Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Arild Vermedal

Deep Reinforcement Learning for Valve Manipulation

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2019

NTNU
Norwegian University of
Science and Technology

Arild Vermedal

Deep Reinforcement Learning for Valve Manipulation

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Preface

This thesis was written during spring 2019 with the objective to perform manipulator tasks using reinforcement learning at the Norwegian University of Science and Technology (NTNU), and concludes my Master of Science in Cybernetics and Robotics. The work involved both a physical setup and a simulated environment to consider the study complete. It is a continuation of my work of fall 2018 exploring the theory of reinforcement learning and its application to a robotic manipulator (Vermedal, 2018). The contributions of this thesis is in the review of the suitability of modern simulation software for robotics and the use of reinforcement learning in combination with behavioral cloning for controlling a real-world robotic manipulator, as well as building a setup for reinforcement learning with a robotic manipulator both in simulation and physically.

The following equipment was provided by NTNU:

- OpenManipulator-X (RM-X52-TNM) robotic manipulator from ROBOTIS, with ROS, MoveIt and Gazebo simulation.
- Raspberry Pi camera v2.
- Base for mounting valve and robotic manipulator.
- ADC0844CCN analog to digital converter, breadbox and wires connecting it to potentiometer and Raspberry Pi.

My Proximal Policy Optimization implementation is based on an implementation by Anjum Sayed (Sayed). It was used with only minor adaption to work for the specific environment used and to work for Python 2.7. It should be noted that this is a different implementation than the one used in my project thesis of last semester. This change was made due to this version's more complete setup of the algorithm, implementation done with tensorflow, and better configuration possibilities. My supervised learning network for behavioral cloning is based on an implementation by Erik Shestopal (Shestopal). This underwent major changes to adapt it from learning to drive from images, to learning to control the manipulator based on previous control signals. In particular, the neural network was changed to match the configuration used in the PPO implementation and the

processing of the sampling adapted to the format used for saving manually-operated task completions.

A final thanks to

- my supervisor Anastasios Lekkas for his guidance and advice
- my fellow masters student Anders Haver Vagle who worked on a similar task, and with whom I collaborated with setting up the simulated environment and designing and building the valve
- NTNU, department of Engineering Cybernetics' workshop, and in particular Stefano Bertelli for the construction of the valve and base including the potentiometer and Åsmund Stavdahl for supplying the ADC and associated circuit parts
- UBIQUITY ROBOTICS for their publicly available Raspberry Pi image with pre-installed ROS kinetic for Raspberry PI 3 model B (Ubiquity Robotics)
- NTNU for providing a workspace and a computer to work with

Abstract

This thesis is inspired by the recent rapid advancements in artificial intelligence, in particular reinforcement learning, and the increased viability of practical applications such as subsea operations in the oil and gas industry, specifically the application of reinforcement learning to the task of controlling robotic manipulators. The aim of this thesis is to develop a platform enabling the training of an RL agent in a simulation and on a real-world manipulator such that the agent is able to switch between them with no reconfiguration and is able to reuse the same policy in both cases, and then use that platform to solve a modelled subsea task using both reinforcement learning and behavioral cloning.

An "OpenManipulator-X (RM-X52-TNM)" manipulator with a simulation implemented in ROS and Gazebo was purchased after a search for a robotic manipulator in a price range affordable for research with an accompanying simulation solution. This software solution for simulation was evaluated for reinforcement learning, showing itself to be ill-suited for episodic training, in particular by being unable to reset the manipulator to the initial configuration, and having a poor real-time factor. However, experience in the simulated environment transferred well to a real-world setting.

A simplified environment for the task of reaching and turning a valve was designed and created for both simulation and real-world. Proximal Policy Optimizaton (PPO) was then used to learn to solve each task in simulation, and the resulting policy used to control the real-world manipulator. Reaching the valve was solved well in simulation and gave equivalent results when applied to the manipulator, while turning the valve was moderately successful in simulation but performed very well on the real-world manipulator. A second solution was attempted using behavioral cloning and several sets of expert demonstrations performing the tasks, giving better performance than reinforcement learning with PPO on reaching the valve, but worse on turning it. Lastly, the two approaches were combined, where behavioral cloning was used to generate a policy in the form of an artificial neural network, and this policy function used as the initial policy for the PPO algorithm. This performed similarly to the PPO approach and greatly reduced the training time.

In this thesis, an overview of the development of the control of robotic manipulator over the last 50 years is presented, followed by an introduction to the current field of reinforcement learning, the basic theory underlying it and the specifics employed by PPO. The process and particulars of setting up the software and hardware are described, and the results of the various task solutions presented. Lastly, the process and results are discussed, and the thesis' conclusions given.

Sammendrag

Denne oppgaven er inspirert av den raske utviklingen innen kunstig intelligens, særlig forsterkende læring, og den økte anvendelsen for praktiske formål, blant annet til undervannsoperasjoner i olje- og gassindustrien, og da spesielt anvendelsen av forsterkende læring til¨å kontrollere robotmanipulatorer. Formålet med denne oppgaven er å utvikle en plattform som muliggjør opplæring av en forsterkende læringsagent i en simulering og på en virkelig manipulator slik at agenten er i stand til å bytte mellom dem uten omkonfigurering og er i stand til å gjenbruke samme politikk i begge tilfeller, og deretter bruke plattformen for å løse en modellert undersjøisk oppgave ved hjelp av både forsterkende læring og atferdsmessig kloning.

"OpenManulator-X (RM-X52-TNM)"-manipulatoren med en simulering implementert i ROS og Gazebo ble kjøpt etter et søk for en robotmanipulator i ee prisklasse som var rimelig for forskning med en tilhørende simuleringsløsning. Denne programvareløsningen for simulering ble evaluert for forsterkende læring, noe som viste seg å være dårlig egnet for episodisk opplæring, spesielt ved å ikke være i stand til å nullstille manipulatoren til den opprinnelige konfigurasjonen og ha en dårlig sanntidsfaktor. Erfaring i det simulerte miljøet overføres imidlertid godt til virkelige omgivelser.

Et forenklet miljø for oppgaven å nå frem til og å vri en ventil ble designet og opprettet for både simulering og virkeligheten. Proximal Policy Optimizaton (PPO) ble da brukt til å lære å løse hver enkelt oppgave i simuleringen, og den resulterende politikken brukes til å kontrollere den virkelige manipulatoren. Å åpne ventilen ble løst godt i simuleringen og gav ekvivalente resultater når den ble brukt på manipulatoren, mens å vri ventilen var svakt vellykket i simuleringen, men ble utført veldig bra på den virkelige manipulatoren. En annen løsning ble forsøkt ved å bruke adferdskloning og flere sett med ekspertdemonstrasjoner som utførte oppgavene, noe som ga bedre ytelse enn forsterkende læring med PPO når det gjelder å nå ventilen, men verre for å vri den. Til slutt ble de to tilnærmingene kombinert, hvor adferdskloning ble brukt til å generere en politikk i form av et kunstig nevralt nettverk, og denne politikkfunksjonen ble brukt som innledende politikk for algoritmen for PPO. Dette presterte på samme nivå som PPO-tilnærmingen og reduserte treningstiden sterkt.

I denne oppgaven presenteres et oversiktsbilde av utviklingen til kontroll av robot manipulatorer over de siste 50 årene, etterfulgt av en introduksjon til det nåværende feltet for forsterkende læring, den grunnleggende teorien som ligger til grunn for den og de spesifikke aspektene som brukes av PPO. Videre presenteres prosessen og opplysningene om å sette opp programvaren og maskinvaren, og resultatene av de ulike oppgaveløsningene. Til slutt diskuteres prosessen og resultatene, og avhandlingens konklusjoner blir gitt.

Contents

List of Figures

Abbreviations

Artificial Intelligence	AI
Artificial Neural Network	ANN
Behavioral Cloning	BC
Computed Torque Method	CTM
Deep Deterministic Policy Gradient	DDPG
Deep Q-Learning	DQL
Degree of Freedom	DoF
Guided Policy Search	GPS
Imitation Learning	IL
Linear-Quadratic Controller	LQR
Machine Learning	ML
Makrov Decision Process	MDP
Mean Squared Error	MSE
Model Predictive Control	MPC
Model Value Expansion	MVE
Model-Ensemble Trust-Region Policy Optimization	ME-TRPO
Nonlinear Model Predictive Controller	NMPC
Normalized Advantage Functions	NAF
Norwegian University of Science and Technology	NTNU
OpenManipulator	OM
Policy Iteration	PI
Probabilistic Inference for Learning Control	PILCO
Proximal Policy Optimization	PPO
Raspberry Pi	RP
Reach Valve	RV

Rectified Linear Units	ReLu
Reinforcement Learning	RL
Robot Operating System	ROS
Robotic Manipulator	RM
State-Action-Reward-State-Action	SARSA
Stochastic Ensemble Value Expansion	STEVe
Turn Valve	TV
Turn Valve Simplified	TVS
Trust Region Policy Optimization	TRPO
Value Iteration	VI

Chapter 1

Introduction

This chapter will explain the motivation for the work as a whole, and the problem description being worked on. It will then give a brief overview of the control of robotic manipulators, before presenting a look at the current state of the art in reinforcement learning (RL) for the control of robotic manipulators (RM). Sections 1.2-1.3 are updated versions of the author's earlier work (Vermedal, 2018).

1.1 Motivation and problem description

Autonomous robotics systems are growing more and more popular in research with the advent of advanced artificial intelligence (AI) control techniques capable of tackling complex, dynamic problems. Yet, real world robotic systems remain dominated by inflexible classic control methods, or are remotely operated by humans, and current research for new RL methods and their performance focus mainly on achieving good results in the real-world by solving problems in simulated enviornments first, and the transferring the learned knowledge to the real-world. (Lillicrap et al., 2015; Schulman et al., 2017).

The main objective in this thesis is to apply state-of-the-art methods in reinforcement learning to control a robotic manipulator. The goal of the RL agent will be to learn a policy for manipulating objects in a way that resembles subsea intervention tasks, such as turning on/off a valve. The task will be first solved in a simulated space using synthetic data. The resulting policy will be used to initiate the task in the real environment. Additionally, the use of behavioral cloning (BC) to generate the initial policy will be explored. The thesis will also evaluate the suitability of the available simulation platforms of RMs for RL.

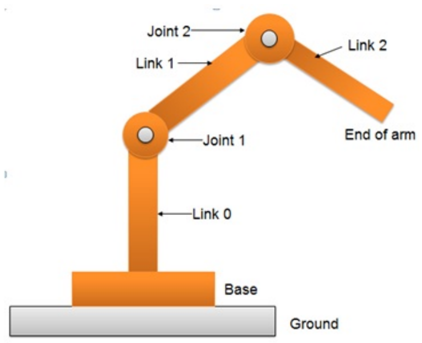


Figure 1.1: The basic shape of an RM. Picture: nptel

1.2 History of Robotic Manipulator Control

To understand the current field of robotic control, and how AI methods compare to other control methods, it is useful to know the history and development of those methods. This literary survey was performed by reading popular and heavily cited textbooks from different times, and looking at their sources, what new developments are being described, and the current state of the art. This was then used as the basis for a new round of research, locating primary sources and survey reports from the then active areas of research.

An RM is a subset of the category of industrial robots. RM may have multiple interpretations and configurations, but will here be defined as an open kinematic chain of rigid bodies interconnected by joints with a specialized tool at the end, as described by Siciliano (2009) and shown in Figure 1.1. RMs have been in development for a long time, and it can be unclear exactly when something we would today call an RM came to be. The precursor to the early RM were the teleoperated systems, where a slave machine copied the movements performed by a human-operated master device, later including force feedback to the operator (Spong, 1989).

A brief overview of the development of the control of RMs through the 60s is given by Paul (1981), starting with George Devols' robot arm. This device has little claim to the term *robot* as it is understood today. Instead, it was guided through a series of motions, which it stored and was then able to repeatably play back. Nevertheless, this simple repeated motions machine was useful for similarly simple tasks, and the machine could be reprogrammed to perform a new task when needed. Only a year later, in 1961 at MIT, another RM was created, using touch sensors to detect when it had run into something, which would then activate a preprogrammed response. The technology remained dependent on predetermined, humanly added knowledge of the environment. It did, however, not rely on absolute movements, and was perhaps the first step towards programming goals and letting the machine determine when they were accomplished. It was also in the 60s that the first real steps in computer vision were taken, and the position and orientation of known, simple objects could be determined from images in real time (Wichman, 1967).

