Renée van der Wijden

Preference-driven demonstration ranking for inverse reinforcement learning

TU Delft

Preference-driven demonstration ranking for inverse reinforcement learning

By

Renée van der Wijden

in partial fulfilment of the requirements for the degree of

Master of Science
in Mechanical Engineering

at the Delft University of Technology,
to be defended publicly on Monday July 4, 2016 at 10:00 AM.

`

Head of committee:	Prof. Dr. Ir. M. Wisse	TU Delft
Supervisor:	Dr. Ir. J. Kober	TU Delft
Supervisor:	Dr. Msc. C. Grappiolo	Alten Nederland
Thesis committee:	Ir. I. Koryakovskiy	TU Delft

Acknowledgements

First of all I want to thank Alten, with providing me with the graduation assignment and the freedom in the exact formulation of the assignment. Especially I would like to thank my supervisor from Alten, Corrado, who helped me to stay positive through his endless humor.

Furthermore, I would like to thank Jens, for his experienced view on reinforcement learning and inverse reinforcement learning. Especially I would like to thank him for his patience in explaining me the terminology.

I thank Smart Robotics for providing me access to an UR5 for performing my experiments. Especially, since it this has been my first experience with (industrial) robotic arms and they helped me a lot with getting around with the robot. Also a special thanks to all the participants of the crowdsourcing experiment.

Finally, I would like to say thank you to my family and friends and to Guido for having the confidence in me, even when I did not.

Renée van der Wijden
Delft, July 2016

Contents

Abstract

New flexible teaching methods for robotics are needed to automate repetitive tasks that are currently still done by humans. For limited batch sizes, it is too expensive to teach a robot a new task (Smith & Anderson, 2014). Ideally, such flexible robots can be taught a new task by a non-expert. A non-expert is a person who knows the task the robot should perform, but does not have experience in programming a robot. A powerful method that would allow for flexible robotics without the use of an expert is inverse reinforcement learning (IRL). IRL aims to learn the cost function out of demonstrations, this cost function is subsequently used to learn a policy which realizes the desired task.

Current implementations focus more on the IRL algorithm itself and assume that there are enough demonstrations available and the quality of these demonstrations is also close enough to the optimal behaviour (Doerr et al., 2015). Whilst actually these demonstrations are very expensive and non-optimal. This thesis focuses on the effect of the quality of input demonstrations on the performance of the learned trajectory. Furthermore, how imperfect demonstrations still can be used, without lowering the performance of the learned trajectory. The first hypothesis is that the performance of the resulting trajectory depends on the average performance of the input demonstrations and the quantity of the input demonstrations has less of an effect. The second hypothesis is that by adding a ranking to the demonstrations, created through the preferences of non-robotic experts, the performance of the learned trajectory would be better than the average performance of the input demonstrations. The preferences of the non-robotic expert are collected through a crowdsourcing experiment. The preferences of the non-robotic expert are used to create an overall performance measurement. This overall performance measurement is used to obtain the sequentially order of the input demonstrations but also to evaluate the final learned trajectories.

The results validate the first hypothesis. The average performance of the input demonstrations is determining the performance of the learned trajectory. The second hypothesis could not be confirmed. The results did not show any improvements in the performance of the learned trajectory when the ranking based on the preference of a non-robotic expert is added. It could be argued that the input demonstrations were too similar or the cost features used in IRL are not specific enough to create different cost functions and therefore create differently performing trajectories.

List of acronyms

IL – Imitation Learning
RL – Reinforcement Learning
IRL – Inverse Reinforcement Learning
PI^2 – Policy Improvement with Path Integrals
MDP – Markov Decision Process
DMP – Dynamic Movement Primitives

List of definitions

Reward function (R) - this is the core of RL from where the robot can learn the optimal policy.

Cost function (C) - this is the negative reward function, as is also shown in Equation 3-6.

Policy (π) - this is a function which determines which action to take depending on which state the robot is in with respect to its environment. This policy determines the trajectory the robot is following. The optimal policy π^* is the one that gives the optimal trajectory.

Trajectory ($\boldsymbol{x}$) - are the Cartesian coordinates of the end effector of the robot while performing the task. Mostly it is seen as a vector with a corresponding time vector.

Non-robotic expert – A person who does not know how to program a robot, but does know the movement the robot should perform.

Robotic expert – A person who has experience in programming a robot.

1. Introduction

In the past decades, robots have made their entrance into the industrial production halls. They can perform tasks precisely and quickly. Therefore for most of the simple repetitive production tasks the robots are outperforming humans. Currently, robots are programmed to perform a fixed sequence of actions which performs the task. When there is a small change in the environment or in the task itself, a change is needed in this fixed sequence of actions and the robot needs to be reprogrammed. This (re)programming needs to be done by a person who knows how to program a robot, also called a robotic expert, and takes a lot of time. Therefore, the use of robots has large start-up/ investment costs. For large batch sizes, the usage of robots is profitable, but for small batch sizes, these start-up costs are too large.

Since the start-up costs of the usage of robots are quite large, a lot of repetitive tasks in the industrial production halls are still done by hand (Smith & Anderson, 2014). For example, in the production of shaving machines the shaving machines are often changing a little bit in shape/ size and to reprogram a robot each time is more expensive than letting a human perform the tasks by hand. Therefore, robots are not used for these relatively 'small' batch sizes.

To reduce the start-up costs it would help if a non-robotic expert is capable of teachting the robot a new task. A non-robotic expert is a person who is not a robotic expert, but knows the task the robot should perform. Moreover, when a robot can learn directly to perform a certain task from a non-expert, the programming of the robot becomes more flexible. Flexible since no robotic expert is needed and anyone can teach the robot a task.

Currently, there are three ways to teach a robot a new task (Kormushev et al., 2013): Direct programming, imitation learning, and reinforcement learning. In the next paragraphs, a brief description is given of those methods.

Direct programming is the current way of teaching robots a task. With this method, the robot performs a fixed sequence of actions, which is given by a robotic programming expert.

With imitation learning (IL) the robot mimics the demonstrator. The robot is typically provided with a demonstration. In most cases the demonstration is given through kinaesthetic teaching. By means of this way of teaching the robot is pulled along the desired trajectory by the human as shown in Figure 1-1. This type of demonstration can be easily given by a non-expert.

Figure 1-1 - Kinaesthetic teaching example (Englert et al., 2013).

This demonstration is used to create a statistical method of the policy. A policy is a function that determines which action is taken depending on which state the robot is in. The state of the robot is the position and velocity of the robot with respect to its environment.

A disadvantage of IL is that when the environment changes, the robot will not perform optimally anymore. Another disadvantage is that IL assumes that the given demonstration is completely optimal since in fact the demonstration can be a suboptimal solution to the task that is needed to be performed. Both disadvantages are caused by the definition of IL. With IL a policy is created to follow the shown trajectory, while actually it is not desired to imitate the specific trajectory that is shown, but the task that is performed (Ng & Russell, 2000). The

task can be interpreted as reaching a goal state as fast as possible or on a more high level to grasp an object. The goal state is the position which the robot wants to move towards.

Reinforcement Learning (RL) is a trial and error approach where the robot needs to practise and make mistakes to find an optimal trajectory. RL tries to maximize the incoming cumulative reward which is given through the reward function. This reward function determines how much reward should be given after an action is taken depending on which state the robot has come into. This reward function typically encodes the desired behaviour, e.g., a high reward is typically given when the robot reached its goal state. To find out which action needs to be taken, the robot needs to explore its state action space. So for each state the robot could be in, it needs to try all actions. By doing this it can create a mapping of its environment and knows which action leads to the highest reward. Therefore, the trade-off of exploiting the known information and exploring for new information is a key element in RL.

A disadvantage of RL is a lot of trials are needed before it completely explored the state action space and can exploit this information to determine the optimal policy. Typically this exploration is done in real time and robots suffer from wear and tear (Kober & Peters, 2012). Another disadvantage is that creating a reward function is not a trivial task (Zhifei & Joo, 2012) and needs to be programmed by a robotic expert. This while the goal of the flexible robotics would be to eliminate this robotic expert and only use demonstrations to come to a trajectory. Figure 1-2 shows a flowchart of RL with respect to the ideal method.

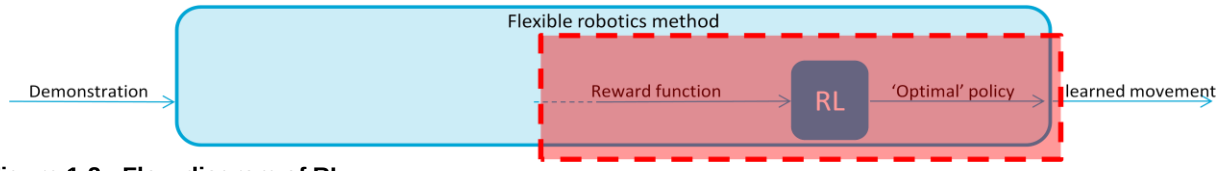


Figure 1-2 - Flow diagram of RL

In Table 1-1 a summary is given of the IL and RL method. Both are good flexible methods, but are still not optimal. Desired is to use a combination of both methods where it uses the demonstrations of a non-robotic expert as an input, while being able to optimise to the wanted task.

Table 1-1 – Summary of the advantages and disadvantages of IL and RL

	Advantage	Disadvantage
IL	A non-expert can be used to provide the demonstrations	Demonstrated trajectory is reproduced - Sensitive to changes in the environment - Given demonstration can be suboptimal to the task that needs to be performed
RL	The wanted behaviour/ task is performed	Lot of trials are needed An expert is needed to design reward function

Inverse reinforcement learning (IRL) is a promising method which overcomes the main disadvantages of IL and RL (Zhifei & Joo, 2012). With IRL the reward function is extracted from a given demonstration or from multiple demonstrations (Ng & Russell, 2000), as is shown in Figure 1-3. This is as the name suggest the inverse of RL where the reward function is used to learn a policy. IRL uses likewise, as is done with IL, the demonstrations of a non-robotic expert as input. With IRL those demonstrations are not directly imitated but are used to find the underlying behaviour in the form of the reward function. This gathered reward function can then be used in a RL algorithm to obtain the optimal policy, as is shown in Figure 1-3. Through this approach, there is no need of a robotic expert to design the appropriate reward function.

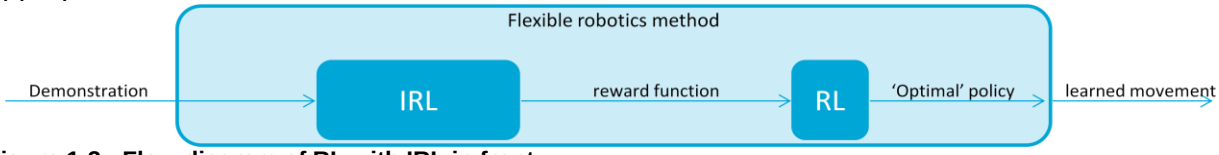


Figure 1-3 - Flow diagram of RL with IRL in front

2. Research question

This thesis is built around the idea that flexible programming a robot benefit from making a perfect trajectory from a set of imperfect demonstrations. Those demonstrations are used as an input for inverse reinforcement learning (IRL) as is shown in Figure 2-1. Decided is to focus this thesis on using the imperfect demonstrations optimally.

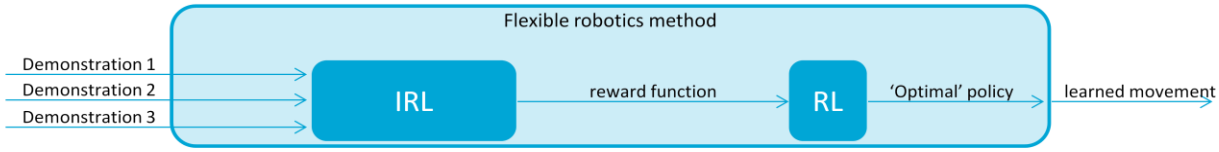


Figure 2-1 - IRL creates a reward function from multiple demonstrations. Afterwards an optimal policy can be found with reinforcement learning (RL). A policy is a function which determines which action to take depending on which state the robot is in, as mentioned in Section 1.

