot g(\mathbf{x}_{wire,e,corner}). \quad (\text{B.4})$$

where  $b_e$  is either  $+1$  or  $-1$  and determines the order of the corners (first left then right or reversed) and the function  $g$  is given by

$$g(x) = \begin{cases} -x & \text{for } x \leq \frac{1}{4}d_{goal} \\ x - \frac{1}{2}d_{goal} & \text{for } \frac{1}{4}d_{goal} \leq x \leq \frac{3}{4}d_{goal} \\ d_{goal} - x & \text{for } x \geq \frac{3}{4}d_{goal} \end{cases} \quad (\text{B.5})$$

Both sets of the wire are rotated by a random angle  $\alpha_e$ . This results in the final  $x$  and  $y$  coordinates of the wire being

$$\begin{bmatrix} \mathbf{x}_{wire,e} \\ \mathbf{y}_{wire,e} \end{bmatrix} = \begin{bmatrix} \cos(\alpha_e) & \sin(\alpha_e) \\ -\sin(\alpha_e) & \cos(\alpha_e) \end{bmatrix} \begin{bmatrix} \mathbf{x}_{wire,e,0} \\ \mathbf{y}_{wire,e,0} \end{bmatrix}. \quad (\text{B.6})$$

Examples of both wires are shown in Figure B.1, where the blue wire is part of the first subset and the red wire is part of the second subset.

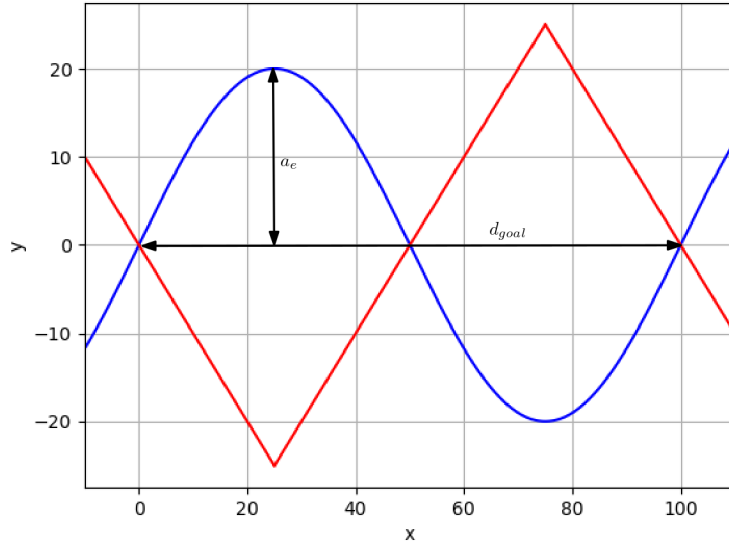


Figure B.1: Examples of both wires. The blue wire is part of the sine wire subset and the red wire is part of the corner subset.



## Appendix C Implementation of the 2D model

The 2D model of the buzz-wire is setup in Python (version 3.7.9). The required libraries and their corresponding versions are shown in Table C.1. Next to that, the 'Robot\_Environment' folder, should be put in the working directory as well. This folder contains the script files of several environments.

Library	Version
numpy	1.18.5
gym	0.17.2
opencv-python	4.4.0.44
matplotlib	3.3.1
imageio	2.9.0
tensorflow	2.3.0
tensorflow-probability	0.11.0
TF-agents	0.6.0