1.2.1 1970–80

In the 1970s, more advanced methods for control were being researched, while most industrial robots in use remained either teleoperated by humans, or copying a set of movements with simple, slow control methods to guide them (Dubowsky and DesForges, 1979). One of the focuses of research were methods of specifying the end of the movement in Cartesian coordinates and have them translated into joint positions and realized. Multiple approaches were used, such as solving complicated polynomials for planning and using a simple PID-like controller to adjust the exact inputs (Paul, 1972). In comparison, past methods would linearize the system and neglect external loads (Dubowsky and DesForges, 1979). Adaptive methods that would not require exact knowledge of the parameters of the RM were being developed. They could estimate both their parameters and the mass properties of an external load (Paul, 1981). Still, they were based on linearization of the system and simplifying the load to a simple mass attached to the end. Landau (1974) compiles a survey of the state of the art research on model reference adaptive techniques, summarizing that adaptive methods remain limited by the available computing power and detail of the modelled dynamics of the system.

1.2.2 1980–90

The works of Uchiyama (1989) details the state of the art of control of robot arms by that time, split into two main categories: motion control and force control. Motion control is based on the control of the absolute position of the end-effector, split again into model-based control and feedback control. Model-based control was a recent innovation for practical use, as it was only possible with a fast and powerful computer. These methods also required a detailed model of the system. The payout for these high requirements were faster and more tightly-controlled motions than feedback methods could achieve. Feedback control is the evolution of the control methods of the 70s and includes schemes like adaptive control methods. Feedback control generally makes use of less detailed models, incorporating only the basic setup of the system while disregarding the need to know the exact parameters. The drawback were slower movements and less ability to quickly answer to new movement commands. In contrast to motion control methods, force control methods were based on more direct interaction with the environment. The use of force and torque sensors would detect collisions and exertions of force such as acceleration. These methods rely on an accurate description of the system and the task to be performed when used in a computer-operated system (Whitney, 1987). Still, most industrial robots depended on human expertise to perform the planning and coordination, which was then stored for later playback (Lewis, 2004). Force control was a useful tool

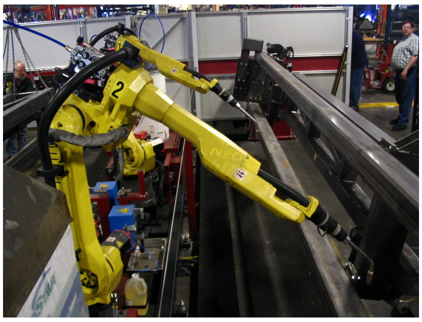


Figure 1.2: A set of six-axis robots used for welding. Picture Wikipedia

for creating teleoperated robots (Uchiyama, 1989). Two important pieces of research were the Computed Torque Method (CTM) (Luh et al., 1980) and the discovery that "the conventional PD feedback servo-loop assures the globally asymptotic stability for point-to-point position control (setpoint control) if the gravity force is carefully compensated.", as stated by Arimoto (1994) (Takegaki and Arimoto, 1981).

1.2.3 1990–2000

Model Predictive Control (MPC) was a major innovation of model-based control that became very popular for a significant number of processes, but less so for the control of RMs (Allgöwer et al., 1999). The Nonlinear Model Predictive Controller (NMPC) however, was found to be suitable for point to point control and showed high efficiency of movement at the cost of significant computation costs. It required detailed knowledge of the system, and manual tuning of costs (reward function equivalent) (Poignet and Gautier, 2000). Force control continued through two main paths: impedance control and hybrid control. Yoshikawa (2000) surveys the state of the art of force control by that time. There is little conceptual difference to the methods of the previous decade, though advances in the theory allowed for a greater number of possible applications.

While there have been advances in the control of RMs outside the field of AI since 2000, they are less interesting with regards to understanding the past of RM control. Methods such as CTM, MPC, NMPC and linear-quadratic controller (LQR) remain at the forefront of robot control. The main improvements are in more powerful computers and efficient calculations (Ajwad et al., 2014; Sciavicco and Siciliano, 2012).

1.3 Robotic Manipulator Control Using Reinforcement Learning: State of the Art

Reinforcement learning is a subset of machine learning focused on an agent exploring an environment and determining a policy for performing actions that maximizes some given reward. RL can to some extent be framed as a different take on the adaptive optimal control methods mentioned in subsection 1.2.1 (Sutton et al., 1992). In terms of control theory, the policy to be optimized is the controller, the environment is the plant, the reward is the cost function, the state is the systems output and the action is the control vector. One of the most significant differences between classical control and RL methods are the latter's ability to function without a priori information of the system model. RL methods can be completely model-free, or they can incorporate a model estimate either supplied by the designer and adapted by the algorithm or completely self taught (Li, 2017). The general version of an RL method is a function of the states S and actions A taken to reach that state. The policy π is the mapping of states to actions with the purpose of optimizing a sum of rewards $R(s, a)$. The goal of any RL method is to revise this policy to reach a "good" or, when feasible, optimal policy. Thus, the sum of rewards is maximized (Kober et al., 2013).

A key difference in RL from other forms of ML is in the exploration of the environment. An RL-based agent is not told what action it should take, but instead is given only a reward based on the action taken. Thus, there is no information gained about the reward from unexplored options, and the addition of uncertainty means that even previously taken actions are not necessarily fully known. Actions may also have long reaching consequences not immediately obvious.

The current field of RL is wide and varied, with some approaches being more applicable for RM than others. Kober et al. (2013) divides RL methods into two groups: *Value-functions approaches* split into dynamic programming methods, Monte Carlo methods and temporal difference methods, and *policy search* methods. Value-function methods are based on calculating the value of each state, and improving that calculation for each iteration. Policy search directly attempts to find and improve a policy.

- Value-functions approaches
  - Dynamic Programming methods
    - *Policy Iteration*
    - *Value Iteration*
  - Monte Carlo methods
    - *SMC*
  - Temporal Difference methods

        * *SARSA*
        * *Q-Learning*
- Policy Search
    - *REINFORCE*
    - *PPO*
    - *DDPG*

A significant amount of past research have been based on value-function approaches. These are however less suitable for RMs due to factors such as a continuous state and action space and high dimensionality. Some approaches leverage the best of both methods with what is called *actor-critic* methods, where the value function, the critic, is used only for policy updates, and not for action selection. A brief look at value-function methods includes State–Action–Reward–State–Action (SARSA), the Q-learning method, its deep reinforcement learning variant deep Q-learning (DQL) (Mnih et al., 2015) and an adaption for continuous action space, Normalized Advantage Functions (NAF) (Gu et al., 2016). Recent policy search methods include guided policy search (GPS) (Levine and Koltun, 2013), deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015) and trust region policy optimization (TRPO) (Schulman et al., 2015). Some methods are based on the stability analysis of Lyapunov functions to create methods safer for physical systems (Chow et al., 2018). Many methods for RL are "deep" methods, methods with one or more hidden layers between the input and the output, such as a neural network. Li (2017) describes this and attempts to explain why that is the case: "Why has deep learning been helping reinforcement learning make so many and so enormous achievements? Representation learning with deep learning enables automatic feature engineering and end to-end learning through gradient descent, so that reliance on domain knowledge is significantly reduced or even removed. Feature engineering used to be done manually and is usually time consuming, over-specified, and incomplete. Deep, distributed representations exploit the hierarchical composition of factors in data to combat the exponential challenges of the curse of dimensionality. Generality, expressiveness and flexibility of deep neural networks make some tasks easier or possible".

A plethora of methods have been designed that can to some degree facilitate the solution of tasks using an RM. A key aspect to any method intended to control an RM is the ability to infer connections without explicitly testing everything. Methods such as PILCO (Deisenroth and Rasmussen, 2011), ME-TRPO (Kurutach et al., 2018), STEVe (Buckman et al., 2018) and MVE (Feinberg et al., 2018) learn the dynamics of the system through exploration. Other methods such as PPO (Schulman et al., 2017) and DDPG function without a model of the system, instead incorporating the dynamics of the systems as a part of the artificial neural network (ANN) mapping state to action. The policy is then improved for each iteration through gradient descent, where the gradient of the policy with respect to the weights of the ANN is used to repeatedly make minute changes such that the estimated

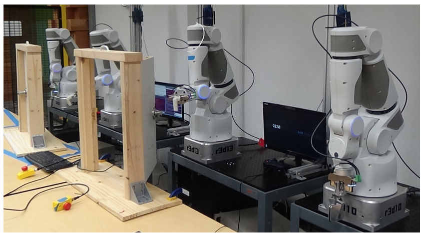


Figure 1.3: A series of RMs learning to open a door, sharing their experience for a faster learning curve. Picture: Yahya et al.

reward is maximized. Gradient decent methods are suited for large solution spaces, and PPO in particular was chosen for its balance of simplicity and sample complexity. In Schulman et al. (2017) it shows good performance with little data.

A key motivation for this work is the question of why RL is necessary. To say that RL is better than classical control, or will replace it in the foreseeable future, would be questionable. There are many problems and applications where the control methods of today are reliable, cheap and effective. In a known environment the current industry standards certainly perform great work. Both robotic control and AI control have greatly evolved over the previous decades, allowing new challenges to be addressed in both fields. The challenge RL seeks to overcome, that today's classical control cannot, is that of flexibility, of adaptability. Many tasks require a human hand simply because there are no current methods that can tackle the amount of varying situations that can be encountered. With the last years' explosion of available data and new techniques, RL is now in a place where these problems can be faced, and hopefully solved. With this in mind, RL is headed towards becoming a useful tool for control of RMs. More and more combined methods employing both RL and classical control are likely to surface in research in the near future.

1.4 An overview of contributions of this thesis

Significant time was spent on a marked survey for the selection of an appropriate RM and software solution that may be used for future research, with minimal prior knowledge and time required to learn the system. The simulated versions of the "WidowX Robot Arm Mark II Kit" (Trossen Robotics), the "Niryo One" (Niryo), and the selected "OpenManipulator-X (RM-X52-TNM)" (ROBOTIS) manipulators were all installed, the significant bugs hindering accurate simulation fixed and the models evaluated. As a result, future work building on this thesis will have a suitable RM with an accurate simulation and know the limitations and pitfalls likely to impede progress.

A physical and simulated valve model was created, for use in modeling subsea intervention tasks. The physical valve modeled features a complete system for accurate real-time information about the valve's position. The valve was designed by the author in collaboration with fellow master student Anders Haver Vagle, and built by NTNU, department of Engineering Cybernetics' workshop. They also provided the parts for the potentiometer and adc, for which the author and Vagle designed a breadboard circuit, and connected to a Raspberry Pi. The collaboration also included writing software incorporating the valve's position to the ROS system.

While not used for the tasks solved in the thesis, a functional visual module was created. This consisted mainly of selecting a camera, the Raspberry Pi Camera V2, and find a software solution that functions with the ROS kinetic software used for controlling the manipulator in real-world a simulation.

The second part of the thesis contribution is in the application of PPO and BC to solve a modelled subsea intervention task in both simulation and real-world. The PPO algorithm was shown to be able to solve the task in a simulated environment and the successful policy was used to control the real-world RM, showing that the experience could be transferred with no adaption.

Additionally, the task was solved using BC with a limited amount of data available to train with. This data was also generated using the real-world RM, requiring the writing of software for recording the operators actions and states of the RM and the valve. Lastly, a combination of BC and PPO was shown to be capable of learning to solve some parts of the task with greatly reduced training time when compared with PPO alone. This combination was in the form of using the policy trained with BC as the initial policy used by PPO. To do this, the BC policy, which was generated with Python 3.5 using Keras as the ANN interface, had to be transferred to the PPO agent that was written for Python 2.7 using Tensorflow.

Chapter 2

Theory

RL control methods differ from classical control methods in that they are not necessarily based on the mechanics and dynamics of the system they control. As such, they, in particular those making use of artificial neural networks, can appear unintuitive from a system control perspective. This chapter will give a brief introduction of basic RL based on Wiering and Otterlo (2012), Chapter 1, before moving on to those particulars that are relevant to the method employed in this thesis. Sections 2.1 and 2.4 are updated versions of the authors earlier work (Vermedal, 2018).