Obtaining good demonstrations is very expensive since demonstrations are often considered suboptimal or sometimes the demonstration did not even succeed in fulfilling the task at hand. There are several reasons for the low quality of the demonstrations. Two examples are noise in the recordings or the human demonstrator is not physically capable of demonstrating the intended behaviour. Even though the demonstrations are suboptimal they do contain valuable information about the desired task.

With traditional IRL the provided demonstration is assumed to be the optimal trajectory. When multiple demonstrations are used they are seen as equally important. Therefore, the quality of all the input demonstrations is important.

If the demonstrations are used optimally and therefore fewer demonstrations are needed to teach the robot the desired task, the solution would become more useful for real life purposes. This helps with reaching the goal of flexible programming a robot for industrial production tasks.

Current implementations focus more on the IRL algorithm itself and assume that there are enough demonstrations available and the quality of these demonstrations is also close enough to the optimal behaviour (Doerr et al., 2015).

The main research question of this thesis is, therefore:

"How can the (suboptimal) demonstration(s) optimally be used for IRL?"

This question leads to the following two hypotheses:

Hypothesis 1: One versus multiple input demonstrations

The performance of the learned trajectory depends on the average performance of the input demonstrations, the amount of input demonstrations is less important.

This first hypothesis looks at the effect of the amount of demonstrations used as input. It is expected that adding more good demonstrations to an already selected set of demonstrations gives a better performance. Whilst when adding a bad performing demonstration the performance decreases.

Hypothesis 2: Without versus with added preferences

When ranking the demonstrations through weight factors of the preferences of a non-robotic expert the performance of the learned trajectory is better than without the added weight factors.

This second hypothesis explores the option of adding preferences of a non-robotic expert to improve the performance. The preferences are added as a weight factor for the demonstration before using it as an input for IRL. It is expected, that now when adding a bad performing demonstration it does not mean that the performance is decreased, which is opposite to what is expected at Hypothesis 1.

To better understand the hypotheses the two key definitions are explained. Section 2.1 explains what is meant with the definition performance, e.g. when is a demonstration performing better than another one. Section 2.2 explains how the demonstrations are ranked through the preferences of a non-robotic expert.

2.1 Performance of the robot

To validate the hypotheses it is important to describe what performance is, and what good or better performance is. Intuitively the concept of performance has several meanings. The dictionary gives the following description:

performance || noun (ACTIVITY) [C or U] how well a person, machine, etc. does a piece of work or an activity: (Cambridge-Dictionaries, 2016)

In other words, when the robot is doing the desired task, it is performing well. Therefore, how to properly measure performance depends highly on the task. The performance measurements can be divided into two categories: The primary measurements, and the secondary measurements, as is shown in Table 2-1. The primary measurements are the measurements which determine if the task was successful or not, the secondary measurements determine the quality of the performed task.

The first primary performance measurement determines if the robot successfully performed the requested task. Another primary performance measurement is to avoid a collision. When the robot has a collision with itself or its environment this can give damage to the robot itself or its environment and the task probably needs to be stopped.

The quality of performance, the secondary performance measurements, is described by other factors such as the execution time and the smoothness of the movement. The execution time is the actual time that the robot is moving to perform the task. The execution time should be as small as possible, which means that the task was performed quickly. Smoothness is important, because if the movements are very jerky, the robot has a lot of wear and tear, therefore, it is not beneficial for the robot itself.

Table 2-1 - The performance measurements can be divided into two categories: The primary measurements and the secondary measurements.

Primary measurements of performance	Secondary measurements of performance:
• Successful execution of the industrial task • No collision	• Smallest execution time • Length of the path taken • Smooth movement

More details about the performance measurements used for this research can be found in Section 5.2.

2.2 Ranking the input demonstrations

The ranking of the demonstrations is ideally done by a non-robotic expert since the goal of the flexible robotic approach is that anyone can teach the robot a production task. Therefore, ranking the demonstrations is preferred over alternatives such as ranking measured results for which still expert knowledge is required.

For adding the preferences to the demonstration, weight factors are the simplest approach to use for most IRL algorithms. Other approaches would be to include it in the IRL algorithm, which would involve changing the IRL algorithm. Therefore, the weight factors

approach was used as shown in Figure 2-2 because it can give a quick analysis if adding this preference information is beneficial.

How to obtain the order of the demonstration through the preferences of a non-robotic expert is studied in Section 3.4, where the fundaments of preference learning (PL) are explained.

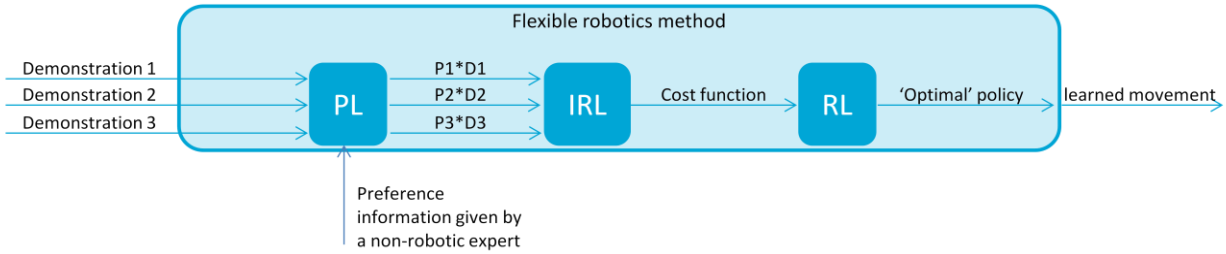


Figure 2-2 - Flow diagram of IRL and RL with added weight factors through the preferences of a non-robotic expert

However, it is important that the performance measurements of Section 2.1 correspond with the preferences of a non-robotic expert. If the performance measurements do not correspond to the non-robotic expert perceived performance unexpected results can occur. In such a case the robot is optimised to the perceived performance, while the learned trajectory is evaluated through different measurements. In other words, the robot can get really good results according to the perceived performance, but is scoring badly on the defined performance measurements.

How this correlation is checked can be found in Section 4.1.2. Furthermore, this section explains how those performance measurements and the preferences are used to create the weight factors.

2.3 Outline of report

First is the background information of the relevant algorithms explained in Section 3. Afterwards the method, which is used to validate the hypotheses is described in Section 4. Subsequently, the industrial production task used for this research is explained with the corresponding performance measurements in Section 5. Section 6 describes the experimental set-up. Section 7 shows the results and finally Section 8 and 8.5 show the discussion and conclusion respectively.

3. Background information

The main goal of this thesis is to teach a robot a task from a non-robotic expert, as is shown Figure 3-1. Focussing on this main goal, the most interesting algorithms are explained.

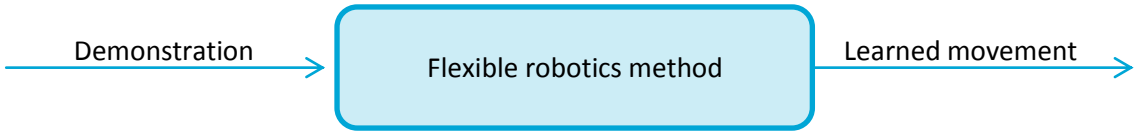


Figure 3-1 - Flexible robotics method which uses a demonstration given by a non-robotic expert to obtain which action the robot should take depend on the state the robot is in, also presented through a policy which realizes a movement.

Figure 3-2 shows the outline of this section. First an explanation about the algorithm imitation learning (IL) is given in Section 3.1. IL directly uses a demonstration to obtain a policy. The section also describes the main advantages and disadvantages.

Section 3.2 describes the fundaments of reinforcement learning (RL) and also it describes which RL algorithm would be suitable for this research. With RL a reward function is used, which design is not a trivial task.

Inverse reinforcement learning (IRL) is explained in Section 3.3. With IRL a cost function can be learned from a demonstration or multiple demonstrations. This method is added to the RL algorithm to be able to learn from a demonstration the movement, as earlier was shown in Figure 1-3.

Afterwards the basic principles of preference learning (PL) are described in Section 3.4. PL is used to rank the demonstrations before using the demonstrations for the IRL and RL algorithm.

Finally, Section 3.5 explains the dynamic movement primitives (DMP). This is an algorithm to parameterize the policy, which is needed for the reinforcement learning algorithm and the inverse reinforcement learning algorithm.

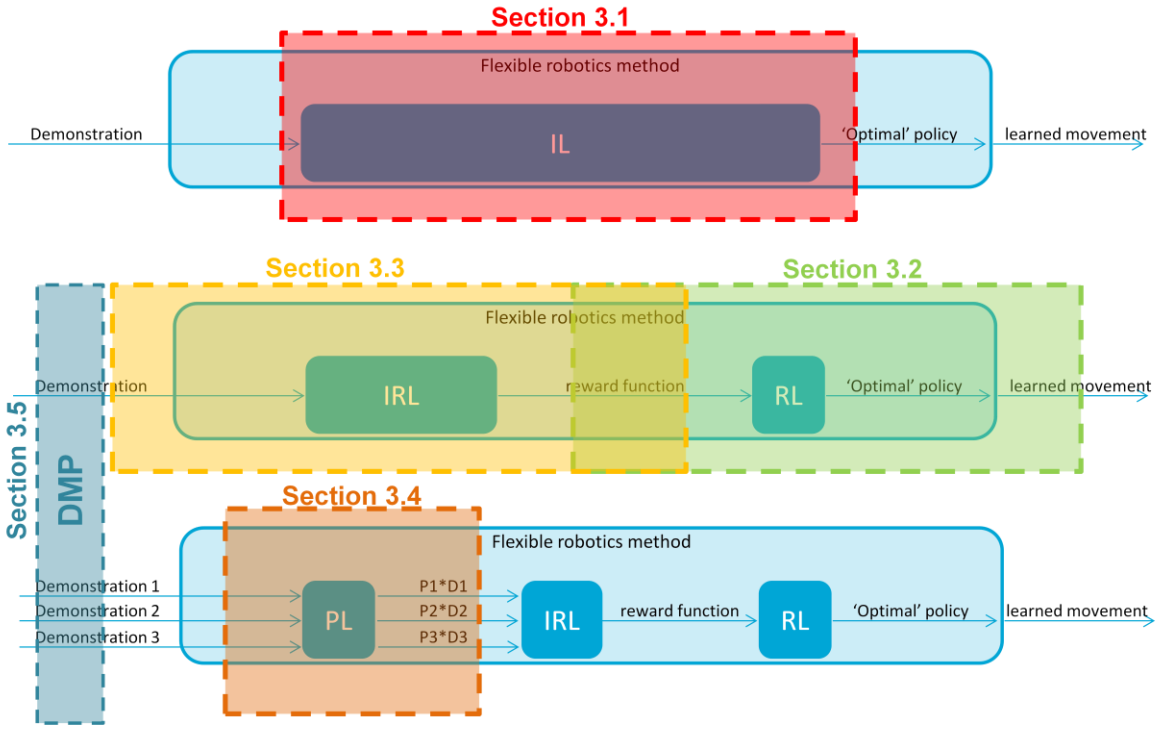


Figure 3-2 – Graphical representation of the outline of this section.

3.1 Imitation learning

With imitation learning (IL) (programming by demonstration or learning from demonstration) a robot learns from one or multiple demonstrations (Argall et al., 2009; Bautista-Ballester et al., 2014). From the demonstrations IL directly imitates the given movement, as is shown in Figure 3-3. With this method a demonstration is assumed to be the optimal performance of the task.

When teaching a robot a policy through IL, two questions need to be answered (Argall et al., 2009), which are discussed in Section 3.1.1. The first question is “how to collect the useful information from given demonstrations?”. The second question is “how to derive an appropriate policy from the obtained information?”. In Section 3.1.3 the advantages and disadvantages of IL are stated.

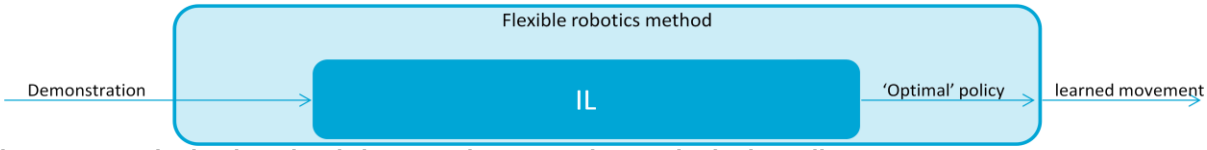


Figure 3-3 – Imitation learning (IL) uses a demonstration to obtain the policy.