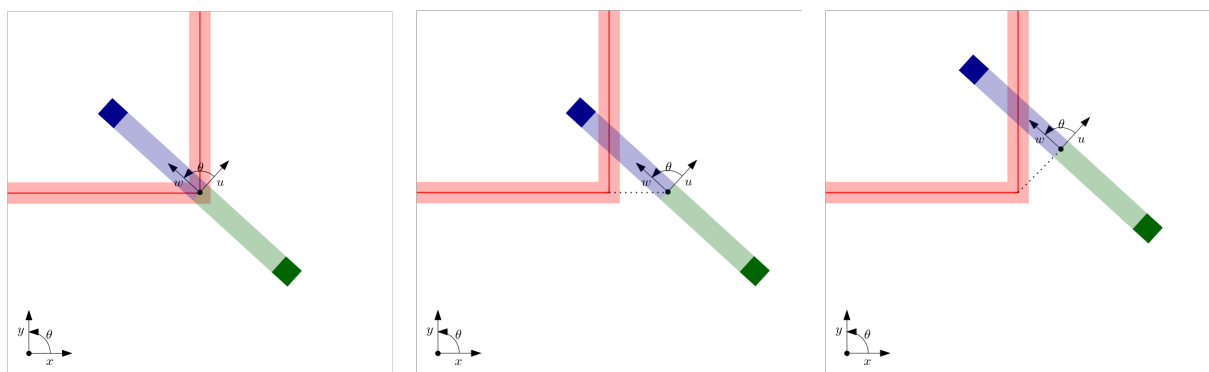
Table C.1: overview of used existing libraries.

To interact with the environment, an instance of the environment has to be created using the command `gym.make(...)` [24], where the name of the environment is filled in on the dots. Names of several 2D buzz-wire environments can be found in Table C.2.

Name	Coordinates	Observations
'RobotWireEnvDefault-v0'	global	camera image and end effector position
'RobotWireEnvDefault-v1'	global	the features discussed in Section 4.5
'RobotWireEnvDefault-v2'	local	camera image and end effector position
'RobotWireEnvDefault-v3'	local	the features discussed in Section 4.5

Table C.2: Implemented 2D buzz-wire environments, with the moves in local or global coordinates and the observations being features or camera image and end effector position.

The first two environments move the loop with respect to the global coordinate system  $(x, y)$ , while the last two environments move in the local coordinate system that is attached to the loop  $(u, w)$ . Taking action 16 results in different moves for these environments, as shown in Figure C.1. Next to the coordinate systems, the observation space differs between the environments. The observation space can either be the RGB values of the camera and the end effector position or the features  $(d_{BD}, d_{AC}, \theta, \phi)$  mentioned in Section 4.5.



(a) Situation before action is taken. (b) Situation after action in  $(x, y)$ . (c) Situation after action in  $(u, w)$ .

Figure C.1: Result of taking action 16 in the different environments.

Besides choosing the coordinate system and the observations space, other parameters of the environment can be set. For example, the thickness of the wire  $r_{wire}$  can be set to 2 as follows

```
env = gym.make('RobotWireEnvDefault-v3', thickness=2)
```

1



Table C.3 shows the parameters of the environment that can be set via `gym.make(...)`. For each parameter the default value is given as well.