2.1 Brief introduction to RL concepts

Reinforcement learning deals with solving problems of sequential decisions with limited feedback. Contrasted with supervised learning, where the agent is told what was correct, and unsupervised learning, where the agent receives no feedback at all, RL agents receive feedback in the form of a scalar reward. The goal of the agent is to maximize the cumulative reward. The agent does this by calculating/learning a policy of what action to take in each possible state. To go further, some notation is required.

A Markov decision process (MDP) is a fundamental tool in RL. Any problem to be solved is almost always stated as an MDP. An MDP consists of states, actions, transitions and rewards. The set of states S is a finite set of all the possible states of the system. A state characterizes all parameters of the problem that are relevant for the agent. The set of actions A is the finite set of all possible actions. Some actions may not be possible in certain states. The transition function $T(s, a, s')$ describes the chance for ending up in a new state s' after preforming action a in state s. It is a mapping from $S \times A \times S \rightarrow [0, 1]$. Lastly the reward function R denotes the reward of an outcome. It may be dependent only on the state reached, $R : S \to \mathbb{R}$, or the path taken to reach it, $R : S \times A \times S \to \mathbb{R}$. The notation is usually simplified for a problem to only specify the parameters that affects the outcome. MDPs also have the Markov Property, which states that the next state is only dependent on the current state. That is, the past is irrelevant, only the current state and the action taken affect the outcome of each state transition.

As stated earlier, the goal of the RL agent is to maximize the received reward. With the notation of the MDP we can formulated that reward as $\Sigma_{i=1}^{h} r_i$. Usually we want to discount the value of future rewards with a factor $\gamma \in [0, 1]$ and use $\Sigma_{i=1}^{\inf} \gamma^{i-1} r_i$. Lastly the average reward variant $\frac{1}{h} \Sigma_{i=1}^{h} r_i$ may also be used. Given the MDP $\{S, A, T, R\}$ the agent attempts to learn a policy $\pi : S \to A$. There also exists a stochastic variant $\pi(s, a) \to [0, 1]$. Often the problem the RL agents are exploring is a stochastic problem, and the exact outcome of an action is not certain. In this case it is useful to consider the expected reward. In the deterministic case the expected reward is the exact reward. Thus, the same notation may be used for both. Since the aim is to maximize the cumulative reward, the expected future rewards must also be taken into account. With this each state s can be assigned an expected value if a policy π is followed, defining the value function $V^{\pi}(s)$. Here the discounted infinite horizon is used as the model for cumulative rewards.

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{i=0}^{\inf} \gamma^i r_{i+1}\right] \quad \forall s \in S, \tag{2.1}$$

The similar state-action value function (Q-function) which additionally specifies the first action taken before following the policy afterwards can be defined similarly.

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{i=0}^{\inf} \gamma^i r_{i+1}\right] \quad \forall s \in S, a \in A. \tag{2.2}$$

Evaluating these equations require complete knowledge of the future expected rewards that will be received when following π from state s. The Bellman Equation gives a recursive definition the value function, creating a set of nonlinear equations

$$V^{\pi}(s) = \sum_{s' \in S} T(s, \pi(a), s')\Big(R(s, \pi(a), s') + \gamma V^{\pi}(s')\Big). \tag{2.3}$$

The Bellman Equation describes the value when following any policy. An optimal policy π^* will fulfill $V^{\pi^*}(s) \geq V^{\pi}(s) \quad \forall s \in S, \forall \pi$. For simplicity V^{π^*} is denoted V^*. The Bellman optimality equation gives an expression for this optimal value function,

$$V^*(s) = \max_a \sum_{s' \in S} T(s, a, s')\Big(R(s, a, s') + \gamma V^*(s')\Big), \tag{2.4}$$

which can also be phrased in terms of the Q-function as

$$Q^*(s,a) = \sum_{s' \in S} T(s,a,s')\Big(R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\Big). \tag{2.5}$$

Then the optimal policy may be extracted from the value function or the Q-function using (2.6) or (2.7) respectively.

$$\pi(s) = \operatorname*{argmax}_{a'} \sum_{s' \in S} T(s,a,s')\Big(R(s,a,s') + \gamma V^(s')\Big). \tag{2.6}$$

$$\pi(s) = \operatorname*{argmax}_{a'} Q(s,a). \tag{2.7}$$

2.2 Neural networks

This section is primarily based on the book by Zurada (1992). An artificial neural network is at its most simple a function approximator, taking some input and transforming it to some output. The least complex form of an ANN is a perceptron, developed in the 50s (Rosenblatt, 1958). A perceptron takes a number of inputs x_1 to x_n each multiplied with a corresponding weight W_1 to W_n to produce a binary output. An ANN is a network of perceptron stacked in parallel, then layered so that each stack of perceptrons take input from the previous layer (or the input for the first layer) and provides its results as inputs for the next input (or as the final output from the last layer). The layers between the input layer and the output layers are referred to as hidden layers. This is illustrated in Figure 2.1. A modern ANN may use several different activation methods for deciding the output of a perceptron such as Sigmoid, Softmax or Rectified linear units (ReLu). These have different effects on learning rate and are fit for different application. Often a mix can be effective, and the choice of activation function is a mayor part of designing an ANN. One benefit of these common activation methods are that they introduce non-linearity, allowing the ANN to represent non-linear functions.

Artificial neural networks are used due to their ability to approximate any function, a highly desired trait when the design of a successful policy function is unknown. A mathematical definition of the behavior of moving efficiently to navigate to, grasp, and turn a valve is not something that can be easily created. An ANN with the ability to approximate any function given sufficient nodes, combined with RLs ability to refine a policy without prior knowledge of the system or task, combines to solve this challenge.

After deciding on a number of layers, nodes per layer, and activation method, the ANN will transform any input to an output. To make this transformation approximate the desired function, the network must be trained. When training an ANN, the parts of the system that changes are the weights. This is done through backpropagation with the use of a loss

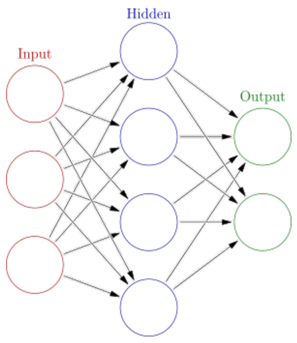


Figure 2.1: An ANN taking 3 scalar inputs, providing 2 scalar outputs and has 1 hidden layer. Picture retrieved from https://en.wikipedia.org/wiki/Artificial_neural_network 22 Nov.

function. A loss function is a measure of how far off target the output was for any given input. This is a whole topic on its own, and a large part of each RL method using an ANN is the design of this function. The important part is that as this function gives some form of measure of how close an output is to being correct, it can be used improve the ANN. This is done using the gradient of the loss function and updating each weight based on this gradient through backpropagation. It uses the chain rule to update each layer in turn, starting with the weights between the last hidden layer and the output, then *propagating backwards* through each layer.

The output of a layer is

$$\boldsymbol{x}^{(n+1)} = g(\boldsymbol{x}^{(n)}\boldsymbol{W}^{(n)}) \tag{2.8}$$

where $g(\boldsymbol{x})$ is the activation function, applied to each element of the vector and $\boldsymbol{W}^{(n)}$ is the matrix of weights attached to the nodes of layer n. For an individual node m, the formula looks like this

$$x_m^{(n+1)} = g(\sum_{i=0}^{k} W_{mi}^{(n)} x_i^{(n)}) \tag{2.9}$$

Each $x_i^{(n)}$ in the formula can be expanded similarly. Backpropagation is then done by calculating the partial derivative of the loss function with respect to each of the individual weights, and adjusting the weight in relation to that value. The exact adjustment varies based on the method used.

2.3 Gradient Descent

Gradient decent is the most common technique used to adjust the weights in the ANN of an RL method. As the ANNs can grow quite large and complex, analytically finding the minimal of the loss function with respect to the weights becomes unfeasible. Instead, the partial derivatives of loss function with respect to the weights is calculated by taking the derivative of Equation 2.9. This then gives the direction each of the weights should be adjusted to improve the results, and by how much. This value however, is only correct at that exact point, and it is not known how large a change would continue to improve the result, before it will worsen it. Therefore, only a short step based on the learning rate of the method is taken, before the derivatives are recalculated and the step repeated. Thus it will iteratively approach the local minima. The choice of learning rate is quite important. If it is too large, the step will overshoot the minima, and may never reach it, while if it is too small it will take an unreasonable amount of steps to reach the minima and take too long to compute for practical uses (Sutton and Barto, 2018; Ruder, 2016).

2.4 PPO

Proximal Policy Optimization is the method employed in this thesis, proposed in Schulman et al. (2017). It is an on-policy method suited for complex continuous state- and action-spaces; exactly what is needed to control an RM. PPO is widely well regarded in research for its potential applications. PPO is often implemented as an actor-critic method, defining an value function to be trained together with the policy function. This is used to create a loss function $L^{VF} = (V_\theta(s_t) - V_t^{target})^2$. When evaluating a new policy, it is compared to the old policy. $R_t(\theta)$ defines the ratio between the new policy and the old so that a measure for the improvement L^{CPI} is defined as

$$L^{CPI}(\theta) = \hat{\text{E}}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t}\hat{\text{A}}_t\right] = \hat{\text{E}}\left[r_t(\theta)\hat{\text{A}}_t\right], \tag{2.10}$$

where $\hat{\text{A}}$ is the estimated advantage function. Maximizing this is stated to lead to excessively large policy updates, so the method proposes several different ways of dealing with this issue. One version is to use trust regions to set restrictions for the new policy, another to adapt the strict restrictions to instead be a penalty, but the main, simplest and most promising trick is to use clipping on this function, but only when the change is too positive, creating

$$L^{CLIP}(\theta) = \hat{\text{E}}\left[\min(r_t(\theta), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{\text{A}}_t)\right], \tag{2.11}$$

with ϵ being a hyperparameter for the method. The complete function for optimizing the policy then becomes

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathrm{E}}\left[L^{CPI}(\theta) - c_1 L_t^{VF} + c_2 S[\pi_\theta](s_t)\right], \tag{2.12}$$

where c_1 and c_2 are hyperparameters and S an entropy bonus.

The method runs through a number of iterations where N actors each interacts with an environment for T timesteps, creating $N * T$ datapoints. These are then used to optimize the policy, before a new batch runs using the new policy, repeating until the agent is finished or simulation stopped. The pseudocode for this training process is described in Figure 2.2.

```
Algorithm 1 PPO, Actor-Critic Style
for iteration=$1, 2, \ldots$ do
    for actor=$1, 2, \ldots, N$ do
        Run policy $\pi_{\theta_{\text{old}}}$ in environment for $T$ timesteps
        Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
    end for
    Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \leq NT$
    $\theta_{\text{old}} \leftarrow \theta$
end for
```

Figure 2.2: Pseudocode for the PPO algorithm from the paper where it was proposed, (Schulman et al., 2017)

The result is a method that avoids overly optimistic adjustments in order to avoid the inaccuracies introduced by moving further away from the old policy, and thus the parameters under which the current data was gathered.

For this thesis, the configuration following in Table 2.1 was used.

Learning rate	0.01
Discount factor	0.99
Entropy coefficient	0.001
Cliprange	0.1 decaying to 0.01 over 10 000 steps
Epochs	10
Batch size	512
Minibatch size	0
Number of hidden layers	2
Nodes per layer	400
Policy and value networks	Separate

Table 2.1: Hyperparameters for PPO

Algorithm 2 Abstract of behavioral cloning

```
    Collect a set of trajectories demonstrated by the expert $\boldsymbol{D}$
    Select a policy representation $\pi_\theta$
    Select an objective function $\boldsymbol{L}$
    Optimize $\boldsymbol{L}$ w.r.t. the policy parameter $\boldsymbol{\theta}$ using $\boldsymbol{D}$
    **return** optimized policy parameters $\boldsymbol{\theta}$
```

Figure 2.3: The basic algorithm of behavioral cloning, from Osa et al. (2018) p48.