3.1.1 Gathering data from demonstrations

The most direct approach to demonstrate a task to a robot is by kinaesthetic teaching. Here the human operates the robot by moving the passive joints in such a way that the desired movement is shown to the robot, as depicted in Figure 1-1. For this kinaesthetic teaching the robot needs to be in the teach mode. The teach mode is the mode where the robot can be moved around freely. The advantage of this method is that the robot can directly record its own joint states and no mapping is needed, a mapping is a function which transforms a set of joint states to a different format. A disadvantage is that kinaesthetic teaching can be quite challenging and physically hard for a human to perform. This is highly depending on the size and type of the robot and the options of the teach mode available.

Another less direct way of demonstrating is to place sensors on the human demonstrator and let her perform the task. Afterwards the data of the sensors need to be transformed into the joint states of the robot, and a mapping function is needed. This approach is therefore considered to be less direct.

3.1.2 The basic principles of IL

The data from the demonstration is used to derive a policy. How to derive the policy can be categorized dependeding on the learning objective, as is shown in Figure 3-4.

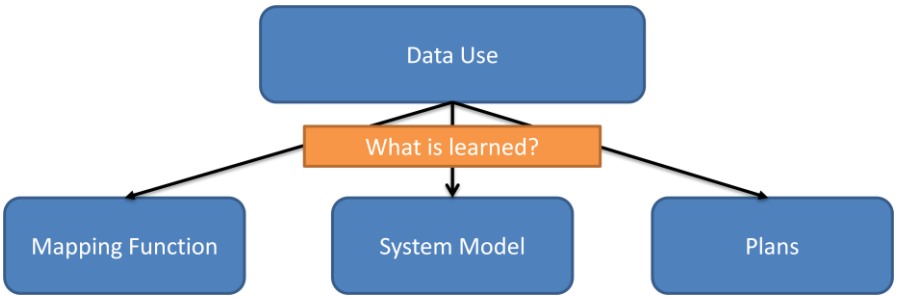


Figure 3-4 – The three categories of IL, which are divided through their learning objective. (Argall et al., 2009)

- Mapping Function, this function consists of directly copying the movement of the demonstration or of multiple demonstrations into the policy (Argall et al., 2009). Since the demonstrations most likely will contain some noise, statistical methods are generally used.

- System model, here the information of demonstration or of multiple demonstrations is used to create a state transition probability function($P(s', s, a)$), also called the system model (Argall et al., 2009). This function contains the probability of reaching the state (s') after a certain action (a) was chosen depending on the state (s) the robot was in previously. This state transition probability function $(P(s', s, a))$ is then used as a starting point for RL to maximize the incoming reward. During RL $(P(s', s, a))$ is updated with the new information that is gathered (Sutton & Barto, 1998). More explanation of RL can be found in Section 3.2.

- Plans, this approach creates a planning framework from the demonstrations (Argall et al., 2009). The planning framework is intended for a high level control task, e.g. it tries to plan, whether a robot should grasp an object or not, rather than the low level control task of how to grasp the object. The industrial production task that is considered during this research is at a low control level, therefore it is chosen to mainly focus on the first two groups.

3.1.3 Advantages and disadvantages of IL

The main advantage of using demonstrations to teach a robot a task is that it is simple to do. So anyone who knows the task the robot should perform is able to demonstrate it.

The biggest disadvantage of IL is due to its definition. With IL is assumed that the demonstration is optimal and therefore this method generalises to the demonstrations. The performance of movement learned through IL is therefore limited to the movement that is shown through the demonstrations. With IL no new strategies can be learned.

3.2 Reinforcement learning

With reinforcement learning (RL) the robot receives a reward from its reward function, depending on which state it is in after a certain action is taken (Sutton & Barto, 1998). This reward is given when the robot performed a desired action. The robot tries to maximise this incoming reward, by updating its policy. The policy that gets the highest reward is the optimal policy.

The continuous learning process is shown in Figure 3-5. The robot being in a certain state $s(t)$ at time step t performs an action $a(t)$. This action $a(t)$ leads to, depending on the environment, the new state $s(t+1) = s'$ of the robot. The reward function $R(a(t), s(t+1))$ determines the corresponding reward $r(t+1)$ of this action based on this new state. The reward function is seen as part of the environment in Figure 3-5.

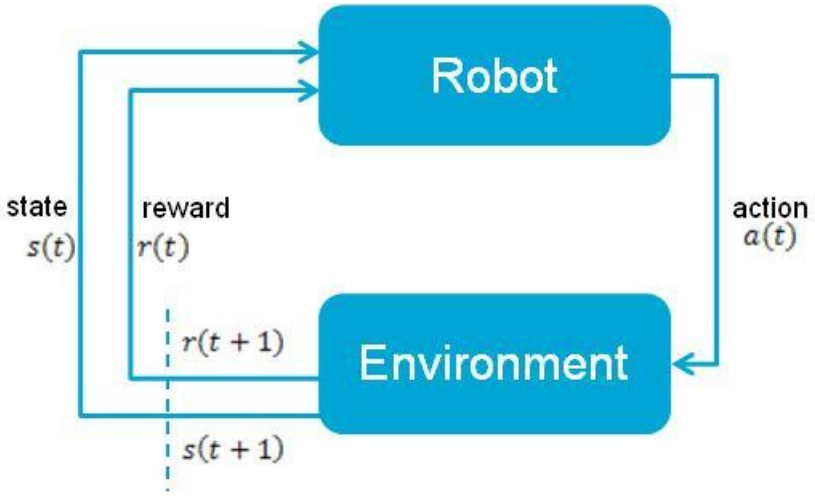


Figure 3-5 – Flow diagram of a RL agent (Sutton & Barto, 1998)

With RL it is assumed that the robot has no information about the environment and therefore it will need to explore it. After each action the robot saves the new knowledge about the environment by updating its state transition probability function $P(s', s, a)$, depending on the new state and reward. The higher the accumulated reward over the trajectory the robot gets from taking a certain action $(a(t))$ while being in state $(s(t))$ the higher the probability $P(s(t+1), s(t), a(t))$ of choosing again this action $(a(t))$ when being at that state $(s(t))$.

As just mentioned the robot needs to explore its environment. This is done by taking random actions and analyse the rewards that would correspond to those actions. On the other hand the robot is trying to exploit its own information about the environment to collect the highest rewards. So it needs to choose whether to get the known high reward or take risks and maybe find an even higher reward. This is also called the exploration-exploitation trade-off (Kober & Peters, 2012). A greedy policy is always choosing for the highest reward which is known by the robot, without further exploring the environment. The ε-greedy method tries to deal with this trade-off (Sutton & Barto, 1998). A robot with an ε-greedy method explores with probability ε and exploit with probability (1- ε).

3.2.1 Problems with the reward function.

Unfortunately, defining the reward function is not a trivial task. This is most easily explained through an example. When it is assumed that a robot is trying to reach a certain point. The robot can then receive a reward when reaching the goal position. Unfortunately, because of the high dimensional character of the robot, it can happen that this goal point is not reached during a trial. The robot receives then a zero reward and it does not know which action was desired and brought the robot closer to the goal position and vice versa which actions made the robot move
away of the goal position and were therefore undesired. The robot therefore needs a shaped reward function. A shaped reward function does not only give a reward when the goal is reached but also when the robot is in a state close to the goal, which guides the robot to the goal position. Another important point for the design of the reward function is that a trade-off needs to be made between precision and speed. The design of this shaped reward function is considered to be difficult and it is usually designed by a robotic expert (Muelling et al., 2014).

The goal of the thesis is to teach the robot a movement without the use of a robotic expert. RL still needs a reward function as is shown in Figure 3-6, which cannot be designed through a non-robotic expert. This is why a look it taken at the alternative methods for creating such a reward function. The main algorithm for creating a reward function from a demonstration of multiple demonstrations is inverse reinforcement learning (IRL), which is discussed in Section 3.3.

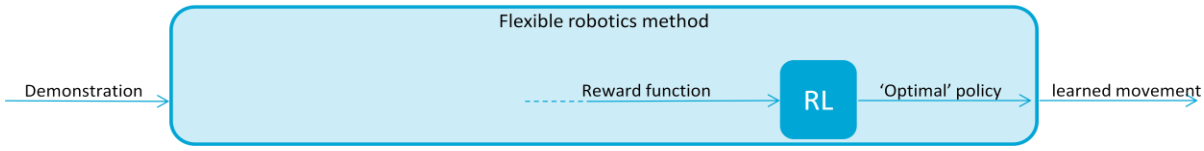


Figure 3-6 - RL creates a policy from a reward function. The reward function can not be given by a non-robotic expert. Therefore RL on its own does not fit to the main goal.

3.2.2 The basic principles of RL

To better understand the basics of RL, the most important principles and formulas are explained. The first important assumption of RL is that the problem conforms the Markov property (Sutton & Barto, 1998). According to the Markov Property the new state s' is only dependent on the previous state s and action a, and information about past states and actions are not included. With RL the robot decides through a probabilistic approach which

action to take; as a consequence the RL is a method that solves a Markov Decision Process (MDP).

The fundament of most RL algorithms is based on maximizing the Bellman Optimality equations (Sutton & Barto, 1998), which are described in the next section.

Bellman optimality equations

Within RL the robot tries to maximise the cumulated rewards obtained by executing a policy. For a task which has a fixed time period T, this means that RL tries to maximize the expected reward over T. Equation 3-1 shows that using a certain policy π the expected return V of state s is the summation of the rewards over a time period T with discrete time steps t.

$$V^{\pi}(s) = E\left\{\sum_{t=0}^{T} R(s', s, a) \,|\pi\right\} \tag{3-1}$$

In most problems the amount of steps that needs to be taken is not known or the amount of steps is infinite. To be sure that the summation converges, the future rewards need be discounted by the discount factor γ as is shown in Equation 3-2. The discount factor γ controls the size of the prediction horizon. The discount factor has typically a value of $0 \leq \gamma < 1$.

$$V^{\pi}(s) = E\left\{\sum_{t=0}^{\infty} \gamma^{t} R(s', s, a) \,|\pi\right\} \tag{3-2}$$

Function V is called the value function which the robot tries to maximize. Next to the value function there is also a state-action value function $Q^{\pi}(s, a)$, which not only depends on the state the system is in but also on the action that can be taken. Equation 3-3 shows the value function and is also known as the Bellman optimality equation (Sutton & Barto, 1998):

$$V^{*}(s) = \max_{\mathrm{a}} \sum_{\mathrm{s'} \epsilon \mathrm{S}} P(s', s, a)\big(R(s', s, a) + \gamma V^{*}(s')\big) \tag{3-3}$$

where $R(s', s, a)$ is the reward function previously described. The optimal value function $V^{\pi^{*}}$ is value function for the optimal policy π^{*}, where V^{*} is the shorthand notation of $V^{\pi^{*}}$

The Bellman optimality equations can be directly used to search for the optimal policy, when the state transition probabilities and value function are known (Sutton & Barto, 1998). Equation 3-4 shows how to find the optimal policy. Here action a^{*} chosen by the policy π^{*} in the current state s which leads to the next state s', which leads to the highest optimal value function V^{*}.

$$\pi^{*}(s) = \arg\max_{\mathrm{a}} \sum_{\mathrm{s'} \epsilon \mathrm{S}} P(s', s, a)\big(R(s', s, a) + \gamma V^{*}(s')\big) \tag{3-4}$$

Equation 3-4 exploits the known information to optimize the policy and does not explore the environment. This means that the equation is a typically greedy methods as mentioned in Section 3.2.

When the states and actions are discrete and finite the value function and policy can be represented through a table (Sutton & Barto, 1998). When for a problem with discrete space the state transition probabilities and the optimal value function are fully explored it is just a matter of looking up in the tables to select the best policy. When the dimensionality of the states and actions are rising, the table grows therefore the computational expenses also

rise. In the case of continuous state-action spaces determining the optimal actions which form the optimal policy is an optimization step on its own (Kober & Peters, 2012).

RL in robotics

When applying RL in robotics there are four major challenges that need to be considered (Kober & Peters, 2012):

- Curse of dimensionality
  For high-dimensional or even continuous spaces, the state and action grow exponentially. As mentioned in Section 3.1.2 IL can be used to obtain an initial policy for the RL method, which can reduce the search space significantly. Another way of reducing the search space is to parameterise the policy as is done with the policy search methods, such as PI^2.