Name	Symbol	Input	Type	Default value
<code>amp_max</code>	$a_{max}$	maximum amplitude of the sine wire	float	10
<code>diameter</code>	$d_{loop}$	diameter of the loop	float	10
<code>distance_roi</code>	$d_{roi}$	distance between center of the loop and point to calculate wire angle	float	5
<code>eval_episodes</code>	$n_{eval}$	optional to fix the validation set, given number is the size of the validation set	integer	None
<code>goal</code>	$d_{goal}$	distance from start to target	float	100
<code>length</code>	$l_{loop}$	thickness of the loop	float	1
<code>n_checkpoints</code>	$n_{check}$	optional, number of checkpoints on the wire	integer	1
<code>n_ele_wall</code>	$n_{wall}$	total number of elements in the wall on both ends on the wire	integer	1000
<code>n_ele_wire</code>	$n_{wire}$	total number of elements in the wire	integer	1000
<code>n_enc_angle</code>	$n_{\phi}$	number of segments in which the angle between the wire and the loop is discretized	integer	11
<code>n_enc_circle</code>	$n_{\theta}$	number of segments in which the angle between the loop and the camera is discretized	integer	21
<code>n_enc_line</code>	$n_d$	number of segments in which the line segments BD and AC are split	integer	9
<code>n_waves_max</code>	$n_{waves}$	maximum number of waves in the sine wire	integer	2
<code>obs_ele_h</code>	$h_{obs}$	height of the camera image	integer	32
<code>obs_ele_w</code>	$w_{obs}$	width of the camera image	integer	32
<code>obs_pixel_size_h</code>	$h_{px}$	height of a single camera pixel	integer	1
<code>obs_pixel_size_w</code>	$w_{px}$	width of a single camera pixel	integer	1
<code>percent_sine</code>		fraction of the wire set being sine wires when switching between corner and sine wires	float	0.5
<code>reset_mode</code>		what kind of reset happens after the wire is hit (back to start, from checkpoint or previous position)	string	'prev_pos'
<code>reward_checkpoint</code>	$r_{check}$	reward given when a checkpoint is reached	float	0
<code>reward_finish</code>	$r_{finish}$	reward given when the finish is reached	float	1000
<code>reward_hit</code>	$r_{hit}$	reward given when the wire is hit (should be negative)	float	-100
<code>reward_progress</code>	$r_{progress}$	reward earned for progressing along the wire	float	0
<code>seed</code>		optional to fix the environment to a single wire	integer	None
<code>thickness</code>	$r_{wire}$	thickness of the wire	float	1
<code>theta_step</code>	$\theta_{step}$	step-size of the rotation	float	$\frac{\pi}{36}$
<code>wire_set</code>		the chosen (sub)set(s) of wire(s), such as the sine and corner wire	string	'switched'
<code>x_step</code>	$u_{step}$	step-size of the longitudinal translation	float	1
<code>y_step</code>	$w_{step}$	step-size of the lateral translation	float	1

Table C.3: Variables of the 2D buzz-wire environment

With the instance of the environment created, it can be interact with it. This is done via the five functions

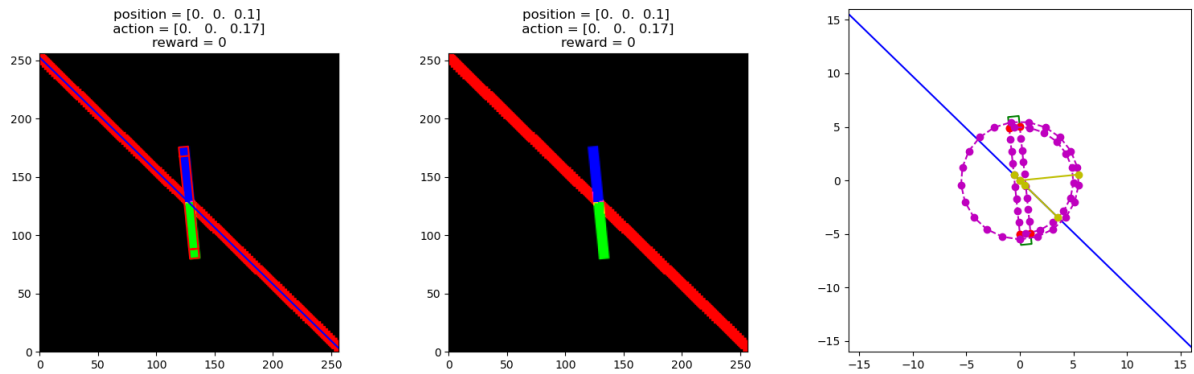
- `env.reset(x_init,y_init,theta_init)`, which resets the environment and starts a new episode. The inputs are optional, and define the starting position and orientation. Otherwise the default values are 0 for the position and the angle is the derivative of the wire at the position (0,0). The reset function returns the loop position to this starting position and orientation. It also resets the collected reward and time step to 0 and resamples the wire. The output of this function is the first observation  $o_0$ .
- `env.step(action)`, which takes a step in the environment. The action  $a_k$  that is fed to the function is converted to a translation and rotation via Table C.4 (if the environments take actions in global coordinates, then  $x$  is replaced by  $u$  and  $y$  by  $w$ ). Then a check is done to see if the loop can perform this move without hitting the wire. Is this possible, then the loop is moved to its new position. Has the wire been hit, then the loop either stays at its current position ('prev\_pos'), is returned to the last checkpoint ('checkpoint') or is returned to the starting position ('start') based on the `reset_mode`. After the loop has moved (or not), the reward  $r_k$  can be calculated using Equation (4.4) and Equation (4.6). From the new state  $s_{k+1}$ , the new observation  $o_{k+1}$  is calculated. The new observation  $o_{k+1}$ , reward  $r_k$  are returned by the function as well as a boolean indicating if the episode has ended (which is only the case if the finish is reached).