2.5 Behavioral cloning

The presented theory in this section is based on Osa et al. (2018) and aims to give a brief overview of the basis of imitation learning (IL) so that the use of behavioral cloning in this thesis can be understood without prior experience with IL.

The basic concept of imitation learning is that a human expert demonstrate a behavior, and a robotic system learns a policy to imitate this behavior. The system must learn to extrapolate from the shown behaviour to tackle minor differences from the demonstrated behaviour, but will almost certainly fail if it encounters something outside of the trained upon states. It is closely related to supervised learning with some crucial differences. Relevant here is the high cost of resetting the state and restarting predictions causing a far smaller dataset to be available.

Behavioral cloning is a subset of IL which focuses on the direct mapping of states to actions. A dataset $\boldsymbol{D}$ demonstrating the desired behavior is collected, and the algorithm, shown as pseduocode in Figure 2.3, is initialized with a policy representation π_θ, here a neural network equal to the ANN used by PPO and a loss function $\boldsymbol{L}$. The loss function $\boldsymbol{L}$ is then optimized with respect to the policy parameters θ as in the RL case. The key difference between BC and RL is that the loss function $\boldsymbol{L}$ is not based on any reward, but instead compares the policy output with the experts action. The loss function employed in this thesis is the mean squared error (MSE) function chosen for its simplicity and being generally recommended for BC.

Chapter 3

Setup and implementation

To accomplish this thesis' goal of testing the performance of a state of the art RL method on a real-world manipulator, the real-world manipulator, the environment for the task and a simulation of both are needed. The setup is based on the task of turning a valve on a subsea operation. A manipulator with an associated simulation solution had to be acquired and installed, and a model of the task environment had to be build for both the manipulator and the simulation. This chapter describes the process, the implementation of the RL agent and the details of the task to be solved.

3.1 Manipulator and valve setup

The setup consists of two key parts: The RM and the valve. The two are fastened to a single wooden base at 30 cm apart measured center to center.

3.1.1 Manipulator

The choice of RM for this thesis had three primary criteria. The manipulator should be robust and reliable, it should come with software for simulation in an accurate physics simulator allowing for interaction with custom objects, and it should be affordable. There are many different RMs available for purchase, ranging for simple toys to multi-million dollar industrial robots. Neither end of this scale fit the requirements, and the available choices were narrowed down to "WidowX Robot Arm Mark II Kit" (5 DoF, $1499,95) (Trossen Robotics), "Niryo One" (6 DoF, 1799,00€) (Niryo), and "OpenManipulator-X (RM-X52-TNM)" (4 DoF, $1199,00) (ROBOTIS).

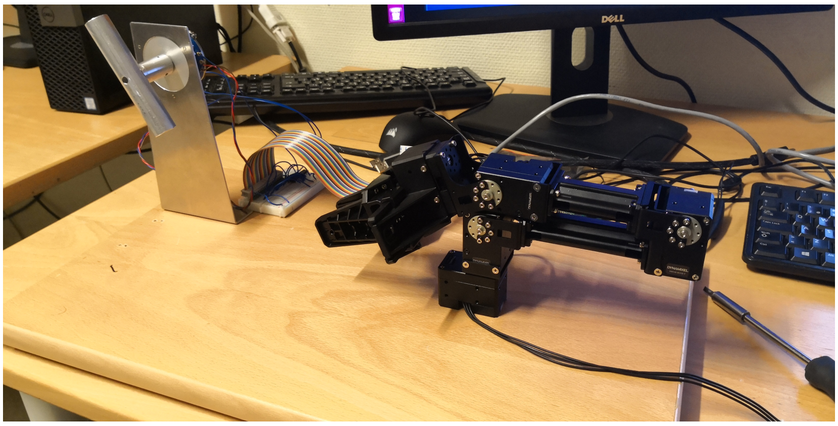


Figure 3.1: Full setup showing the robot, the valve and the base everything is mounted on.

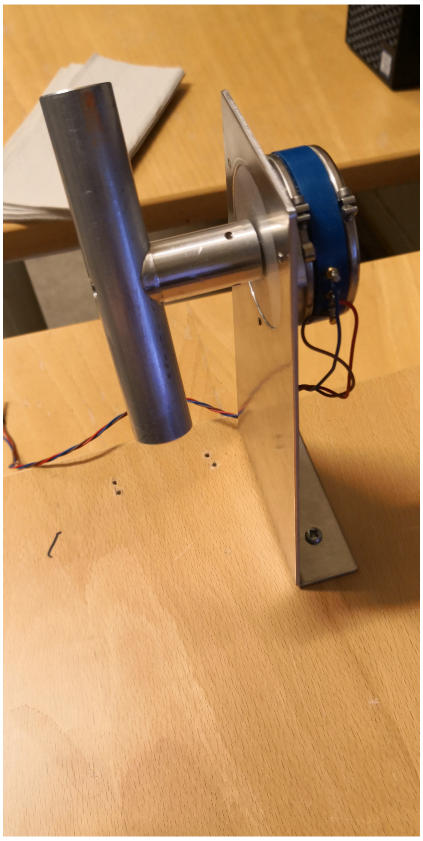
Having established that all were in an acceptable price range and feature similar hardware, the choice came down to their simulations. All three uses ROS Kinetic and Gazebo 7 for their physics simulation, so there were no choice in simulator. The final decision was made based on the models each used in the simulation. The WidowX model did not include the wrist rotor, and the Niryo model lacked the end-manipulator, immediately ensuring the simulation would differ from reality. The WidowX could however be assembled with that joint, losing 1 DoF. Both the WidowX and the Nyrio One had inaccurate or unrealistic parameters describing their masses and inertia, and after the author's supervisor inquired with both companies it was clear any improvements would not be made in time for use in this thesis. The final manipulator, OpenManipulator-X has a far more detailed model description and as such was chosen for this thesis.

The OpenManipulator-X features five XM430-W350-T motors, giving 4 DoF plus a 1 DoF parallel gripper. It has a reach of 380 mm and can hold a payload of 500g. The RM arrived in parts after ordering, but assembly was easily done using the provided manual. Joint 1 allows the robot to rotate around its base. Joint 2 - joint 4 can move forwards or backwards in a single plane dependent on the position of joint 1. Lastly, joint 5 controls the opening and closing of the gripper. The gripper is not controlled by the agent in any of the tasks used in this thesis, giving a total potential of 4 DoF. X-position, Y-position and Z-position in the work space, and the tilt of the gripper. When joint 1 is locked, this is reduced to 3 DoF in a plane. In the coordinates of the plane the X-position, the Z-position and the tilt of the gripper can be controlled. With 2 DoF using joint 2 and 3 the tilt is dependent on the position, and any point can only be reached by a single configuration.

3.1.2 Valve

The valve consists of two hollow cylinders fasted together in a T-shape, mounted on a thin plate at 18 cm above the wooden base. The top of the T points in the positive Y direction. The valve rotates freely around the Y axis. Attached to the valve is a potentiometer measuring the position of the valve. This is connected to an ADC, ADC0844CCN, which again is connected to a Raspberry Pi 3B (RP). The RP must then establish an ROS topic and publish the position of the valve, at a rate of 100 Hz. This requires the RP to have an installation of ROS Kinetic. The OM platform includes a guide to installing ROS Kinetic on the RP. However, for the 3B and 3B+ this have some issues. The provided guide works for this simple setup, but leaves the RP unable to install additional ROS packages, which are required to use the RP camera with ROS. Multiple guides to solving this issue exists on the internet, however none worked for the author. Finally the RP image provided by Ubiquity Robotics proved to be a stable solution that worked out of the box (Ubiquity Robotics).

The code to publish the reading from the ADC is quite simple, consisting of two main parts. The first reads the output of the ADC connected to the GPIO pins of the RP following the procedure given by the ADCs datasheet. The other parts creates an ROS node and publishes the data from the ADC at a specified rate, limited to an aproximal maximum 1000 Hz by the ADC, to give the agent up-to-date information of the valve. As the control of the robot uses a 1s delay after each movement, before it reads the environment, a rate of 100 Hz was deemed sufficient. If processing resources are scarce, this can instead be implemented as an ROS service where the RP is polled for the valves position when a reading is required.


(a) Valve model

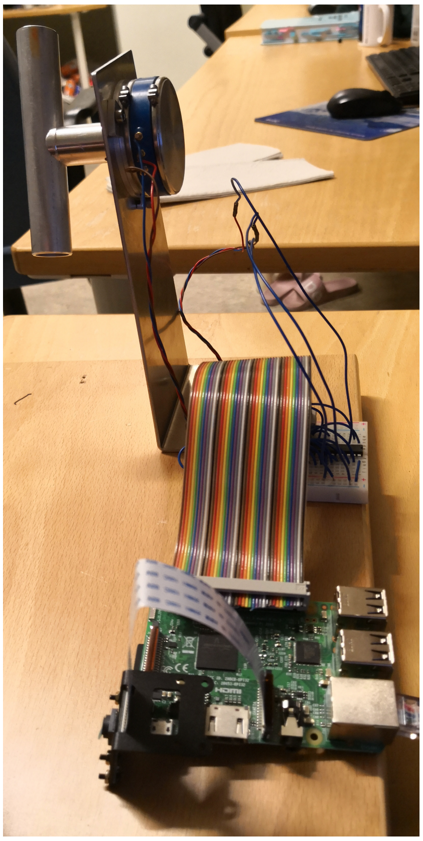
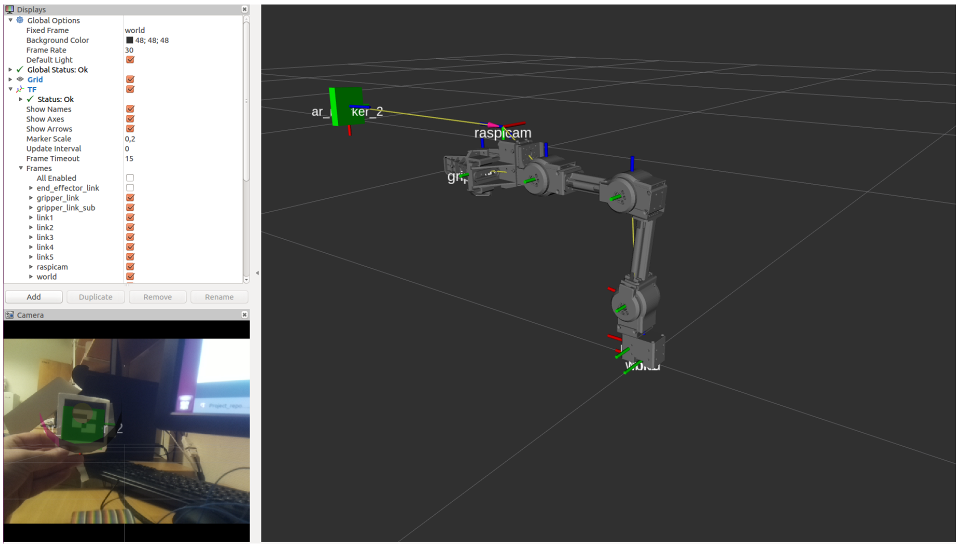

(b) Vavle, potentiometer, adc and Raspberry Pi.

Figure 3.2: The valve model and associated components

3.1.3 Camera

The RP camera can be used to identify the location of the valve. This can be done in a multitude of ways, including RL, but for this thesis the simple solution of identifying known markers was used. The ROS package "ar_track_alvar" is an easy to use solution that directly publishes the coordinates of any identified markers in the cameras view to an ROS topic. Using the setup from the OM platform, these coordinates are given relative to a camera mounted on top of the robot as shown in Figure 3.3 and 3.4. With an offset to account for the relative position of the gripper from the camera, this can be used to get a vector from the gripper to the valve. Used in conjunction with the potentiometer measuring the position of the valve and knowing where on the valve the marker is, this can give a vector from the gripper to the goal position for a secure grip. This is set up to showcase the possibility of doing this for future work, and is not used in the tasks performed in this thesis.



```
Displays
▼ Global Options
    Fixed Frame            world
    Background Color       ■ 48; 48; 48
    Frame Rate             30
    Default Light          ✓
▶ ✓ Global Status: Ok
▶ Grid                     ✓
▼ TF                       ✓
  ▶ ✓ Status: Ok
    Show Names             ✓
    Show Axes              ✓
    Show Arrows            ✓
    Marker Scale           0,2
    Update Interval        0
    Frame Timeout          15
  ▼ Frames
      All Enabled          ☐
    ▶ end_effector_link    ☐
    ▶ gripper_link         ✓
    ▶ gripper_link_sub     ✓
    ▶ link1                ✓
    ▶ link2                ✓
    ▶ link3                ✓
    ▶ link4                ✓
    ▶ link5                ✓
    ▶ raspicam             ✓
    ▶ world                ✓
```

Figure 3.3: Screenshot showing how the raspberry pi camera identifies a maker within its field of view and calculates the relative vector in the work space.