- Curse of real-world samples
  In robotics, real hardware is used which suffers from wear. Therefore, when the real robot finds an optimal policy through trial and error is expensive. Again, when using IL to obtain an initial policy the amount of trials can be reduced. Another approach to minimise the wear and tear of the hardware would be by simulation.

- Curse of Under-Modelling and Model Uncertainty
  As just mentioned, simulation can be used to reduce the cost for real-world interaction. Unfortunately, creating such a simulation that is accurate enough is extremely difficult.

- Curse of Goal Specification
  To define a good reward function can be extremely difficult. Therefore is chosen to look into the IRL algorithms to obtain the reward function from demonstrations. More about IRL and how to obtain a good reward function can therefore be found in Section 3.3.

The RL algorithms which are mostly used in robotics are policy search methods (Kormushev et al., 2013). The idea behind policy search RL is that a smaller policy space is used instead of the huge state and action spaces (Kormushev et al., 2013). The policy space is characterized by the chosen policy parameterization θ and contains all possible policies regarding this certain policy parameterization.

$$\theta_{t+1} = \theta_t + \Delta\theta_t \qquad 3\text{-}5$$

Policy search methods update the policy parameters iteratively per learning step with small time steps t through small changes $\Delta\theta_t$, see Equation 3-5, and drastic changes are prevented. Drastic changes can be hazardous for the robot and its environment. Another reason for only allowing for small changes in the policy is that otherwise the initial policies given or domain knowledge become useless (Peters & Schaal, 2008b; Schaal, 1997). In comparison to the conventional value function-based approaches, such as the Bellman equations 3-4, policy search methods have less dimensionality and an increased convergence speed. Most consolidated policy search algorithms for RL in robotics are (Kormushev et al., 2013):

- Policy gradient search algorithms (Peters & Schaal, 2008a; Williams, 1992)
  With this method, the policy parameters are iteratively updated, through a gradient of the reward function ($\nabla_\theta R$), then $\Delta\theta_t = \alpha \cdot \nabla_\theta R$. Most policy gradient search algorithms have high sensitivity for the learning rate α and exploratory variance, with the exploratory variance is for example meant the ε from the ε-greedy method. The

learning rate parameter is not a trivial task to determine, but is critical for achieving good performance.

- RL based on the Expectation-Maximization (EM) algorithm (Kober & Peters, 2009)
  For this algorithm the learning rate parameter is not needed, which would allow avoiding the issues just introduced.
- Policy improvement with path integrals (PI^2) (Theodorou et al., 2010)
  For this algorithm also no learning rate is needed, similar to the EM based algorithms.
- Stochastic optimization
  Viable alternatives for direct policy search RL can be found in the field of stochastic optimisation (Stulp & Sigaud, 2012), such as cross-entropy (Rubinstein & Kroese, 2013), covariance matrix adaptation evolution strategy (Hansen, 2006). These methods are not well-established in RL research and therefore these methods are not further discussed in this thesis.

Each different policy search method has its advantages and disadvantages. However, an open issue remains when it is appropriate to use a certain method (Kober & Peters, 2012).

For this research is chosen to use PI^2 algorithm of Theodorou et al. (2010). This algorithm was chosen because it is a proven RL approach and it has also a good inverse reinforcement learning (IRL) variant, namely inverse PI^2. More information about inverse PI^2 can be found in Section 3.3.2.

From now on this thesis uses the term cost function, instead of reward function. The cost function is actually a negative reward function as is shown in Equation 3-6. The robot would either try to maximize its reward function or minimize its cost function.

$$C(s,a) = -R(s,a) \tag{3-6}$$

This choice is made because the PI^2 algorithm is based on a cost function that needs to be minimized, instead of a reward function that needs to be maximized.

3.2.3 Policy improvement with path integrals

A promising RL method for robotics is the policy improvement by path integrals (PI^2) algorithm, introduced by Theodorou et al. (2010). For the parameterisation of the policy they use a variant of the DMPs, which are explained in Section 3.5, and use a combination of value function approximation and direct policy learning by approximating a path integral. It is a probabilistic learning method without open parameters such as the learning rate, except for the exploration noise. This exploration noise ϵ_t is Gaussian with a zero mean and variance Σ_ϵ and is added to the policy parameters θ.

` The outline of PI^2 is briefly provided. More details can be found in (Kalakrishnan et al., 2013; Theodorou et al., 2010).

The total cost $C(x)$ of the trajectory x is given by Equation 3-7. The trajectory x is also called the path and is considered to be the degrees of freedom, e.g., the y and z coordinate, over time. Here the cost function is composed of two parts; the terminal cost represented by q_{tN}, and the time depended cost summed over the duration of the trajectory $\int_{t=0}^{T} r_t dt$. Where r_t is described by Equation 3-8.

$$C(x) = \int_{t=0}^{T} r_t dt + q_{t_N} \tag{3-7}$$

$$r_t = q(s,t) + \frac{1}{2}(\theta + \epsilon_t)^T L(\theta + \epsilon_t) \tag{3-8}$$

Equation 3-8 shows that r_t is also split into two parts. The arbitrary state-dependent cost $q(s,t)$ and the control cost represented by the policy parameters with some exploration noise $(\theta + \epsilon_t)$ multiplied by the semi-definite weight matrix L.

For updating the policy parameters the algorithm first creates K roll-outs of the current policy parameters with some exploration noise. A roll-out is a run of the DMPs with some policy parameters to see which trajectory the policy parameters represent. The costs of the roll-outs are calculated based on the corresponding trajectories. The lower the trajectory costs, the higher the probability $P(x_{k,t})$ of taking that trajectory, which is calculated through the formula of Equation 3-12. The parameter λ is defined through L and the variance of the noise, by $\lambda L^{-1} = \Sigma_{\epsilon}$.
The integral of Equation 3-7 is approximated through Equation 3-9 and 3-10.

$$\int_{t=0}^{T} q(s,t)\,dt \rightarrow \sum_{j=t}^{N-1} q_{k_j} \tag{3-9}$$

$$\int_{t=0}^{T} \frac{1}{2}(\theta+\epsilon_t)^T L(\theta+\epsilon_t)\;dt \rightarrow \frac{1}{2}\sum_{j=t}^{N-1}\left(\theta+M_t\epsilon_{k,t}\right)^T L\left(\theta+M_t\epsilon_{k,t}\right) \tag{3-10}$$

Therefore Equation 3-7 can be rewritten as shown in Equation 3-11.

$$C(x) = q_{t_N} + \sum_{j=t}^{N-1} q_{k_j} + \frac{1}{2}\sum_{j=t}^{N-1}\left(\theta+M_t\epsilon_{k,t}\right)^T L\left(\theta+M_t\epsilon_{k,t}\right) \tag{3-11}$$

Afterwards update of the policy parameters per time step is calculated through Equation 3-13. Here the summation over the K roll-outs is taken of the probability of the roll-out $P\left(x_{k,t}\right)$ multiplied by the change that it had evoked, the exploration noise $\epsilon_{k,t}$.

$$P\left(x_{k,t}\right) = \frac{e^{-\frac{1}{\lambda}C(x_{k,t})}}{\sum_{j=1}^{K} e^{-\frac{1}{\lambda}C(x_{j,t})}} \tag{3-12}$$

$$\delta\theta_t = \sum_{k=1}^{K} P\left(x_{k,t}\right) M_t\epsilon_{k,t} \tag{3-13}$$

where M_t projects the exploration noise ϵ_t onto a basis vector so it can be multiplied to the probabilities. When the roll-out has a high cost, the probability $P\left(x_{k,t}\right)$ of this roll-out is low and therefore its influence on the policy parameters is also low.

Finally the update equation for the policy parameters of PI^2 looks as shown in Equation 3-14, for a policy with N time steps.

$$\theta_{m+1} = \theta_m + \frac{1}{N}\sum_{t=1}^{N}\delta\theta_t \tag{3-14}$$

As a summary the pseudo code for PI^2 can be found in Figure 3-7.

Given:

- Parameterized policy (from DMPs) $\pi(\theta)$ with basic functions (g_t), more information about this can be found in Section 3.5.
- Cost function $C(x) = q_{t_N} + \sum_{j=t}^{N-1} q_{k_j} + \frac{1}{2}\sum_{j=t}^{N-1}(\theta + M_t\epsilon_{k,t})^T L(\theta + M_t\epsilon_{k,t})$
- ϵ is zero mean noise with variance Σ

Calculate for:

- Create K roll-outs with $(\theta + \epsilon)$, resulting in K trajectories $\tau_1 \dots \tau_K$
- Calculate for $k = 1 \dots K$ and $t = 1 \dots N$
  - $M_t = \frac{L^{-1} g_t g_t^T}{g_t^T L^{-1} g_t}$ (projection of L on the policy basis functions)
  - $C(x_{k,t}) = q_{t_N} + \sum_{j=t}^{N-1} q_{k_j} + \frac{1}{2}\sum_{j=t}^{N-1}(\theta + M_t\epsilon_{k,t})^T L(\theta + M_t\epsilon_{k,t})$
  - $P(x_{k,t}) = \frac{e^{-\frac{1}{\lambda}C(x_{k,t})}}{\sum_{j=1}^{K} e^{-\frac{1}{\lambda}C(x_{k,t})}}$
  - $\delta\theta = P(x_{k,t}) M_t \epsilon_{k,t}$
- $\partial\theta = \sum \delta\theta$
- $\theta_{new} = \theta_{old} + \partial\theta$ → (if $\delta\theta = P(\tau) M_t(\theta + \epsilon_{k,t})$ then $\theta_{new} = \sum \delta\theta$

Figure 3-7 - Pseudo Code for PI²

3.3 Inverse reinforcement learning

With IRL a cost function can be obtained from demonstrations given by a non-robotic expert, as is shown in Figure 3-8. This cost function can be used to learn the policy and therefore the task by RL afterwards.

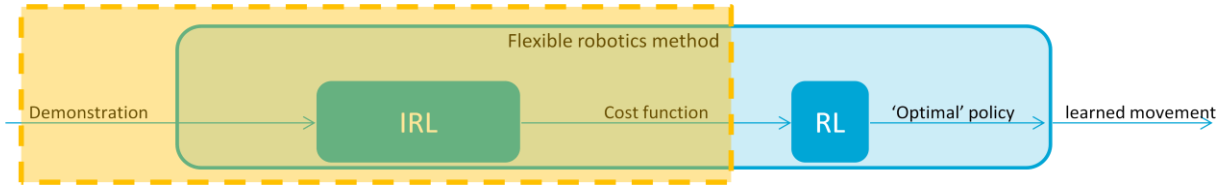


Figure 3-8 - IRL creates a cost function from a demonstration or multiple demonstrations.

First the fundaments of the IRL algorithms are discussed in Section 3.3.1. Afterwards, in Section 3.3.2 the chosen IRL method is explained in more details.

3.3.1 The basic principles of IRL

For the explanation of the fundaments of IRL, the term reward function instead of cost function is used again. When explaining the algorithm inverse PI² the term cost function is used again.

Most IRL approaches assume that the reward function can be expressed as a linear combination of the reward features (Abbeel & Ng, 2004) as shown in Equation 3-15.

$$R(s) = w^T \cdot \phi(s) = w_1 \cdot \phi_1 + w_2 \cdot \phi_2 + \cdots + w_n \cdot \phi_n \qquad 3\text{-}15$$

In which $\phi(s)$ is a vector of n reward features over the states and w is a vector of the corresponding weights. Those reward features are task depended and defined before running the IRL algorithm. The IRL algorithms calculate the weights vector.