Discrete action $a$	Longitudinal movement $\Delta u$	Lateral movement $\Delta w$	Rotation $\Delta\theta$
0	$-\frac{1}{\sqrt{2}}u_{step}$	$-\frac{1}{\sqrt{2}}w_{step}$	$-\theta_{step}$
1	$-\frac{1}{\sqrt{2}}u_{step}$	$-\frac{1}{\sqrt{2}}w_{step}$	0
2	$-\frac{1}{\sqrt{2}}u_{step}$	$-\frac{1}{\sqrt{2}}w_{step}$	$\theta_{step}$
3	0	$-w_{step}$	$-\theta_{step}$
4	0	$-w_{step}$	0
5	0	$-w_{step}$	$\theta_{step}$
6	$\frac{1}{\sqrt{2}}u_{step}$	$-\frac{1}{\sqrt{2}}w_{step}$	$-\theta_{step}$
7	$\frac{1}{\sqrt{2}}u_{step}$	$-\frac{1}{\sqrt{2}}w_{step}$	0
8	$\frac{1}{\sqrt{2}}u_{step}$	$-\frac{1}{\sqrt{2}}w_{step}$	$\theta_{step}$
9	$-u_{step}$	0	$-\theta_{step}$
10	$-u_{step}$	0	0
11	$-u_{step}$	0	$\theta_{step}$
12	0	0	$-\theta_{step}$
13	0	0	0
14	0	0	$\theta_{step}$
15	$u_{step}$	0	$-\theta_{step}$
16	$u_{step}$	0	0
17	$u_{step}$	0	$\theta_{step}$
18	$-\frac{1}{\sqrt{2}}u_{step}$	$\frac{1}{\sqrt{2}}w_{step}$	$-\theta_{step}$
19	$-\frac{1}{\sqrt{2}}u_{step}$	$\frac{1}{\sqrt{2}}w_{step}$	0
20	$-\frac{1}{\sqrt{2}}u_{step}$	$\frac{1}{\sqrt{2}}w_{step}$	$\theta_{step}$
21	0	$w_{step}$	$-\theta_{step}$
22	0	$w_{step}$	0
23	0	$w_{step}$	$\theta_{step}$
24	$\frac{1}{\sqrt{2}}u_{step}$	$\frac{1}{\sqrt{2}}w_{step}$	$-\theta_{step}$
25	$\frac{1}{\sqrt{2}}u_{step}$	$\frac{1}{\sqrt{2}}w_{step}$	0
26	$\frac{1}{\sqrt{2}}u_{step}$	$\frac{1}{\sqrt{2}}w_{step}$	$\theta_{step}$

Table C.4: Possible discrete actions and their translation in longitudinal and lateral direction and rotation.

- `env.seed(seed)`, which seeds the random number generator. With this function, wires can be reproduced. The input and output of this function is the seed.