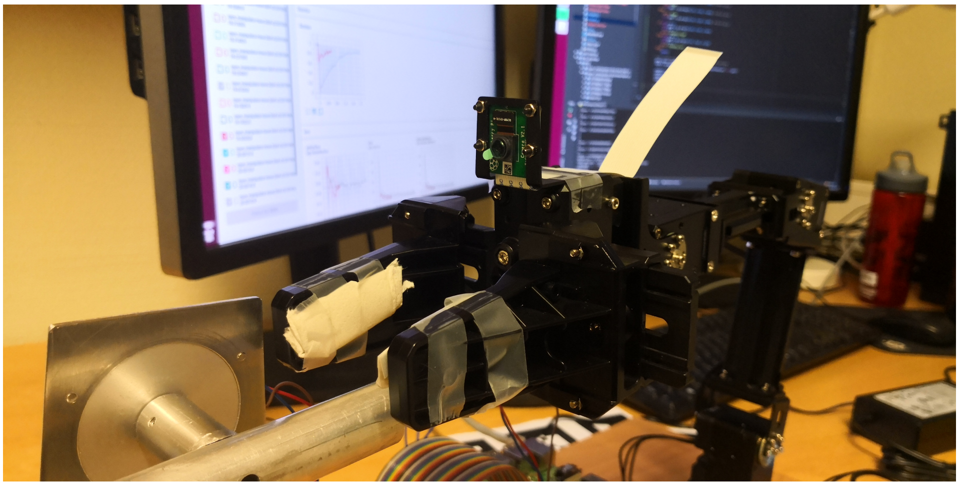


Figure 3.4: Picture showing how the Raspberry Pi camera can be mounted on the RM

3.2 Simulation

Training an RL agent can take a long time, and is often unfeasible to do with real world robots. For this reason, a simulation of the robot is required. Key advantages of simulation are the ability to simulate faster than real-time, avoiding damage to the robot and the environment, and complete control of each episode. For RMs, Robot Operating System (ROS) is a popular operating system for control of robots. It is compatible with the physics simulation software Gazebo, giving similarity between simulation and real world control.

3.2.1 ROS

Robot Operating System is an ecosystem for developing and running control systems for robotic systems. It provides a basis where small or large modular packages can be installed and used for further development. ROS itself does little but facilitate the communication between various nodes. Each node is a process that perform the actual tasks. Nodes communicate through topics and services. A topic is a buss that anyone can publish or subscribe to. Each subscriber receives everything that is published, with no information about who published it. Services are one on one, where a node can offer a service, and other nodes may request an action to be performed. They can then get information about how far along the action is, and when it is complete.

Gazebo

Gazebo is a robotics simulator that through the gazebo_ros_pkgs metapackages can be used to simulate a real robot to be controlled using ROS. The simulation gets the robot configuration through an URDF file, detailing each of the joints and links of the robot. It detects collision between objects, and may simulate a camera or depth-sensor that sees the scene in question. It is currently not well suited for RL using episodic training, as it lacks functionality to reset the joint states of a robot without forcing a restart of all other nodes in the ROS environment.

MoveIt

MoveIt runs on ROS and takes on the task of creating the control signals to make the robot reach the desired goal. The goal can be specified in joint- or task-space, and can be a single point or a set of points making up a trajectory. The goals can be specified in terms of the angles of the joints or XYZ coordinates, in addition to velocity and effort of the motors. Lastly, the points can have a specified time they should be reached. This project wishes to let the RL agent have control of the robot, so only the single point version is used. It would be desirable to be able to control the motors of the robot directly without the use of MoveIt, but this was not implemented at the time this project was completed.

3.2.2 OpenManipulator

OpenManipulator (OM) is platform facilitating the control of an OpenManipulator-X RM. It also includes a complete setup for simulating the robot using Gazebo in place of the physical robot. At the time of writing there are several difference in regards to how the OpenManipulator-X RM and its simulation can be controlled, forcing the use of MoveIt to achieve consistency. The simulated robot contains mass and inertia information that alongside the joint configuration appears to accurately represent the physical RM. Teleoperation in the form of a keyboard or a joystick is also implemented by the platform. It is through an approximation of this the agent control the RM.

The OM platform forces several restrictions to the software environment. It uses ROS Kinetic, which means Ubuntu 16.04 must be used, and Gazebo 7. For the programming in python, ROSPY, the ROS python library, requires that python 2.7 is used, and to maintain the same environment for all scripts interacting with each other, they must all be written for python 2.7. With those restrictions in place, the OMs setup guide can be followed, installing the required programs and ROS packages. After the complete setup is finished, the simulated robot appears in Gazebo and can be controlled. The simulation at this point suffered from unstable control and a faulty gravity simulation. This was solved by introducing a set of missing "gazebo_ros_control" variables found in an unused configuration file in the "open_manipulator_simulations" package.

Running the simulation gives a maximum real-time speedup factor of 2.5 on the hardware used in this thesis. The simulation was also installed on a powerful server "g2.8xlarge" provided by Amazon Web Services (Amazon). On this server a maximum real-time factor of 3 was achieved. As Gazebo is limited to utilizing a single core, it is not expected that any notable higher real-time factors are achievable on consumer-grade hardware.

3.3 Software setup

Having completed the physical setup and the installation of the simulation a solution giving the PPO algorithm control of both manipulators was needed. The solution should allow the agent to control either, with no difference in configuration. The OM platform was created with this in mind, and the same ROS topics and services are used, with the switch between simulation and real-world being performed when launching OMs MoveIt controller.

The framework of Open AI GYM, (Open AI), was used to connect the agent with the OM platform. GYM is often used in research to benchmark RL methods, as it offer multiple default environments ranging from introductory problems to video games to robotic manipulators. As such, many implementations of RL methods are made to interact with a GYM environment, and interpreting the simulation as a GYM environment takes advantage of this to give a quicker and easier simulation setup.

A functioning GYM environment must implement a minimum three functions: make, step and reset. The make function sets up the environment, which in this case entails generating the ROS nodes for communication with the simulation and reporting the parameters the agent needs to know: the number of state, the number of actions and whether these are discrete or continuous. The step function takes an action from the agent, processes that to a MoveIt command and waits for the movement to be completed. It then observes the environment, recording the state variables, and calculates the reward. The reset function resets the simulation, then observes the environment similarly to the step function. As described in subsection 3.2.1 resetting the environment is not a simple operation. To solve this an automatic reset function was written for the simulation. The function attempts to directly set the joint positions of the manipulator and the valve to the specified initial position. In several cases such a direct approach would not work, as all joints attempts to reach their goal simultaneously and the end point of the manipulator would get stuck on the ground, on the robot itself, or on the valve. Multiple exceptions were written handling each case by moving the robot through a fixed set of goals, circumventing whatever was blocking it.

The PPO implementation used is based on an implementation by Anjum Sayed (Sayed). It was used with only minor adaptions to work for the specific environment used and to work for python 2.7, as it was already made to interact with a GYM environment.

3.3.1 Behavioral cloning

For capturing the demonstration data used for behavioral cloning, a framework for controlling the RM using a keyboard, recording the RM and control signal, was created. The code accepts input from the expert operator in the form of a keyboard input corresponding to an action. The action is then carried out using the MoveIt package. After the movement is completed the state is captured and a new action may be input. Each state is saved with the accompanying action, capturing the complete trajectory until the operator deems the gripper to be in he correct position. After completing the task a sufficient number of times, the dataset is fed to the behavioral cloning system, which learns a policy copying the shown behavior as described in section 2.5. Finally the weights and biases of the ANN are saved, and can be loaded as the initial configuration for the policy ANN used by the PPO algorithm.

This setup can be used both for capturing expert demonstrations on the simulation and the real-world manipulator. There is little to no time saved by using the simulation when the real-world version is available, so that was used for all demonstrations. The operator is presented with the same information that is available to the agent, with the addition of being able to see both the robot and the valve, and positions the robot at the randomly selected goal position.

3.4 Tasks

The overall motivation for this thesis is to approach a complete solution where RL is used by RMs for subsea operations. This work focuses on the task of turning a valve. The task have been broken down into two key parts. Approaching the valve and getting a grip, and performing the motion of turning the valve. Here we consider the two parts of the task separate and train a policy for each part. The RM is assumed to be initialized somewhere close to the valve, in a position it can reach it. The first policy is then one which can reach the valve. When this policy completes its goal, the gripper is engaged and a grip is achieved. Then the second policy, which is assumed to start from an initial position of gripping the valve, performs the motion of turning it.

3.4.1 Choice of actions

There are three different methods that may be used for controlling the simulated robot.

- 1. Through a series of ROS Gazebo topics, the reference value for the PIDS controlling the joints can be set directly by publishing a message. This method is well suited for an RL approach, giving the agent the maximum amount of control. However, this method of control is only present for the simulated robot, and can not be used for the physical model.
- 2. Through an ROS MoveIt service the robot can be controlled as described in subsection 3.2.1. This can then be used to let the agent set a increment to one or more joints, or directly set the value of the joints. If desired it can also be used to give directions in XYZ-coordinates.
- 3. OMs prebundled software. This allows for teleoperation through keyboard, through preset or randomized poses with RViZ or using a GUI. This method is useful for testing the simulation, but is effectively a selection of user interfaces for method 2.

As the goal of the simulation is to learn a policy for operating the real manipulator, the same control method must be used for both, which forces the choice of using the MoveIt service.

The main advantage of an RL method over classical control methods is its adaptability, and as such the choice was made to move the robot one move at a time rather than generating a complete trajectory. For the sake of consistency a delay was added after each step, allowing the robot to come to a stop before the state was observed. This control method is quite slow with even small movements taking almost a full second to complete, but allows the policy to be generated without taking into account velocity, acceleration and time. To further increase stability and decrease complexity the control signal was made discrete, so that at each step the policy outputs a number between 0 and $2n$, where n is the number of joints in use, and is interpreted as one of the following commands.

- 0 - Joint1 increase angle
- 1 - Joint1 decrease angle
- 2 - Joint2 increase angle
- 3 - Joint2 decrease angle
- 4 - Joint3 increase angle
- 5 - Joint3 decrease angle
- 6 - Joint4 increase angle
- 7 - Joint4 decrease angle

The delta of each movement is set to $\Delta = 0.03$ radians. The choice was made to balance fine movements when near the objective with the larger distances needed to approach it. This gives a minimum step count of 5-15 for most configurations using 2 DoF, allowing each episode to be completed rapidly.

The robot is capable of 4 DoF. Joint1, the rotational joint at the base allowing an almost complete 360°rotation of the robot is locked for three reasons. In the event that the robot is not oriented towards the valve, correcting the position of joint 1 is independent of all other joints and is far simpler to solve with a separate method in a preliminary step before the tasks described in this thesis are begun. The second reason is to simplify the tasks and increasing the likelihood of the agent being able to find a good policy. Thirdly this drastically decreases the likelihood that an exploring agent crashes the robot into the valve.

After the agent have issued a command, the robot is allowed time to reach the new position and stop before a new command is issued. Initially, this was implemented by monitoring the state of the joints, and waiting for them to reach within a near margin of the target, $\epsilon = 0.001$. However, this lead to a strange case where the path MoveIt creates to reach the next point would sometimes first visit a point in the far opposite direction of the target relative to the robot. To avoid this, a simple time delay was used instead, with $t_{step} = 1s$.