When the demonstrated policy π^D is assumed to be the optimal policy π^*, Equation 3-16 must hold (Ng & Russell, 2000).

$$E_{s' \sim P_{ss'}^{a_D}}[V^{\pi}(s')] \geq E_{s' \sim P_{ss'}^{a}}[V^{\pi}(s')] \qquad 3\text{-}16$$

When multiple demonstrations are given π^D is the average of those. Equation 3-16 states that the expected value function, see Equation 3-2, of the next state (s') must be always larger when taking the demonstrated action a_D according the demonstrated policy (π^D) than taking any other action ($a \neq a_D$). More formally:

Equation 3-17 can be obtained, by combining Equation 3-2 and Equation 3-15.

$$V(\pi) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \middle| \pi\right] = E\left[\sum_{t=0}^{\infty} \gamma^t w^T \cdot \phi(s_t) \middle| \pi\right] = w^T \cdot E\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \middle| \pi\right] \quad \text{3-17}$$

The next step would be to define the feature expectation $\mu(\pi)$ as is shown in Equation 3-18.

$$\mu(\pi) = E\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \middle| \pi\right] \quad \text{3-18}$$

The value function can then be rewritten according to Equation 3-19.

$$V(\pi) = w^T \mu(\pi) \quad \text{3-19}$$

Assuming that the demonstrated policy is the optimal policy it can be stated that the value function of the demonstrated policy π^D is always bigger or equal than any other value function of policy $\pi^i \neq \pi^D$. This was also stated in Equation 3-16 which eventually leads to Equation 3-20.

$$w^T \mu(\pi^D) \geq w^T \mu(\pi^i) \quad \text{3-20}$$

By satisfying this condition a two main issues arise. The first one is that constraint 3-20 is ill-posed. The trivial solution of $w = 0$ is obviously always an answer. There are several approaches to deal with this ambiguity problem. The second issue is that the weights need be updated iteratively, and this makes the process computational expensive.

Ambiguity problem

Since it is tried to approximate the true $R(s)$ 3-15, there may not exist R other than $R = 0$ that would provide the optimal policy. Therefore constraint 3-20 can be relaxed in the optimisation problem by adding penalty $q(x)$, with q being $q(x) = x$ if $x \geq 0$ and $q(x) = cx$ otherwise with c being a positive constant (Abbeel & Ng, 2004).

In Equation 3-21 the expression $\sum_{s \in S_0} \min_i \left(q\left(w^T \mu(\pi^D) - w^T \mu(\pi^i)\right)\right)$ is used to find the policy π^i which is closest to the demonstrated policy π^D. Afterwards the difference between those two policies is maximized by changing w to obtain the *true* weights w^* of the 'true' reward function.

$$w^* = \underset{w}{\operatorname{argmax}} \left(\sum_{s \in S_0} \min_i \left(q\left(w^T \mu(\pi^D) - w^T \mu(\pi^i)\right)\right)\right) \quad \text{3-21}$$
$$s.t. \left|w_j\right| \leq 1, \qquad i = 1, \dots, d$$

There are also other approaches created to solve the drawbacks of Equation 3-20 (Abbeel & Ng, 2004), such as the max margin approach (Abbeel & Ng, 2004; Ratliff et al., 2006), and the Bayesian IRL (Ramachandran & Amir, 2007) among others.

Computational Burden
Defining the weights w of the reward function involves an iterative procedure. In which the following steps are taken (Kalakrishnan et al., 2013):

1. Find the optimal policy for the current estimate of the reward function, for example by using Equation 3-4.
2. Update the reward function estimate, by using for example Equation 3-21, using the output of the previous step.

For solving step 1 a RL algorithm needs to be used to test the estimate 3-21 of the weights w of the reward function and this is a computational expensive step on its own. In step 2 the output of the RL algorithm is then used to update the reward function 3-21. Iteratively solving for the weights (w) of the reward function 3-15 is heavily computational expensive. Although the approach is shown to be working well for discrete state-action spaces of low dimensionality or simple theoretical cases (Abbeel & Ng, 2004; Ramachandran & Amir, 2007; Ratliff et al., 2006), unfortunately for robotics in real-life scenarios this becomes difficult (Zhifei & Joo, 2012).

A few attempts are made to apply IRL in real-life domains have been conducted. For example Muelling et al. (2014) uses IRL to find the underlying strategies for table tennis.

A successful applied IRL algorithm used for a robotic arm is created by Kalakrishnan et al. (2013). They use the RL policy search method, PI^2 (Policy Improvement with Path Integrals) algorithm and reverse it for finding the reward function. More information about PI^2 algorithm can be found in Section 3.2.3.

Another disadvantage of IRL is that it assumes the demonstration to behave optimally. It can handle multiple demonstrations, but it is assuming that all given demonstrations are equally important.

3.3.2 Inverse PI^2

For PI^2 the cost function is given in Equation 3-11. This definition of the cost function can almost be used directly as the cost features (Φ) for an IRL approach as is shown in Equation 3-22:

$$C(x_{k,t}) = \begin{bmatrix} w_q \\ w_R \\ w_{q_{t_N}} \end{bmatrix}^T \begin{bmatrix} \sum_{j=t}^{N} \phi_{k,j} \\ \frac{1}{2}(\theta + M_t \epsilon_{k,t})^T L (\theta + M_t \epsilon_{k,t}) \\ \phi_{t_N}^k \end{bmatrix} = w^T \Phi \qquad 3\text{-}22$$

here is assumed that the state-dependent cost function is linearly parameterized through with user provided features $\sum_{j=t}^{N} \phi_{k,j}$, in other words: $\sum_{j=t}^{N-1} q_{k_j} = w_q^T \sum_{j=t}^{N} \phi_{k,j}$. Another assumption is that the shape of the quadratic control cost matrix is given and can be scaled through the weight factor w_R is a scaling factor, thus $L = w_R \hat{L}$. Where $\hat{L}$ is the shape of the control cost. The final assumption is that the terminal cost q_{t_N} is are the terminal cost feature $\phi_{t_N}^k$ times the weight factor $w_{q_{t_N}}$. The full weight vector of the cost function can be found through solving for the constrain 3-23:

$$\mathrm{w}^* = \underset{w}{\operatorname{argmin}} \left(-\log \frac{e^{-w_i^T \Phi_i^*}}{\sum_{i=1}^{K} e^{-w_i^T \Phi_{i,k}}} + B|w|^1 \right) \quad \text{3-23}$$

where the second term is the regularization term, here the weights are penalized by the sum of the its absolute value ($B|w|^1$) of its vector, and B is a predefined constant to determine the size of the regularization term. This regularization term is to prevent the weight vector from becoming larger and larger at each iterative step. Unfortunately, taking the absolute value of the weights $|w|$ in the cost function, make the standard iterative optimisation models less appropriate. Therefore a slightly more elegant algorithms need to be used, e.g. Orthant-Wise Limited-Memory Quasi-Newton (OWL-QN) (Andrew & Gao, 2007). Another approach, to be able to use the standard iterative optimisation methods, is to choose a different regularization term, through taking the squared ($p = 2$) or even a higher power p in $B|w|^p$.

To run the inverse PI^2 K noisy versions of the demonstrated trajectory needs to be created. Inverse PI^2 assumes that those noisy versions of the demonstrated trajectory are less optimal, than the demonstrated trajectory and by this it tries to find the cost function that has the lowest cost for the demonstrated trajectory and a higher cost for the noisy demonstrations

As a summary, the pseudo code for inverse PI^2 is given in Figure 3-9.

Given:

- Shape of the cost function $\phi = \begin{bmatrix} \sum_{j=t}^{N-1} q_{k_j} \\ \frac{1}{2} \sum_{j=t}^{N-1} \left(\theta + M_t \epsilon_{k,t}\right)^T L \left(\theta + M_t \epsilon_{k,t}\right) \\ \phi_k \end{bmatrix}$
  - Which include the features q_{k_j}, ϕ_k
- D demonstrated trajectories $x \dots x_D$
  - Parameterized policy (from DMPs) $\pi(\theta)$ with basisfunctions (g_t)
  - Including the noisy versions of that demonstrated trajectory. Here some exploration noise is added to the policy parameters $\theta + \epsilon$ is used.

Calculate:

- $w^* = \mathrm{argmin}_{\mathrm{w}} \left(-\sum_{i=1}^{D} \log \frac{e^{-w^T \phi_i}}{\sum_{j=1}^{K} e^{-w^T \phi_{k,i}}} + B|w|^p \right)$

Figure 3-9 - Pseudo Code for inverse PI^2

3.4 Preference learning

For this research is chosen to focus on using the preferences of a non-robotic expert to obtain a weight factor for the demonstrations before putting it in an IRL approach, as is shown in Figure 3-10. The gathered demonstrations can be put into the IRL algorithm weighted, instead of assuming that the demonstrations are equally important. Similarly is done by Sugiyama et al. (2012), though they used it for a different application namely for a dialog control problem.

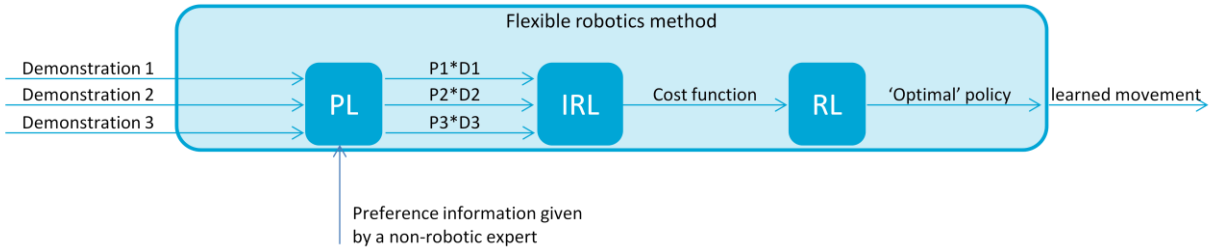


Figure 3-10 - Adding preference information (P1,P2,P3) in form of a weight factor to the demonstrations before putting it in the IRL algorithm.

To better understand the possibilities of preference learning (PL), the basic principles are explained in the next section.

3.4.1 The basic principles of PL

There exist several approaches to include preference learning (PL). Among them, the focus in this thesis is given on general preference-based learning Fürnkranz and Hüllermeier (2010). For an extensive overview on preference learning and its diverse approaches see Fürnkranz and Hüllermeier (2010) and Silva et al. (2006) among others.

In general preference-based learning attempts to learn a preference model (order relations) from observed preferences and is a subfield of supervised learning (Sugiyama et al., 2012). The output of a PL algorithm can be divided into three categories; Label ranking, instance ranking and object ranking (Fürnkranz & Hüllermeier, 2010).

A quick overview off all three the categories can be found in Table 3-1. With label ranking the demonstrations are put into different categories which have no order with respect to each other. With Instance ranking the demonstrations are also put into categories which have an order with respect to each other. Finally, with object ranking a function is obtained which puts the demonstrations on order.

Table 3-1 - The three categories of Preference Learning; Label ranking, Instance ranking or Object Ranking

Category	Output	More details
Label ranking	A given set of labels	With this approach the demonstrations are ranked as either P=a, b or c
Instance ranking	A given set of labels which have an order with respect to each other.	With this approach the demonstrations are classified as P=1,2,3,4,5 Which would represent like 1 Bad 2: Not so good 3: Neutral 4: Close to good 5: Good
Object ranking	An ranking function is obtained	With this approach the demonstrations can be ranked by a formula. P(D)=..

This research is looking into a manner to rank the demonstrations with an order. Therefore, only instance and object ranking are relevant. Ideally, when ranking the demonstrations, the ranking would give the most information about how well the demonstrations are performing the task. This would lead to be using object ranking.

Unfortunately, object ranking is not useful for this research because of several reasons. The first disadvantage: for creating such an object ranking model a test set, features are needed and with this model new demonstrations can be ranked. That is not in correspondence with the goal of the research. This research assumes to have a D demonstrations and it wants to use all demonstrations for learning to imitate the behaviour. Another disadvantage of using object ranking is that with this approach a cost function is created, then only known as the ranking function. It would not be useful to create a cost function twice. So the problem that needs to be solved is most similar to Instance ranking, here the demonstrations would be put in an order with respect to each other. More information on how the preferences are used can be found in Section 4.1.2.

The preference information can be retrieved through three types of input. (Dembczyński et al., 2010)

- Pair-wise comparison; here two demonstrations of the complete set are compared to each other.
- Complete order; here the full set of demonstrations is put into an order.
- Rating on a finite scale; here the demonstrations are given a rate about how well they perform.