- `env.render(mode)`, which visualizes the current state  $s_k$ . There are three possible visualizations: 'human', 'rgb\_array' and 'full\_view'. They are shown in Figure C.2a, Figure C.2b and Figure C.2c respectively. If the function receives no input, the 'human' mode is used. This mode plots a camera image with an overlay of the exact outline of the wire and the loop. At the top of the plot the position of the loop  $(x_k, y_k, \theta_k)$ , the previous action  $(\Delta u_{k-1}, \Delta w_{k-1}, \Delta \theta_{k-1})$  and the total reward of that episode are shown. In this mode the total reward is returned. Changing the mode to the 'rgb\_array' representation removes the outline of the loop and wire in the plot. Moreover, the output of the function is changed to the rgb-values of the plot instead of the total reward. The final representation, 'full\_view' plots the whole wire, the loop and the features. In this mode the output of the function is reverted back to the total reward of the episode.
- `env.close()`, which closes the visualization. This function is simply used to stop the visualization of the current state.



(a) Camera view with overlay of exact situation. (b) The camera view without an overlay. (c) Zoomed view of the environment with the features.

Figure C.2: Visualizations of the environment.

To enable interaction with the learning algorithm in TF-agents, the gym environment is wrapped to conform to the standard method used in TF-agents. While TF-agents can interact with the wrapped environment, operations can not be parallelized resulting in a relatively slow algorithm. To speed up the algorithm, the environment is wrapped again to a Tensorflow environment, which allows parallel operations. Wrapping the environment is done as follows

---

```

1     env_suited = suite_gym.wrap_env(env, discount=GAMMA) \\
2     env_train = tf_py_environment.TFPyEnvironment(env_suited)

```

---

where `GAMMA` is the discount factor  $\gamma$ . The first wrapper conforms the environment to the standard TF-agents environments and the second wrapper creates the Tensorflow environment.

## Appendix D Settings of the training sessions

The settings of the training environment during training sessions can be found in the table below.

Environment settings	Training session of		Training sessions with bugs	
	policy A	policy B	policy C	policy D
amp_max	20	20	40	20
diameter	10	10	10	10
distance_roi	5	5	5	5
eval_episodes	100	100	50	100
goal	100	100	100	100
length	1	1	1	1
n_checkpoints	1	1	1	1
n_ele_wall	1000	1000	100000	1000
n_ele_wire	1000	1000	10000	1000
n_enc_angle	11	11	11	11
n_enc_circle	21	21	21	21
n_enc_line	9	9	9	9
n_waves_max	2	2	2	2
obs_ele_h	256	256	256	256
obs_ele_w	256	256	256	256
obs_pixel_size_h	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
obs_pixel_size_w	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
percent_sine	0.75	0.75	.60	0.75
reset_mode	'prev_pos'	'prev_pos'	'prev_pos'	'prev_pos'
reward_checkpoint	0	0	0	0
reward_finish	2000	2000	0	2000
reward_hit	-100	-100	-100	-100
reward_progress	2000	0	2000	0
seed	None	None	None	None
thickness	1	1	1	1
theta_step	$\frac{\pi}{18}$	$\frac{\pi}{18}$	$\frac{\pi}{18}$	$\frac{\pi}{18}$
wire_set	'switched'	'switched'	'switched'	'switched'
x_step	1	1	1	1
y_step	1	1	1	1

Table D.1: *Environment parameters during the training sessions.*

The training settings of the sessions are shown in the table below.

Environment settings	Training session of		Training sessions with bugs	
	policy A	policy B	policy C	policy D
ALPHA_START	0.01	0.01	0.01	0.01
ALPHA_END	0.0001	0.0001	0.0001	0.0001
EPSILON_START	1.0	1.0	1.0	1.0
EPSILON_END	0.05	0.05	0.05	0.05
GAMMA	0.2	0.2	0.2	0.2
EPOCHS	1000	1000	20	1000
COLLECT_STEPS_PER_EPOCH	2000	2000	2000	2000
TRAIN_STEPS_PER_EPOCH	100	100	100	100
BATCH_SIZE	1000	1000	1000	1000
RB_CAPACITY	1000000	1000000	1000000	1000000