3.4.2 Reach Valve (RV)

When reaching the valve there is one key question. Where is the valve in relation to the gripper. This is achieved by having the valve in a known location, and using kinematics to find the location of the RMs gripper, with the manipulators base as the origin. In reality the location of the valve would likely not be known, and must be located. This works considers that a separate task, which a visual module would solve and provide coordinates for the RL agent. An experimental setup for this was made to check the feasibility of further work in this direction, and can be found in subsection 3.1.3.

The reward function is based on whether each step decreased the distance from the gripper to the goal or not, with an additional reward when the goal is reached and a punishment if the robot exits the work area. This is intended to teach the agent directly that moving closer to the goal is good, instead of forcing it to infer it from an increasing reward the closer it gets. The punishment from leaving the work area stems from the practical concern of the physical robot being damaged, and prevents the robot from simply learning to end the episode as quickly as possible by crashing into the ground. The reward function is included in appendix A.1

The state variables, the input to the ANN policy function, consists of n joint states plus a vector [X,Z] giving the distance from the gripper to the goal, where n is the number of joints used. In most cases 2 DoF is used, with joint 2 and joint 3, or joint 3 and joint 4 being active. With 3 DoF joint 2, joint 3 and joint 4 are used. This requires greater care when used, as the robot can become stuck in positions where it is difficult to reset using preset commands.

The following setup was used when solving this task. The robot starts in a known, static initial configuration a few centimeters away from the valve. The valve is randomly set to one of three positions. Horizontal with the gripable portion pointing towards the robot, or vertical with the gripable portion pointing either down or up. Relative to the robots base this results in the following target coordinates. The Y portion of the coordinates are ignored, as the robot is given no control over the rotational joint.

- Horizontal, pointing down: [X = 0.270, Z = 0.135]
- Vertical, pointing toward: [X = 0.233, Z = 0.169]
- Horizontal, pointing up: [X = 0.270, Z = 0.204]

Joint 2 and joint 3 are used, giving the robot a wide reach, easily able to reach any of the given targets coordinates. This gives a state of (joint 2 position, joint 3 position, X, Z) and an four possible actions (Increase joint 2, decrease joint 2, increase joint 3, decrease joint).

3.4.3 Turn Valve (TV)

With the initial position of gripping the valve, and keeping the gripper locked for the entire operation, this task focuses on not losing the valve while also rotating it to a goal position. This tasks uses the position of the valve as feedback. This feedback may not be available for all real world applications, and may have to be supplemented or replaced with visual feedback. That is considered outside the scope of this thesis, where it is assumed that the rotational position of the valve is available.

The reward function is based on improving the position of the valve. A reward is given on each step where the valve is in a new best position, measured as being closer to goal position than any other previously attained position, proportional to the improvement. An extra reward is given when the valve reaches the goal position. Lastly, a punishment is given when it is determined that the gripper is no longer gripping the valve.

The state variables, the input to the ANN policy function, consists of n joint states plus the position of the valve, where n is the number of joints used. The n joint states are as in the task of reaching the valve described in subsection 3.4.2. The position of the valve is normalized such that the starting position is 0, and the goal position is 1.

Attempts to solve this task revealed that the simulated robot and valve interaction was far from realistic, with major problems of maintaining a grip even while not moving. When a grip was maintained, movements struggled to be executed, with the grippers being unable to slide along the grasped valve and instead blocking the robot completely. Lastly resetting the robot was often difficult to do automatically, lacking a proper software reset to initial conditions. On the basis of these difficulties, a simplified task was designed.

3.4.4 Turn Valve Simplified (TVS)

Instead of assuming the initial position of gripping the valve, the robot starts with the gripper closed, in the vicinity of the valve. The robot is then to attempt to push the valve with the gripper to rotate it to the goal position. The reward function as well as the state variables remains as in the non-simplified task and is included in appendix A.2

When solving this task, the robot was initialized to known, static position. Joint 2 and joint 3 were used, giving a state of (joint 2 position, joint 3 position, valve position) and an four possible actions (Increase joint 2, decrease joint 2, increase joint 3, decrease joint 3).

Chapter 4

Results

After completing the setup for the simulation, the manipulator, the RL agent and the recording of the operator demonstrations everything was ready to begin attempts at solving the tasks.

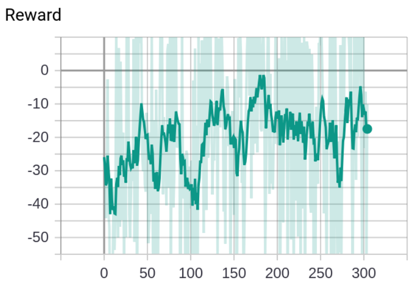
4.1 Reach valve

Several different approaches were made for solving the RV task. The first was to let the agent learn a policy for solving the task with no prior knowledge or training. Afterwards the task was attempted using both pure BC and a combination of RL and BC

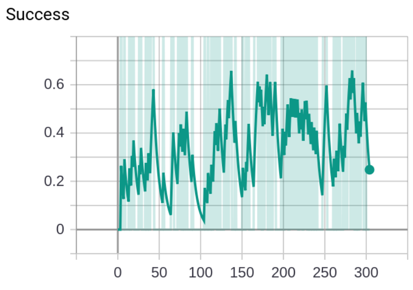
4.1.1 Reinforcement learning

For a reference a completely untrained PPO agent were given control of the manipulator. Figure 4.1 shows 300 episodes completed in 3 hours and 2 minutes. The agent begins with a success rate of 25% mostly consisting of reaching the closer horizontal goal position and slowly approaches 33% near the end.

Then the RL agent employing PPO was given time to learn the task. The results of this is shown in Figure 4.2, showing the moving averaged results in solid gray, and the exact results in partly transparent gray. Graph 4.2b shows that the algorithm quickly started improving its success-rate, and by episode 1 000 averaged above 80%. The graph goes on to show that the success-rate altered between brief periods of 100% success and longer stretches of approximately 90% success. The total real-world runtime for the 2 000 episodes were 13 hours and 2 minutes, an average time of 23.5 seconds per episode.

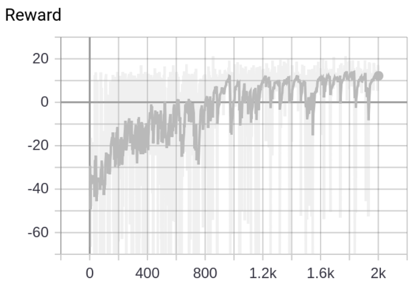


(a) Smoothed reward per episode

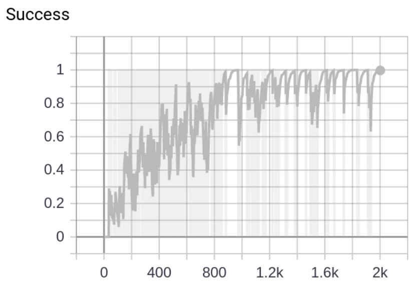


(b) Smoothed binary success per episode

Figure 4.1: RV task performed by PPO with no training, on the real-world robot



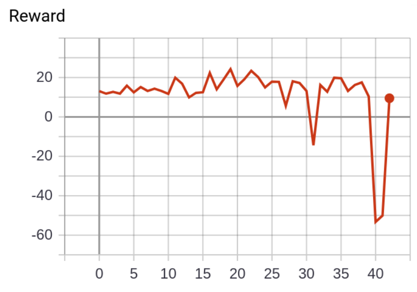
(a) Smoothed reward per episode



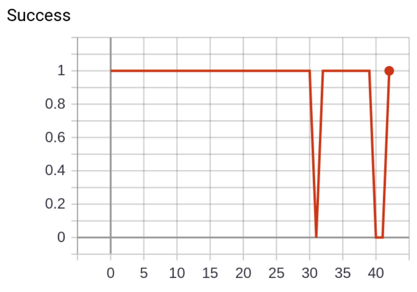
(b) Smoothed binary success per episode

Figure 4.2: RV task performed by PPO with a randomly initialized policy, in simulation.

This policy was then employed on the manipulator, with the results shown in Figure 4.3. Graph 4.3b shows a total of 2 failures over 43 episodes



(a) Reward per episode

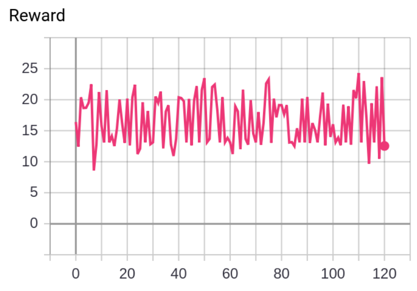


(b) Binary success per episode

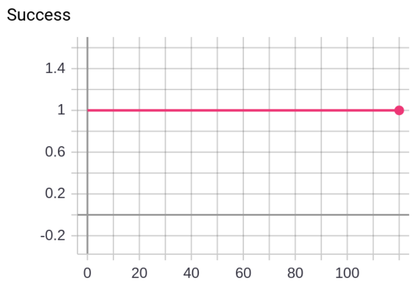
Figure 4.3: RV task performed by PPO with only simulated training, on the real-world robot

4.1.2 Behavioral cloning

The second approach to solving the RV task was done with the use of BC to generate a solid policy accurately copying the expert behavior. For this, the BC was run with 10 epochs. This policy was then used the control the manipulator, achieving a 100% success rate over 120 episodes



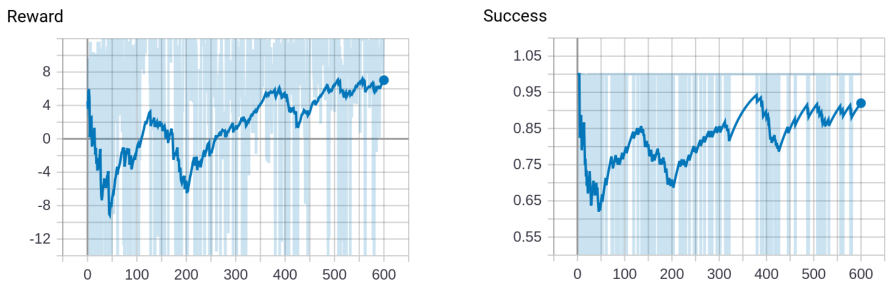
(a) Reward per episode



(b) Binary success per episode

Figure 4.4: RV task performed by PPO with simulated training initialized with BC using 10 epochs, on the real-world robot

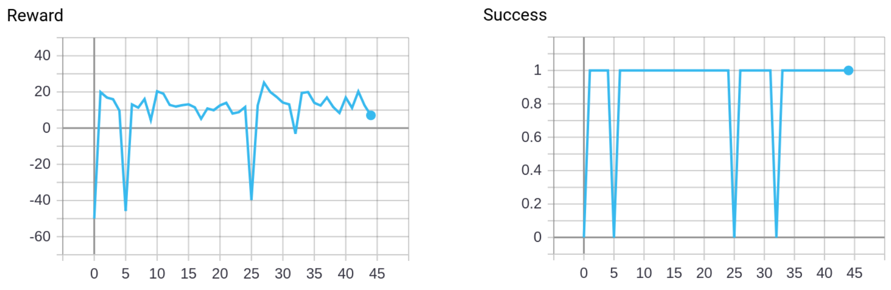
The second BC based approach was to first employ BC to find an initial set of weights and biases for the policy ANN. The ANN approximating the value function remains randomly initialized. Figure 4.5 shows the moving average of the reward (a) and success rate (b) per episode. Note that due to the way the smoothed graph is made, a few successful episodes in the start makes it appear that the initial success rate is 100% and the falls to 65%. This is not the case, it was a lucky start, which took a few episodes to resolve into a meaningful average. The policy improves over 500 episodes in 3 hours and 5 minutes to reach an average success rate of 90%.



(a) Smoothed reward per episode (b) Smoothed binary success per episode

Figure 4.5: RV task performed by PPO with a policy initialized using BC with 1 epoch, in simulation

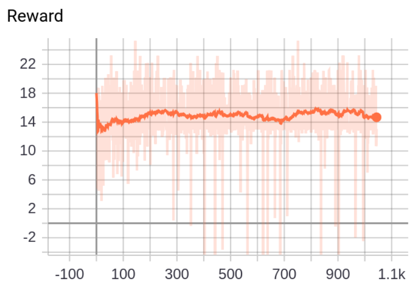
The result of this combined approach was then used to control the manipulator, seen in Figure 4.6. Graph 4.6b shows that the policy failed 4 times over 44 episodes.