According to Thurstone's law of Comparative Judgment in general for humans the set of pair-wise comparisons is the most intuitively to give (Thurstone, 1927). A pair wise preference of two actions looks as is shown in Equation 3-24 (Cheng et al., 2011):

$$a_i \succ_s a_j \iff P\left(x(a_i) \sqsupset x(a_j)\right) > P\left(x(a_j) \sqsupset x(a_i)\right) \qquad 3\text{-}24$$

here taking action a_i is preferred over taking action a_j while being at state s, and $P(x \sqsupset y)$ is the probability that trajectory x is preferred over trajectory y. Trajectory is the path that has been taken. When action a_i is preferred over action a_j the probability of taking action a_i should also be higher than the probability of taking action a_j.

By collecting all the pair wise preferences of the combinations of two demonstrations from the whole set, the final order can then be obtained.

3.5 Dynamic movement primitive

Dynamic Movement Primitive (DMP) is a method to parameterize the policy Ijspeert et al. (2013). In robotics most algorithms use such a parameterized policy; this is due to the high-dimensional character of its problems.

One of the most common algorithms of parameterization of the policy is DMPs, because of their simplicity. DMPs are a specific form of the nonlinear dynamic motor primitives. The nonlinear dynamic motor primitive was introduced by Ijspeert et al. (2002b). PI^2 and inverse PI^2 are also using DMPs for the parameterization of the policy.

3.5.1 The basic principles of DMPs

This section starts with the definition of the nonlinear dynamic motor primitives of Ijspeert et al. (2002b). Afterwards, more details are given about the DMPs.

Nonlinear dynamic motor primitives assume that the movement plan for each degree of freedom is represented by a second order differential equation as is shown in Equation 3-25:

$$\ddot{q}_d = f(q_d, \dot{q}_d, z, g, \tau, \theta) \qquad 3\text{-}25$$

where $q_d, \dot{q}_d$ and $\ddot{q}_d$ are the desired position, velocity, and acceleration for a degree of freedom. The internal state of the dynamical system is z which evolves according to a canonical system $\ddot{z} = f_c(z,t)$, where t is representing the time. The goal state is g of each degree of freedom, τ is the movement duration and θ are the open parameters, also called the policy parameters.

A DMP is a damped spring model with non-linear terms which represents the acceleration of each degree of freedom. These dynamic motion primitives can be divided into two categories, based on the kind of task they represent: point attractors or oscillators. For a point attractive DMP, the degree of freedom is moved from a start position to a goal position (Ijspeert et al., 2013). With an oscillatory DMP the degree of freedom is performing a repetitive movement, so the start and goal state are equal (Ijspeert et al., 2013). Since for

this research only the point attractive DMPs are used, only this category is briefly explained below. A more in-depth explanation can be found in Ijspeert et al. (2013).

Discrete movements:
Equation 3-26 is the main equation to describe the acceleration through a DMP of a point attractive motion:

$$\tau \ddot{q}_d = \alpha_v(\beta_v(g - q_d) - \dot{q}_d) + f(x) \qquad 3\text{-}26$$

here q_d is the state of the degree of freedom the DMP is describing. The goal state of the DMP is given through g, and the gains α_v and β_v are predefined constants. Whilst the first term of the equation describes a damped spring model which makes it move from the start state to the goal state, the second part corresponds to the forcing term that modifies how it moves from the start state to the goal state. The forcing term $f(x)$ is shown in Equation 3-27:

$$f(x) = \frac{\sum_{i=1}^{N} \psi_i(x)\theta_i}{\sum_{i=1}^{N} \theta_i} x(g - q_0) \qquad 3\text{-}27$$
$$\text{Where: } \psi_i(x) = \exp\left(-\frac{1}{2\sigma_i^2}(x - c_i)\right)$$

where the initial state is described as q_0, θ_i are the policy parameters and σ_i and c_i are the defining the width and the centre of the Gaussian basis function ψ_i. This forcing term uses an extra nonlinear function $\dot{x}$ to define itself over time (t), shown in Equation 3-28:

$$\tau \dot{x} = -\alpha_s x(t) \qquad 3\text{-}28$$

where τ is a time constant, α_s is a predefined constant, t is the time and x is the phase variable, which is used to describe the forcing term.

Given:
- The gain parameters: α_v, β_v
- Discreet path of one degree of freedom of the whole trajectory in time steps (dt)
  Position (q) , velocity $(\dot{q})$, and acceleration $(\ddot{q})$

Calculate:
- Calculate the phase variable:
  - $\tau \dot{x}(t) = -\alpha_s x(t) \quad \rightarrow \quad x(t) = (1 + t)e^{-\alpha_s t}$
- Calculate centre and width of Gaussian basis functions
  - $c(t) = t_{start} : \frac{t_{end} - t_{start}}{nrefs} : t_{end}$
  - $c\big(x(t)\big) = c(x)$
- Gaussian basis functions $\psi = \exp\left(h \cdot (x - c)^2\right)$
- $f_{target}(s) = \frac{-\alpha_v(\beta_v(g - q_d) - \dot{q}_d) + \tau \ddot{q}_d}{(g - q_0)}$ with g is the goal state/position
- $f(s) = \frac{\sum \psi \cdot x}{\sum \psi} \theta$
- Solve for the policy parameters through: $\min_\theta(\sum_s \left(f_{target}(s) - f(s)\right)^2$

Figure 3-11 - Pseudo Code for the initialization of a point attractive DMP.

3.6 Summary

In this Section the algorithms IL, RL, IRL, PL and DMP are explained. IL directly uses the demonstrations to learn the task. The main disadvantage of IL is that the performance of

the task completely depends on the performance of the demonstration. In other words IL can only generalise to the shown movement, it can not learn a completely different movement.

With RL a cost function is needed to learn a task, defining such a cost function is a difficult task. Therefore, IRL was introduced. RL in robotics are often policy search methods to reduce the dimensionality of the optimisation problem. For this research is chosen to use the PI^2 algorithm.

With IRL the demonstrations of a non-robotic expert can be used to obtain a cost function. For IRL is chosen to use the inverse PI^2 algorithm.

For collecting the preferences of the non-robotic expert is chosen to use the pairwise preferences, since they are most intuitively to give for humans. Those pair-wise preferences are used to obtain an subsequentaly order of the demonstrations.

For PI^2 and inverse PI^2 the trajectories need to be parameterised through the DMPs. Each degree of freedom has its own DMP, which is a second order differential equation.

4. Method

As mentioned in Section 2 most of the current IRL algorithms are not utilising the effect of the input demonstrations for the learning process. With hypothesis 1 the effect of multiple demonstrations as input is studied. It is expected that adding more demonstrations does not necessary improve the performance but that it depends on the average performance of the input demonstrations. With hypothesis 2 the effect of adding the preferences of the non-robotic expert on the demonstrations is studied.

For this research, the demonstrations and preferences of a non-robotic expert need to be gathered. How those demonstrations and preferences are gathered is explained in Section 4.1. This data is afterwards used to learn an optimal policy through inverse PI^2 and PI^2 as is shown in Figure 4-1. The selection of the parameters for the inverse PI^2 and PI^2 algorithms is shown in Section 4.2.

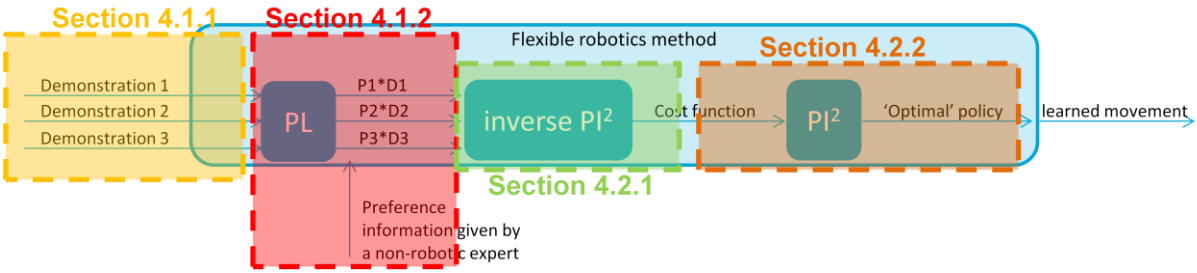


Figure 4-1 - Graphical outline of this chapter.

4.1 Gathering of demonstration data

For testing the hypotheses two data sets are required: a set of demonstrations and information on the quality of the demonstration. As explained in more detail in Section 4.1.1, gathering the demonstrations is necessary to evaluate both hypotheses. Additionally, preferences of the non-robotic expert are collected to determine the perceived performance of demonstrations and test the second hypothesis. Section 4.1.2 explains how the preferences are collected, which give information about how well the demonstrations were performing the task according to the evaluator.

4.1.1 Demonstrations

Five steps need to be taken to gather the data of the demonstrations, as is shown in Figure 4-2. The steps are described in the sequential order in the remainder of the paragraph.

Figure 4-2 - Steps to obtain input trajectories

Record
The demonstrations are obtained through kinaesthetic teaching, where the demonstrator grabs the robot and pulls it according to the desired motion. During this kinaesthetic teaching, the relevant coordinates need to be recorded.

Aligning
Before the demonstrations are parameterised and fed to on the robot, the demonstrations need to be aligned by time. Alignment is necessary for better comparison of the demonstrations. The most common way of aligning demonstrations is to use dynamic time

warping. This method assumes that all demonstrations would have the same execution time and would change the velocities and acceleration to scale all the demonstrations in one unique time frame. Dynamic time warping is often used in task such as speech recognition (Koenig et al., 2008; Sakoe & Chiba, 1978), since with dynamic time warping different time spans can be compared to see if the same word is said.

Since one of the performance measurements is the execution time, rescaling all demonstrations to one unique time frame is not desirable. This makes dynamic time warping not usable. Therefore alternatively, it was decided to take an alignment point which is characteristic of the movement. Afterwards, the time is scaled so all demonstrations fit.

Parameterise the demonstrations
When using inverse PI^2 and PI^2 the recorded coordinates need to be parameterized through DMPs, as mentioned in Section 3.5.

Replay the DMPs of the demonstrations on the robot.
Once the demonstrations are parameterised they need to be fed back on the robot. This is necessary because the DMPs are approximating the coordinates of the trajectory. While those trajectories are fed back to the robot a movie is recorded. Those movies are used to obtain the preferences of a non-robotic expert as is explained in more detail in Section 4.1.2. It is necessary to make the movies on the trajectories of the DMPs since they are the input for the algorithm and therefore also the preferences of the non-robotic expert needs to make on those trajectories. Otherwise, when the performance measurements are evaluated on the original kinaesthetic teaching trajectories, this gives a bias between what is used as an input for the inverse PI^2 algorithm, namely the DMPs, and what is evaluated through the non-robotic expert, the original trajectories.

4.1.2 Extracting preferences over the demonstration

The demonstrations need to be evaluated by a non-robotic expert. This evaluation leads to a ranking system. In order to retrieve the ranking values of each demonstration, it was decided to use pair-wise preferences gathered through a crowdsourcing experiment. The set-up of the experiment was inspired on the work of Grappiolo et al. (2014). Furthermore, the correlation between the perceived performance gathered through the crowdsourcing experiment and the performance measurements needs to be checked, as it is mentioned in Section 2.2. The crowdsourcing experiment is organized as an online survey, in order to maximize the chances to obtain preferences from non-robotic experts. In this survey people were asked to give their preference between two demonstrations; demonstration A and demonstration B. The answers that could be given were:

a) demonstration A performs better than demonstration B (A>B)
b) demonstration A performs worse than demonstration B (A<B)
c) demonstration A and demonstration B are performing equally (A=B)
d) I don't know which one is performing better (A?B)

Figure 4-3 depicts the layout of the questionnaire, the full questionnaire can be found in Appendix E. The movies could be replayed and stopped at each time step. Participants were asked to rate 7 combinations of demonstrations, which in total would take around 15 to 20 minutes to complete the survey. The combinations of the demonstrations used in the survey were chosen randomly but is taken into account that each combination has roughly the same amount of preferences.