Table D.2: *Environment parameters during the training sessions.*

## Network architecture

The neural network used during the training sessions of policy A, policy B and policy D to approximate the action-value function consist of an input layer, an output layer and 5 times a fully connected layer followed by a dropout layer. The dropout rate of the dropout layers is set to 0.1. The fully connected layers all have 50 neurons. Only during training of policy C a different network was used. This other network consisted of 2 fully connected layers with 50 neurons in them and no dropout layers. The input and output layer of the network is unchanged as the features and number of actions were not altered.

## Appendix E Problems during training

At first, policies were trained on the environment that moves in the global coordinate system and has a continuous reward. However, this complicated the reward function and made the policy hard to understand, due to the reward function being defined along the longitudinal axis. Looking at the buzz-wire, the loop will always make progress along the wire, when moving longitudinally (in the  $u$  direction). This leads to a clear policy of trying to move forward whenever there is no large turn ( $\phi$  very large or small), compare policy D in Figure E.2 to policy C in Figure E.1. Thus the choice was made to switch to the environment in which the moves are made in the local coordinate system instead of the global coordinate system.

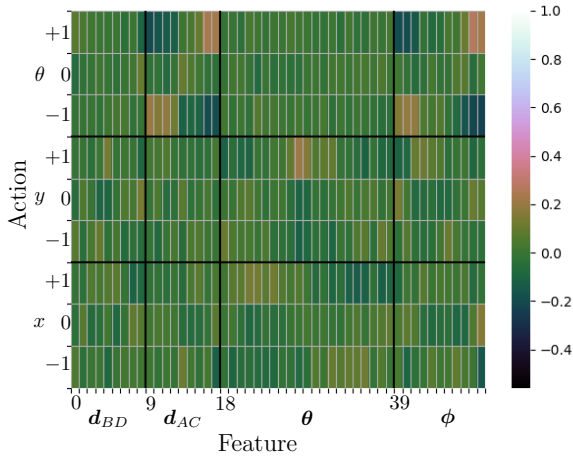


Figure E.1: Covariance between the features and actions under the policy learned in an environment which moves in the global coordinate system (policy C in Appendix D).

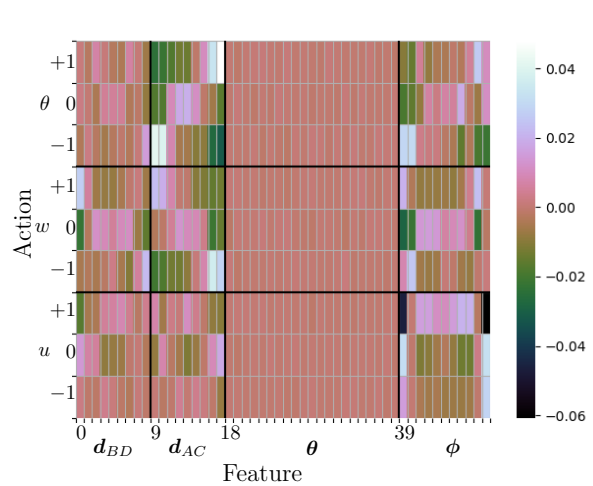


Figure E.2: Covariance between the features and actions under the policy learned in an environment which moves in the local coordinate system (policy D in Appendix D).

The second bug that was encountered is related to the step size. The actions were defined as  $-1$ ,  $0$  or  $+1$  times the step size in any directions. Steps in multiple directions were allowed as well. This resulted in a policy that tried to rotate the move 45 degrees from the heading of the wire and then translate in both the  $u$  and  $w$  direction, see Figure E.3.