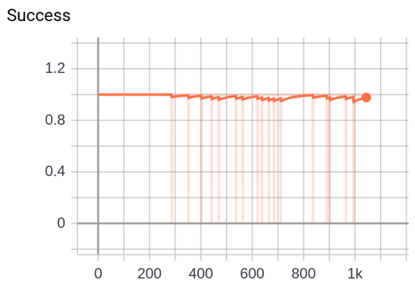
(a) Reward per episode (b) Binary success per episode

Figure 4.6: RV task performed by PPO with simulated training initialized with BC using 1 epoch, on the real-world robot

Lastly, the PPO algorithm was used with the policy from running BC with 10 epochs, that had achieved a 100% success rate, as the initial policy. The results with this as the initial policy can be seen in 4.7. Graph 4.7b shows that the initial policy generated by cloning the behavior of an expert operator on the real-world robot achieves a 100% success rate in the initial episodes in the simulation. After a while, a few failed episodes are introduces and the final 200 episodes have an average success rate of 97%.



(a) Smoothed reward per episode



(b) Smoothed binary success per episode

Figure 4.7: RV task performed by PPO with a policy initialized using BC with 10 epochs, in simulation.

4.2 Turn valve simplified

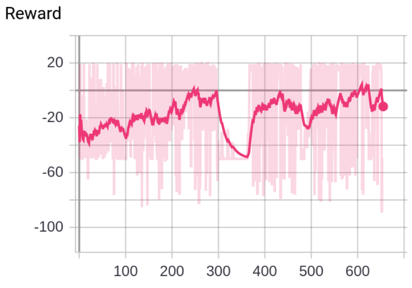
The task of turning the valve was, as mentioned earlier, more difficult to simulate and a simplified version of the task was designed. Even so, there were several difficulties with the interaction between the valve and the robot, and especially with resetting to initial conditions.

4.2.1 Reinforcement learning

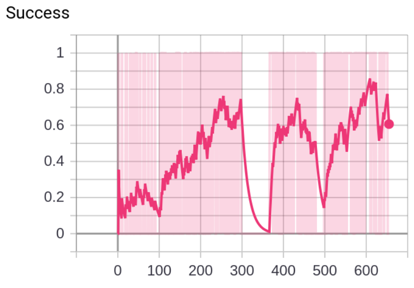
The first attempt at solving the task involved turning the valve a full 90°from a upright position, to a horizontal position. The results of this can be seen in Figure 4.8. After some initially promising results two problems emerged. First, at episode 300 a string of failures can be seen. This was caused by a fault in the reset function, causing the valve to be stuck in a position the RM was unable to affect. Secondly, the robot was not able to fully turn the valve while in contact, but had to resort to throwing the valve the final 30°. This was undesired, as it is far removed from the behavior present in a manipulator grasping a valve.

The second attempt modified the goal position to 60°such that the robot could manipulate the valve into the position while maintaining contact, and expanded the reset function to cover more possible positions. Unfortunately the Gazebo and MoveIt combination remains unsuited to this, and several difficult to diagnose behaviors occurred requiring constant monitoring and human intervention. In particular, the manipulator could force its closed grippers around the valve, but was unable to free itself. Gazebo includes functionality for applying a brief force to a given link in a specified direction which was able to free the gripper, but this functionality have no API forcing the use of the GUI to employed the force.

Figure 4.9 shows the second attempt, where it can be seen that the agent steadily improves

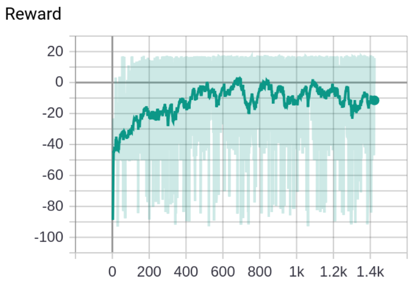


(a) Smoothed reward per episode

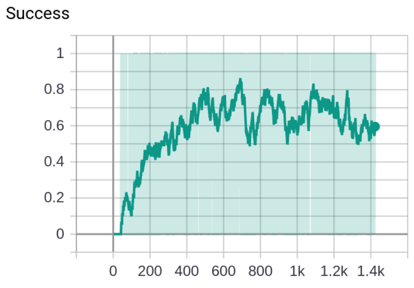


(b) Smoothed binary success per episode

Figure 4.8: TVS task performed by PPO with a randomly initialized policy, in simulation.



(a) Smoothed reward per episode



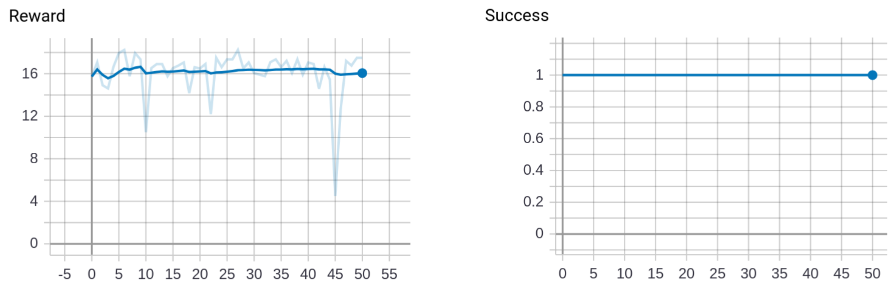
(b) Smoothed binary success per episode

Figure 4.9: TVS task performed by PPO with a randomly initialized policy, in simulation.

its performance. The total training time was 15 hours, with a real-time factor of 1.6 for a total of 22.4 hours of simulated training.

This policy was then used to solve the task with the real manipulator, the results of which can be seen in Figure 4.10. Graph 4.10b shows that the policy managed a 100% success rate over 50 episodes. Graph 4.10b shows that the performance was in general similar in all episodes, with the exception of episode 45 where the total reward was 4.2 instead of the average 16. Observing the policy, it had learned two clearly distinct methods of solving the task. The first method was to move the gripper so it touched the middle point of the valve, where the two cylinders meet. Then it would move upwards while pressing forwards, rotating the cylinder with it. The second method was the more traditional method of moving the gripper to touch the upper part of the valve, then moved in what resembled an arc to push the valve to turn.

The real-world valve was far easier to rotate, and the real-world dynamics of the interaction between the valve and the gripper was more likely to have to gripper deflect of the valve and continue moving, while in the simulation the same movement would result in the



(a) Smoothed reward per episode (b) Binary success per episode

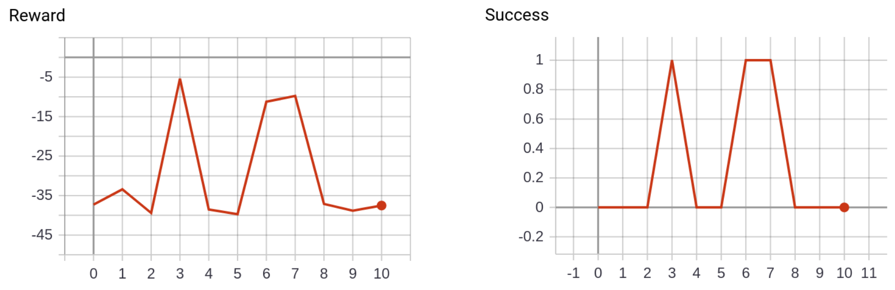
Figure 4.10: TVS task performed by PPO with a randomly initialized policy, on the real-world robot.

gripper being brought to a halt by the valve.

4.2.2 Behavioral cloning

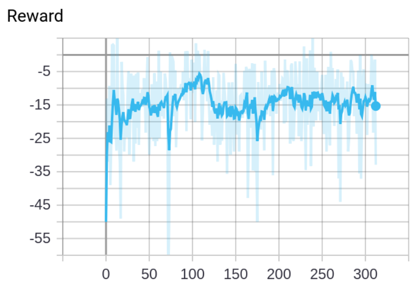
Like with RV, BC was used to attempt to solve the TVS task. 20 samples of the operator demonstrating turning the valve with the second technique described above was collected. The lower dataset size was chosen as there were no variation between runs. After BC used the dataset to learn a policy, this policy was set to control the manipulator. Figure 4.11 shows the results of 10 runs, of which 3 were successful. Graph 4.11a shows that the successful episodes were completed near the very end of the episode.

This policy was then used as the initial policy for PPO, which trained in simulation. As seen in Graph 4.12b, only 1 episode out of 312 were successful, and Graph 4.12a shows that the policy underwent no measurable improvement. When observed, the manipulator

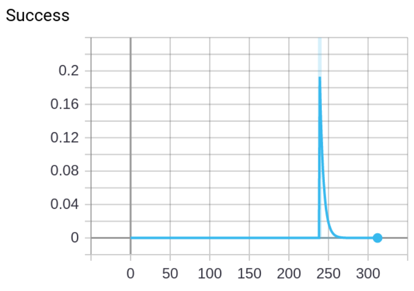


(a) Reward per episode (b) Binary success per episode

Figure 4.11: TVS task performed by BC using 10 epochs, on the real-world robot.



(a) Reward per episode



(b) Binary success per episode

Figure 4.12: TVS task performed by PPO with a policy initialized using BC with 10 epochs, in simulation.

was operating above the valve, moving the gripper nearly directly upwards until the end of the episode.

Chapter 5

Discussions

This chapter will interpret and discuss the results from Chapter 4 and the process of obtaining them described in Chapter 3.

5.1 Performance

Figure 4.2 shows that PPO can be used to solve the task of reaching the valve in some capacity. After a significant time spent training the algorithm reaches periods of 100% success, and an average of 90% across the last 500 episodes. The total training time was 13 hours, in a simulation with a real-time factor of 2.5. This can be viewed as the equivalent of 32 hours of training, if the training was done using a real-world RM. It should be noted that slightly higher real-time factors are achievable on different hardware, and greater improvements are available with newer versions of Gazebo. However, the simulation solution for the "OpenManipulator-X (RM-X52-TNM)" is only available for Gazebo 7.

PPO have been used to perform tasks of similar and greater complexity already in the very paper that proposed it (Schulman et al., 2017). The ability to solve the RV task in simulation is not a considerable achievement. The goal of this thesis is rather to see how this policy obtained from fully simulated training performs on a real-world system. And, as the result in Figure 4.3 show, the policy maintained a similar performance after the transfer. This is promising for the use of simulation to train an agent to later control a manipulator for a more complex task where real-life training is unfeasible due to reasons such as time needed, potential damage to people or equipment, or a difficult to recreate environment.

While the RV task was solved satisfactorily with the use of PPO, TVS had worse performance. In Figure 4.9b, the average success rate rarely exceeds 70%. The task itself does not require any complex trajectories and can be completed quite directly. Thus, the exact reasons the agent was not able to find a better policy are unclear. Possible reasons are a poor reward function and an unsuited configuration of the ANN. However, when the policy was employed on the real-world manipulator, it archived a 100% success rate for 50 episodes, a significant difference from the performance in the simulation. This shows that the policy performed in the simulation is at least logical and capable of succeeding, but not certain to succeed in the simulated environment. The real-world manipulator had more slack between each part, which caused the gripper to be in a slightly different position than in simulation, with the same join positions. The this deviance is only present in the vertical direction, it did not greatly affect the RV task, but it is likely that the slightly different point of contact between the gripper and the valve, combined with the different interaction between the two in the real-world is the cause of the greatly improved performance.

In stark contrast to the transition from simulation to real-world shown in Figure 4.9 and 4.10, the performance of the BC trained policy performed moderately well on the real-world manipulator, but succeeded only once across over 300 episodes in simulation. Employing BC on the RV task had far better results, with a 100 success rate for over 100 episodes. From this it can be understood that BC and RL in the form of PPO can tackle different problems. Figure 4.5 showing the performance of a policy initialized using a very quick BC training session, tells us that a combination of BC and PPO can potentially be used to solve tasks where PPO alone takes unfeasible long to train, even in simulation. The success rate starts around 65% and improves to 90% over 500 episodes, where as PPO alone, seen in Graph 4.2, requires around 1 000 episodes to reach the same level of performance. The number of validation episodes on the real-world manipulator was constricted by time available and as such can not validate the success rate with any great accuracy, but they do show that a policy with a high success rate in simulation also has a high success rate when applied to the real-world manipulator.