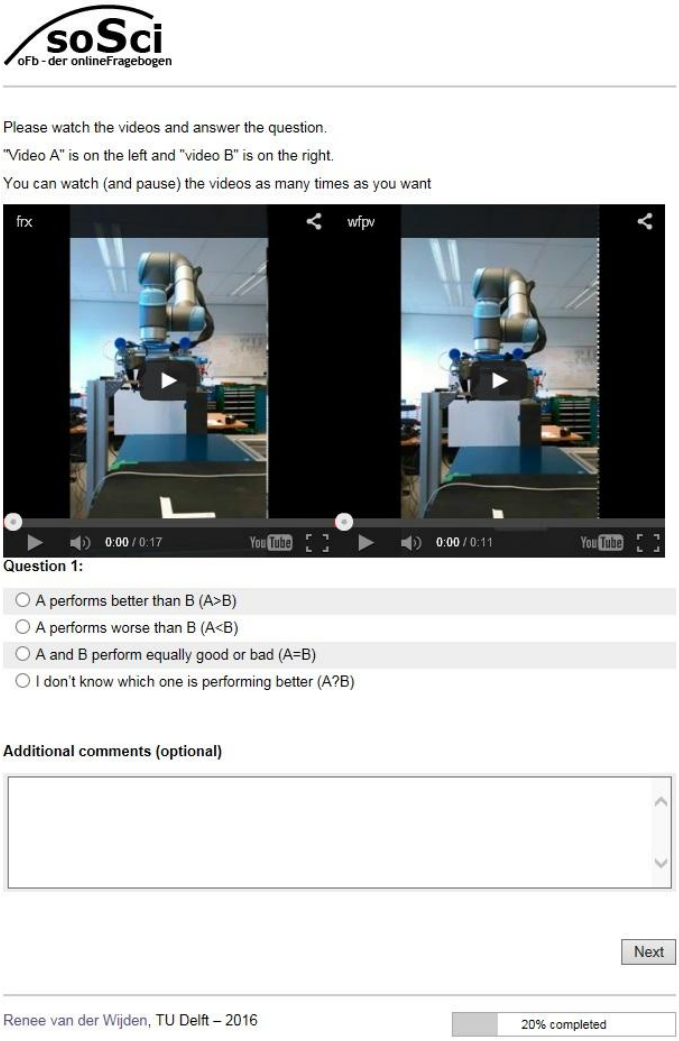


Figure 4-3 - Screenshot of the crowdsourced questionnaire

Analysis of the correlation

To calculate the correlation between the performance measurements and the perceived performance of the non-robotic expert Equation 4-1 is used (Grappiolo et al., 2014).

$$c(p) = \sum_{i=1}^{N} \frac{p_i}{N} \qquad 4\text{-}1$$

Where N is the amount of pairwise preferences available and

- $p = 1$ if the performance measurement agrees with the preference
- $p = -1$ when they do not agree.

The $c(p)$ ranges from -1 when there is absolute disagreement and 1 when there is complete agreement.

Creating weight factors from the crowdsourcing experiment data

This paragraph describes how this research uses the preferences of the crowdsourcing experiment to obtain weight factors that can be used as input for inverse PI[2]. First, the overall order of the gathered demonstrations needs to be gathered. Therefore, it is important to create an overall performance measurement P. As is shown in Equation 4-2 this performance measurement exist out of the n performance measurements times the weight vector $\boldsymbol{v}$. The normalisation of the performance measurements is done through a min-max normalisation,

done similarly as described in Equation 4-4, furthermore they are rewritten in such a manner that a higher value also means a higher performance. Important is that the overall performance measurement P has a high correlation $c(P)$, Equation 4-1, with the perceived performance of the non-robotic expert. Therefore, to find the weight factor $\boldsymbol{v}$ the multivariable optimisation problem, shown in Equation 4-3, was solved through the *fmincon* function of MATLAB *(MathWorks, 2016)*. This optimisation was done with random initial values for $\boldsymbol{v}$ and multiple times to make sure that not only a local optimum was found. This overall performance measurement can be used next to evaluating the Demonstrations to evaluate also the new learned trajectories.

$$P = \boldsymbol{v}\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ : \\ p_n \end{bmatrix} \tag{4-2}$$

$$\boldsymbol{v} = \underset{v}{\operatorname{argmax}}\,(c(P(\boldsymbol{v}))) \tag{4-3}$$

When this overall performance measurement is created, the demonstrations can be put into an order. When used as input for inverse PI^2 two options exist:

1. The correlated overall performance measurement is directly used as the weight factor. The weight factors now not only contain which demonstration is performing better, but also how much better.
2. Only the corresponding order of the input demonstrations is used for creating the weight factor. In this case is only known which demonstration is performing better than the other, and not how much better.

In the first case, the performance measurements are used. Therefore, it is known how much better demonstration A is over demonstration B. When the overall performance measurement of demonstration A is much higher than the performance measurement of demonstration b than the weight factors for example would be looking like $P_a = 0.812$ and $P_b = 0.188$. The weight factors are normalized to make sure they sum up to one.

For the second case is only known that for example demonstration A is better than demonstration B. Therefore, demonstration A would obtain a twice as high weight factor than demonstration B, $P_a = \frac{2}{3}$, and $P_b = \frac{1}{3}$.

The first option contains more information on the performance, but in reality often only a sequential order of the demonstrations would be available. This brings the second option closer to reality. In the end, when applying this framework in real life only a handful of demonstrations would be recorded and rated.

4.2 Parameter selection of the learning algorithms

Now that the demonstrations and preferences are gathered the learning process can start. The robot first learns a cost function through inverse PI^2 and afterwards PI^2 is used to learn a trajectory, as is shown in Figure 4-4.

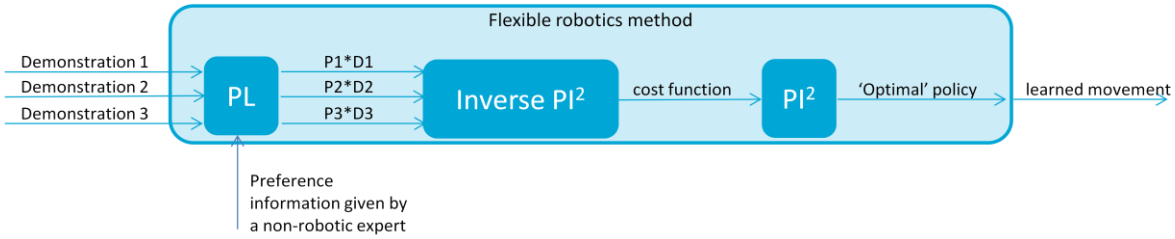


Figure 4-4 - Flow diagram of how the policy is learned through IRL and RL

For applying the inverse PI^2 and PI^2 algorithm some parameters, e.g. the cost features and exploration noise, need to be selected. Section 4.2.1 explain the parameters for inverse PI^2 and Section 4.2.2 explains the parameters for PI^2.

4.2.1 The parameters for inverse PI^2

According to the pseudo-code of inverse PI^2, shown in Figure 3-9, three parameters need to be set: the cost features, the exploration noise to create the noisy trajectories of the existing trajectory and the regularization term.

Cost features for inverse PI^2

For inverse PI^2 the cost function exists linearly out of different cost features $\phi(x_k, t)$, as is shown in Equation 3-22. Those cost features need to be determined by hand. Ideally, a generic set of features could be used, which means that this set of features can be used for multiple tasks. Unfortunately, defining such a set of features would not be a trivial task to do. Therefore, it is chosen to use task specific features.

Since the cost features determine how the cost function looks, they also determine how well the learned trajectory is performing. Therefore a relevant starting point for determining the cost features is to look at the performance measurements. Afterwards, some 'steering' features are discussed. With steering, it is meant that although they not directly are connected with the performance measurements they help to obtain the trajectory that is fulfilling the task.

Normalizing the features

Important for inverse PI^2 to work properly is that the features are all in the same range; otherwise features can become dominant. For example, one feature is having a maximum value of 1000 while the other one has a maximum of 0.1. If they are equally important, the weights need to be respectively 0.0001 and 1. This might lead to a disturbed outcome when using those features.

Among the commonly used methods to normalize features is chosen for the min-max normalization, because of its simplicity. Equation 4-4 shows the normalization method.

$$\phi_{lnorm} = \frac{\phi_l - \min(\phi_l)}{\max(\phi_l) - \min(\phi_l)} \qquad 4\text{-}4$$

The $\min(\phi_l)$ is the minimal value of that feature ϕ_l and $\max(\phi_l)$ the maximal value. The min and max value are determined by taking respectively the lowest and highest occurring feature value in the set of gathered demonstrations.

Exploration noise

The exploration noise in inverse PI^2 is used to create trajectories that are close to the demonstrated trajectory but not similar. As mentioned before in Section 3.3.2, Inverse PI^2 uses the difference between the noisy versions of the demonstrated trajectory and the demonstrated trajectory to obtain the cost function. The exploration noise should therefore not have an as low as possible value so the weights are more accurate to the demonstrated trajectory. On the other hand, the exploration noise should be high enough to detect

differences between the noisy versions of the demonstrated trajectory and the demonstrated trajectory.

Regularization term

The standard regularization term that is described for inverse PI^2 is $B \cdot |\boldsymbol{w}|$. Table 4-1 shows the effect on the learned weights when changing this regularization term. The table shows an example where inverse PI^2 has to learn with a random subset of features to check what the exact effect is on the learned weights when changing the regularization term.

Table 4-1 – The table shows what happens if the B and p parameters of the regularization term $B|w|^p$ are varied. Here is learned from 1 demonstrations and only took into account 8 random features. The weights were bounded through the optimisation method from -1 to 1.

	p=1	*p=2*	*p=4*	*p=6*
B=1	$\boldsymbol{w} = \begin{bmatrix} 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \end{bmatrix}$	$\boldsymbol{w} = \begin{bmatrix} 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \end{bmatrix}$	$\boldsymbol{w} = \begin{bmatrix} 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \end{bmatrix}$	$\boldsymbol{w} = \begin{bmatrix} 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \end{bmatrix}$
B=10	$\boldsymbol{w} = \begin{bmatrix} -0.0000 \\ 1.0000 \\ 1.0000 \\ 0.9999 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \end{bmatrix}$	$\boldsymbol{w} = \begin{bmatrix} 0.4504 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \end{bmatrix}$	$\boldsymbol{w} = \begin{bmatrix} 0.6078 \\ 0.9309 \\ 1.0000 \\ 0.8444 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \end{bmatrix}$	$\boldsymbol{w} = \begin{bmatrix} 0.6842 \\ 0.8838 \\ 0.9450 \\ 0.8336 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \end{bmatrix}$
B=100	$\boldsymbol{w} = \begin{bmatrix} -0.0000 \\ 0.0000 \\ 0.0000 \\ -0.0000 \\ 0.0000 \\ 0.9996 \\ 0.9987 \\ 0.9974 \end{bmatrix}$	$\boldsymbol{w} = \begin{bmatrix} 0.0488 \\ 0.1723 \\ 0.2458 \\ 0.1705 \\ 0.4040 \\ 1.0000 \\ 0.8692 \\ 0.7041 \end{bmatrix}$	$\boldsymbol{w} = \begin{bmatrix} 0.2900 \\ 0.4412 \\ 0.4946 \\ 0.4287 \\ 0.5746 \\ 0.9722 \\ 0.7548 \\ 0.7045 \end{bmatrix}$	$\boldsymbol{w} = \begin{bmatrix} 0.4380 \\ 0.5637 \\ 0.6032 \\ 0.5495 \\ 0.6563 \\ 0.9064 \\ 0.7772 \\ 0.7462 \end{bmatrix}$

Table 4-1 shows that having a higher B value lowers the value of the weights. Additionally, when having a high B the learning takes longer. When the value of p is higher the 'high' weights are heavier penalized then the 'low' weights, so the values of the weights converge more towards each other.

For this thesis, a p-value of 4 is chosen. The arguments for this choice are the variations in the weights and the weights are not too close to each other. The B value was kept relatively low, namely, $B = 10$, because of the learning speed.

4.2.2 The parameters for PI^2

For this research is chosen to use PI^2 as a reinforcement learning algorithm, as is mentioned in Section 3.3.2. The learning is done in a simulated world, to prevent wear and tear on the robot. With RL this learning in simulation significantly speeds up the learning process. The PI^2 algorithm needs four parameters to guide the learning process:

- The cost function
- The initial trajectory
- Amount of iterations
- Exploration noise

The cost function that is used for PI^2 comes directly from inverse PI^2. For the initial trajectory is chosen to use the mean of all demonstrations. The reason for this being that a DMP close to the solution is needed and it is not wanted to reuse one of the demonstrations as initial learning trajectory. Using one of the demonstrations is not possible since in theory that input demonstration is the best solution for the cost function. So to see whether RL actually learns a policy another new trajectory was needed, that has the same starting state and final state.