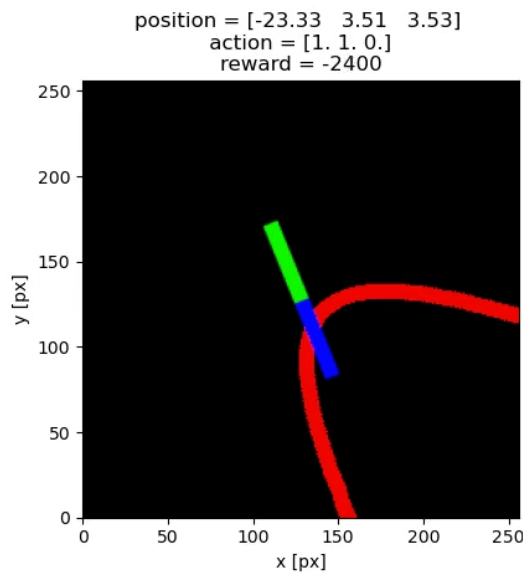


Figure E.3: The agent getting stuck due to the large steps in wants to take in the forward and lateral direction.

As a result the loop can take larger steps and reach the finish faster, compare Figure E.4a to Figure E.4b. However, the larger step size and the orientation make the policy less robust and can lead to the agent getting stuck as shown in Figure E.3. This second problem is solved by normalizing the step translation such that the speed in any direction would be equal.

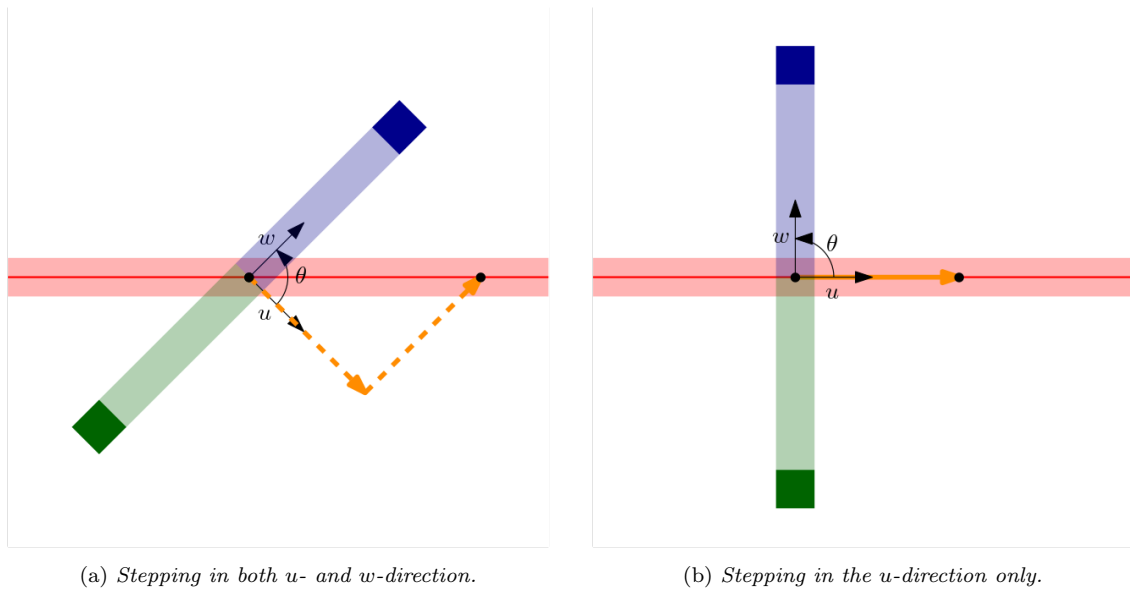


Figure E.4: The travelled distance of two different moves when the step-size is not normalized.