The design of the tasks along with the starting position and use of only two joints allows the agent to quickly stumble upon a reward as shown in Graph 4.1a. The greatest benefits of BC as it is used in this thesis is that it gives the RL agent a policy that allows it to start its exploration around a state-space where the rewards exists. This then suggest that while BC was not needed for the RV or TVS tasks, and PPO was able to learn a suitable policy from scratch, a more complex task with a larger state- and action-space where the rewards are spare and as such less likely to be stumbled upon is likely to benefit more from an pretrained initial policy.

A key part of RL problems is the design of the reward function. In this thesis only a single version have been used for each of the task, and there is room for the possibility that a different reward function would improve training time and results. The reward function used is designed to give the agent a positive reward when it improves its state proportional to either how good that state is (RV) or how much the state is improved

(TVS). Other designs will give rewards proportional to the value of the state and let the RL agent infer which actions will lead to a higher reward. The thought behind the reward function used is that the agent skips having to learn approximately the true value of a state before it can decide if another state is better, and instead simple learns that when in the state, moving to the next state gives a reward. This avoids the situation where the choice to move away from the goal gives a slightly lower reward, and it instead gives no reward, heavily encouraging the "correct" move. This depends on the reward function being design such that this is indeed the desired behavior, and does make some additional assumptions on the desired behavior. In contrast, a sparse reward function, such as only getting a reward when completing the task, would theoretically teach the algorithm to make the optimal decisions. Actually making it to that end-point would require significant amounts of training and may in some cases be infeasible. It may however make an appropriate task for combining BC and RL.

5.2 Valve identification

As mentioned in Subsection 3.4.2, the coordinates of the goal position used in RV were measured manually. In subsection 3.1.3 the use and mounting of a Raspberry Pi Camera was introduced, alongside software capable of tracking a marker and giving the position relative to the cameras position on the robot. Knowing the location of the marker on the valve and the mounting of the camera on the manipulator, this can be combined to create a vector to the goal position from the gripper as required by the RV task.

$$V_{goal} = V_{gripper_to_cam} + V_{cam_to_marker} + V_{marker_to_gripable} \tag{5.1}$$

This can also be done without the use of a marker, by directly identifying the valve from the image. There are multiple methods for visual object identification (Viola et al., 2001), including methods intended for use underwater (Foresti and Gentili, 2000; Lee et al., 2012).

With the camera mounted on the manipulator as in Figure 3.4 the valve leaves the cameras field of view as the gripper nears the goal location. The goal may be remembered from the last frame where the valve could be identified, but this will leave the manipulator weak to disturbances which transposes or rotates the base of the manipulator or valve. An alternate approach where the camera is positioned somewhere separated from the manipulator will alleviate these issues, but will require calculating the kinematics of the manipulator to find the relative vector from the gripper to the goal position.

5.3 Software and Simulation

The process detailed in Section 3.2 and 3.3 to get the simulation running, and the RL agent able to control both the simulated and the real-world manipulators was long and time consuming. The errors in the provided software for the manipulator simulation, while minor, are time consuming to debug, and requires the operator to learn the underlying system. Such barriers in the path of performing the actual desired research hinders advancement in control of physical robotic manipulators, particularly as researchers with a background of artificial intelligence may not have any experience with the control of robotic systems, or with ROS in general. The Open AI project aims to aid the start-up process for AI research and enabling researchers to get started with far less time spent setting everything up. Those solutions remains grounded in simulation and this author is aware of no similar project for any simulation that is paired with an actual, affordable yet capable, robotic manipulator.

Chapter 6

Conclusions

This thesis has looked at the past of robotic manipulators, and the basis of RL with the distant goal of an RM controlled by AI performing subsea operations. A market survey was conducted to choose a suitable platform, and the physical and simulated valve model were created. A simplified task of reaching and turning a valve was created, split into navigating to the valve and, once there, turning it. The task was solved using PPO using a simulation and the resulting policy applied to a real-world RM. The same tasks were then solved using BC, and the results of both approaches discussed.

The OpenManipulator platform, employing ROS, Gazebo and MoveIt for simulation shows several critical limitations in relation to use for RL. Chief among them is the lack of a simple reset for episodic learning, the notable control differences from the real robot, and the low speed up factor. Solutions to these problems are needed for the full power of simulated training to be harnessed. The additional problem encountered in the thesis of the interaction between the valve and the robot prevented training with a grasped valve, but may be remedied by a different valve model.

PPO have proved itself as an algorithm able to tackle the high dimentionality of RMs with the hard to define tasks it needs to perform, and was therefore chosen for this thesis although the specific tasks being performed may be better solved with other methods. The results show that PPO is capable of solving the task, but at a significant time cost in training. However, when simple BC is employed to give PPO an initial policy, it can reach the same level of performance in half of the time. The RV task being solved in the thesis could be solved with just the use of BC, but for more complex tasks where the expert demonstrations are not complete, the union of BC and RL may lead to better results in shorter time than either could achieve on their own.

The use of a camera to identify the location of the valve proved exceedingly feasible for simple models using a known marker, and shows that a full visual module locating a valve through other means can be employed with the existing RL agent without any changes to the latter.

Future work may be done refining the simulated environment to better match realty, with particular focus on the interaction between the manipulator and other objects. With the OM platform and the setup described in this thesis, future work using the same manipulator may focus on longer, sequential tasks where discrepancies between model and reality will have greater effect. The combination of BC and RL may be further explored in tasks where the reward function is sparse, either by choice or necessity, and the RL is method unable to make any headway in solving the problem. A comparison of an RL approach and a classical control approach to the same problems may further aid in illustrating where RL can be employed for the greatest benefit.

Appendices

Appendix A

Reward functions

A.1 Reward function for RV task

```
reward = 0
if distanceToValveImproved:
  reward += 1 - self.dist*10
else:
  reward -= 1 + self.dist*10
If reachedGoal:
  reward += 10
elif outsideWorkarea:
  reward -= 50
```

A.2 Reward function for TV and TVS task

```
reward = 0
if valvePos > bestValvePos:
  reward += 1 + (valvePos - bestValvePos)*10
  bestValvePos = valvePos

if valvePos >= goal:
  reward += 10

if outSideWorkarea:
```

Bibliography

Ajwad, S., Ullah, M., Khelifa, B., and Iqbal, J. (2014). A comprehensive state-of-the-art on control of industrial articulated robots. *Journal of Balkan Tribological Association*, 20(4):499–521.

Allgöwer, F., Badgwell, T. A., Qin, J. S., Rawlings, J. B., and Wright, S. J. (1999). Nonlinear predictive control and moving horizon estimation—an introductory overview. In *Advances in control*, pages 391–449. Springer.

Amazon. Amazon web services. `https://aws.amazon.com/`. Accessed: 2019-04-18.

Arimoto, S. (1994). State-of-the-art and future research directions of robot control. *IFAC Proceedings Volumes*, 27(14):3–14.

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. (2018). Sample-efficient reinforcement learning with stochastic ensemble value expansion. *CoRR*, abs/1807.01675.

Chow, Y., Nachum, O., Duenez-Guzman, E., and Ghavamzadeh, M. (2018). A lyapunov-based approach to safe reinforcement learning. *arXiv preprint arXiv:1805.07708*.

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472.

Dubowsky, S. and DesForges, D. (1979). The application of model-referenced adaptive control to robotic manipulators. *Journal of dynamic systems, measurement, and control*, 101(3):193–200.

Feinberg, V., Wan, A., Stoica, I., Jordan, M. I., Gonzalez, J. E., and Levine, S. (2018). Model-based value estimation for efficient model-free reinforcement learning. *CoRR*, abs/1803.00101.

Foresti, G. L. and Gentili, S. (2000). A vision based system for object detection in underwater images. *International journal of pattern recognition and artificial intelligence*, 14(02):167–188.

Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. (2016). Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838.

Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. (2018). Model-ensemble trust-region policy optimization. In *International Conference on Learning Representations*.

Landau, I. (1974). A survey of model reference adaptive techniques-theory and applications. *Automatica*, 10(4):353–379.

Lee, D., Kim, G., Kim, D., Myung, H., and Choi, H.-T. (2012). Vision-based object detection and tracking for autonomous navigation of underwater robots. *Ocean Engineering*, 48:59–68.

Levine, S. and Koltun, V. (2013). Guided policy search. In *International Conference on Machine Learning*, pages 1–9.

Lewis, F. L. (2004). Robot manipulator control : theory and practice.

Li, Y. (2017). Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Luh, J. Y., Walker, M. W., and Paul, R. P. (1980). On-line computational scheme for mechanical manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 102(2):69–76.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Niryo. Niryo one. `https://niryo.com/niryo-one//`. Accessed: 2019-06-21.

nptel. course on mechatronics and manufactoring automation. `https://nptel.ac.in/courses/112103174/`. Accessed: 2018-08-30.

Open AI. Gym. `https://gym.openai.com/`. Accessed: 2019-06-17.

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

Paul, R. (1972). Modelling, trajectory calculation and servoing of a computer controlled arm. Technical report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE.

Paul, R. P. (1981). *Robot manipulators: mathematics, programming, and control: the computer control of robot manipulators*. Richard Paul.

Poignet, P. and Gautier, M. (2000). Nonlinear model predictive control of a robot manipulator. In *Advanced Motion Control, 2000. Proceedings. 6th International Workshop on*, pages 401–406. IEEE.

ROBOTIS. Openmanipulator-x (rm-x52-tnm). `http://www.robotis.us/openmanipulator-x-rm-x52-tnm/`. Accessed: 2019-06-10.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Sayed, A. rl-examples. `https://github.com/Anjum48/rl-examples/tree/master/ppo`. Accessed: 2019-04-02.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015). Trust region policy optimization. *arXiv preprint arXiv:1502.05477*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sciavicco, L. and Siciliano, B. (2012). *Modelling and control of robot manipulators*. Springer Science & Business Media.

Shestopal, E. Carnd-behavioral-cloning-p3. `https://github.com/erikshestopal/CarND-Behavioral-Cloning-P3`. Accessed: 2019-06-07.

Siciliano, B. (2009). Robotics : Modelling, planning and control.

Spong, M. W. (1989). Robot dynamics and control.

Sutton, R. S. and Barto, A. G. (2018). Reinforcement learning: An introduction (bibinfoedition2 ed.).

Sutton, R. S., Barto, A. G., and Williams, R. J. (1992). Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems*, 12(2):19–22.

Takegaki, M. and Arimoto, S. (1981). A new feedback method for dynamic control of manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 103(2):119–125.

Trossen Robotics. Widowx robot arm mark ii kit. https://www.trossenrobotics.com/WidowxRobotArmMK2/. Accessed: 2019-06-21.

Ubiquity Robotics. Raspberry pi images. https://downloads.ubiquityrobotics.com/pi.html. Accessed: 2019-05-07.

Uchiyama, M. (1989). Control of robot arms. *JSME international journal. Ser. 3, Vibration, control engineering, engineering for industry*, 32(1):1–9.

Vermedal, A. (2018). Reinforcement learning for control of robotic manipulators. Unpublished.

Viola, P., Jones, M., et al. (2001). Robust real-time object detection. *International journal of computer vision*, 4(34-47):4.

Whitney, D. E. (1987). Historical perspective and state of the art in robot force control. *The International Journal of Robotics Research*, 6(1):3–14.

Wichman, W. M. (1967). Use of optical feedback in the computer control of an arm. Technical report, STANFORD UNIV CALIF DEPT OF COMPUTER SCIENCE.

Wiering, M. and Otterlo, M. (2012). Reinforcement learning : State-of-the-art.

Wikipedia. A set of six-axis robots used for welding. https://en.wikipedia.org/wiki/File:FANUC_6-axis_welding_robots.jpg. Accessed: 2018-09-07.

Yahya, A., Li, A., Kalakrishnan, M., Chebotar, Y., and Levine, S. Collective robot reinforcement learning. https://www.youtube.com/watch?time_continue=7&v=QZvu8M02BeE. Accessed: 2018-09-07.

Yoshikawa, T. (2000). Force control of robot manipulators. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 220–226. IEEE.

Zurada, J. M. (1992). *Introduction to artificial neural systems*, volume 8. West publishing company St. Paul.