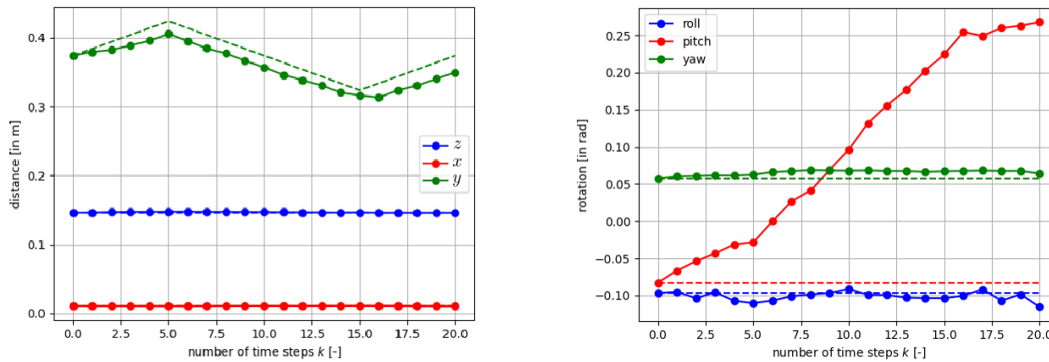
After fixing both these problems, new policies were trained. These policies performed much better, as shown by the results in Chapter 5.

## Appendix F Results of the hysteresis experiments on the Niryo One robot arm

### One robot arm

The hysteresis of the Niryo One robot arm was tested by moving the arm in a given dimension  $n_{steps}$  in the positive direction, then moving it  $2n_{steps}$  in the negative direction and finally moving it  $n_{steps}$  in the positive direction again. If there is no hysteresis the robot arm should end in the same position as it started in. The dotted lines show the positions and orientations the arm should move to.

Figure F.1a shows that the  $y$  position is not followed well in the positive direction. Moving in the positive  $y$  direction, means the robot arm has to overcome gravity. This might explain why loop does not reach the  $y$  position it should. As a result, the final  $y$  coordinate is lower than the starting  $y$  coordinate. Looking at Figure F.1b, the pitch also has a large drift compared to the roll and yaw.

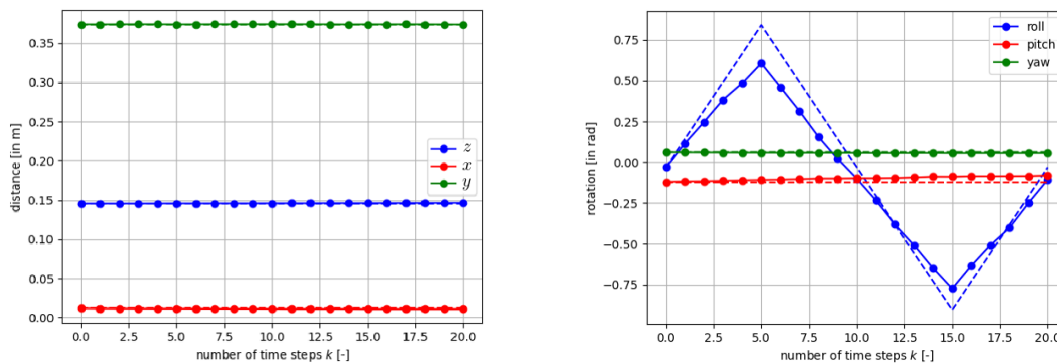


(a) The position of the loop.

(b) The orientation of the loop.

Figure F.1: The position of the loop while taking 5 steps in the positive  $y$  direction, followed by 10 steps in the negative  $y$  direction and then another 5 steps in the positive  $y$  direction.

Figure F.2a shows that there is no drift in the position when only changing the roll. This is because the roll is changed by turning the final joint of the robot arm. Therefore, other joints that can influence the position are not changed and thus there is no drift. As seen in Figure F.2b, the other rotations also hardly drift because of this. Only the roll itself is not following the command signal well. This might be caused by friction in the motor.



(a) The position of the loop.

(b) The orientation of the loop.

Figure F.2: The position of the loop while taking 5 steps in the positive  $\theta$  direction, followed by 10 steps in the negative  $x\theta$  direction and then another 5 steps in the positive  $\theta$  direction.