The amount of iterations that needs to be taken is a consideration. On one side it is important that there are enough iterations to converge, on the other side taking more iterations means that the learning takes longer. The amount of iterations needed also depends on the exploration noise, when per iteration a lot of exploration is done less iteration are needed to fully explore the state space.

The exploration noise can be determined through two parameters. The first parameter controls the size of the exploration noise. The value of the size of the exploration noise is typically between zero and one. In other words the size of the noise which is added to the DMP policy parameters of the initial trajectory for exploration. The second parameter to control the exploration noise is how many new roll-outs are made at each learning step or iteration. When more roll-outs are re-used the algorithm explores less of the environment. When no roll-outs are re-used, no learning occurs because at each iteration the information of the previous run is thrown away.

5. Task

Chosen is to focus on one specific task for validating the hypotheses. This section describes the chosen task that is performed. Section 5.1 gives the description of the task and Section 5.2 describes the performance measurements.

5.1 Task description

Even though multiple tasks can be learned with the same IRL and RL approach, for this research is chosen to focus on one specific task. This since the research wants to prove that ranking the demonstrations is beneficial and is not focussed on the flexibility of the algorithms.

The production task that is taken into account for this research is the breaking off of the three floes from the injected moulded part. Figure 5-1 shows the injected moulded part. The task was chosen because it is a real production task that is currently done by humans and too expensive to install a robot.

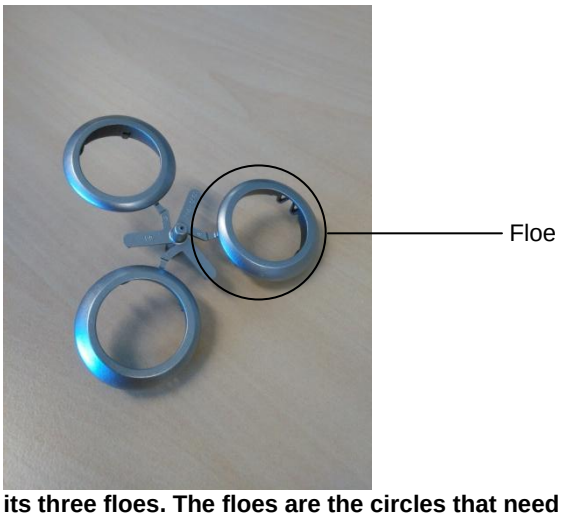


Figure 5-1 - Injected moulded part with its three floes. The floes are the circles that need to be broken off.

The floes are broken off through pulling it through the ‘breaking tool’, which is shown in Figure 5-2. Figure 5-3 shows how the three floes are in contact with the ‘breaking tool’ when the injected moulded part is pulled through the ‘breaking tool’. The robot that is chosen to perform this task in the UR5 (Universal Robot 5), shown in Figure 5-4. The UR5 was chosen because it has the teach mode and was available for this research.

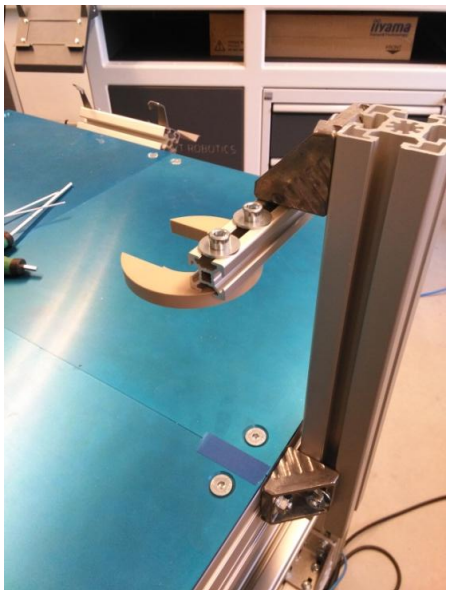


Figure 5-2 - 'Breaking tool'

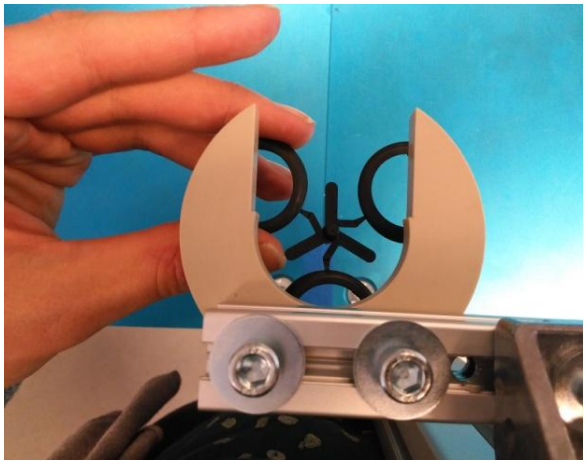


Figure 5-3 – Injected moulded part in place for breaking

Figure 5-4 - Universal Robots - UR5

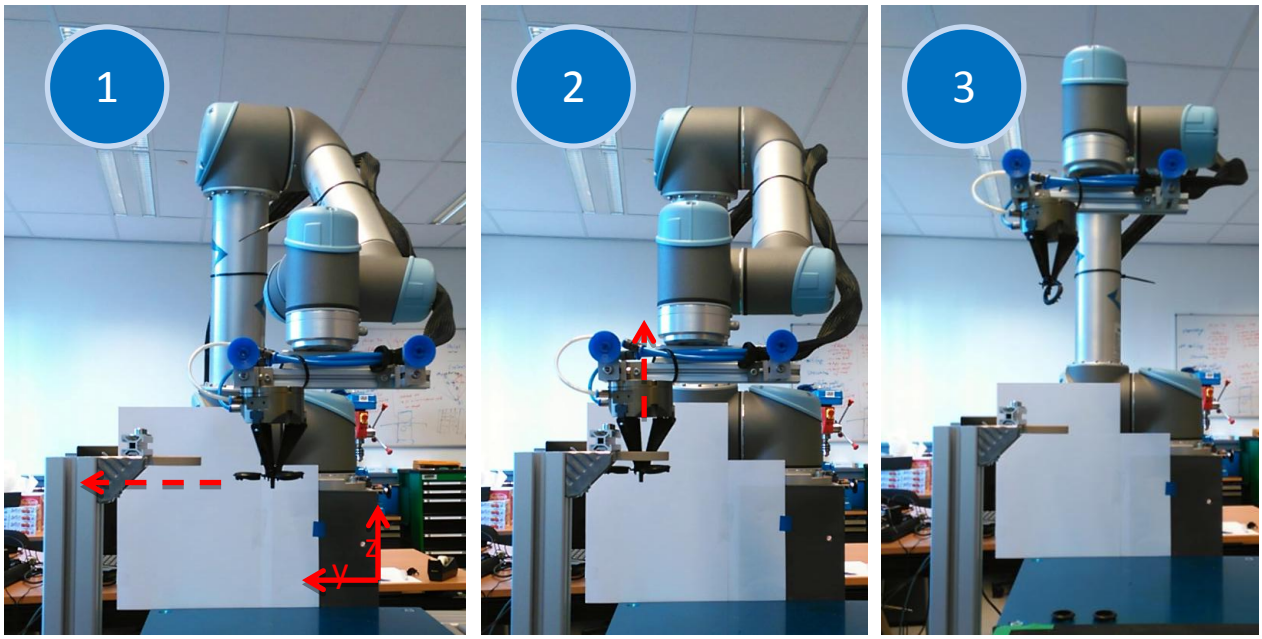


Figure 5-5 - Schematic drawing of the task

Figure 5-5 shows the three steps of the task. The first step for the robot is to bring the injected moulded part to the breaking tool. The second step is to accelerate in the upwards direction to break off the flows. Finally, in the last step, the task is concluded when the robot has stopped.

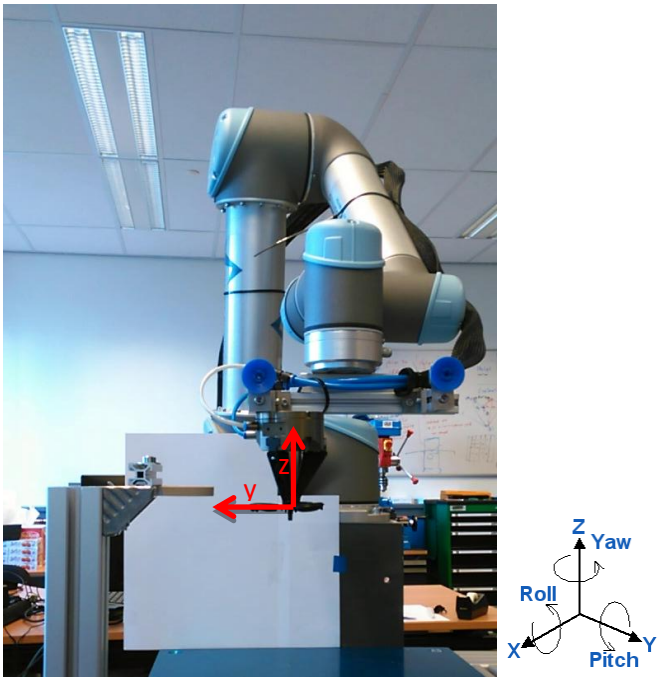


Figure 5-6 - Demonstration Set-Up

The gripper of the robot has in total six degrees of freedom, movement in the x-direction, y-direction, z-direction plus roll, pitch, and yaw.

For simplicity of this research is chosen to only control the y coordinate and the z coordinate, whilst keeping the other coordinates constant. As can be concluded from the task description those are the most important coordinates of the end effector of the robot.

5.2 Performance measure of the production task

The performance measurements for a generic production task is discussed in Section 2.1. The production task used in this research is to break off three floes of the injected moulded

part. Therefore, the first primary performance measurement is the amount of floes broken off. The performance measurements for this production task are summarized in Table 5-1.

Table 5-1 - The performance measurements can be divided into two categories: the primary measurements and the secondary measurements.

Primary measurements of performance	Secondary measurements of performance:
- Floes broken off - No collision	- Smallest execution time - Length of the path taken - Smooth movement

5.2.1 Performance measurement tools

This section describes how the performance measurements are calculated. It is important to make the performance measurements measurable to objectively compare different trajectories to each other. Per performance criteria, it is explained how it should be measured.

Primary measurements:
When the robot is performing the task, it is simple to define if all **floes are broken off** or how many floes are broken off. Unfortunately, when the algorithm is running it is not capable of checking whether the learned trajectory would break off the floes. Only when the trajectory is examined in real life, it would be possible to determine the amount of floes broken off.

What can be done is to analyse whether the floes should have been in contact with the breaking tool, but this does not necessarily mean that the floe is broken off. This can be done through analysing whether the floe has passed the coordinates of where the floes are in contact with the 'breaking tool'.

To summarise;

- Real life: count the amount of floes broken off. The answers possible are 0, 1, 2, or 3 floes are broken off.

Simulated: Do the floes have been in contact with the breaking tool. Does it pass the breaking point $\{y_{break}; z_{break}\}$. In other words, is there a timestep t for which the following statement is true? $(y(t) - y_{break}) + (z(t) - z_{break}) = 0$. Where $y(t)$ is the y coordinate at time step t and y_{break} is the y position where the floe is in contact with the breaking tool. For $z(t)$ and z_{break} the same definitions hold though for the z coordinate respectively. When there isn't a point where it passes this breaking point, the shortest distance, $p_{SimulationOfBrokenFloes}$, to this point is measured.

Collision: If there is a collision in real life between the robot and its environment, the emergency button needs to be pushed. Therefore, it is better to check before the robot is playing the trajectory if there is no collision with the breaking tool.

Figure 5-7 shows the limits which the robot should not pass. To avoid a collision the robot cannot move to low during the task, especially when close too the 'breaking tool'. In other words, the z coordinate should not be below z_{min}. For the y coordinate a similar definition holds, the robot should not move too much to the left so pass the y_{max} coordinate. So to check whether the movement doesn't have any collision the following statements need to be checked:

- Is there a time step t for which the following statement is true? $y(t) - y_{max}$>0 (When the answer is yes, there is a collision)
- Is there a time step t for which the following statement is true? $z(t) - z_{min} < 0$ (When the answer is yes, there is a collision)

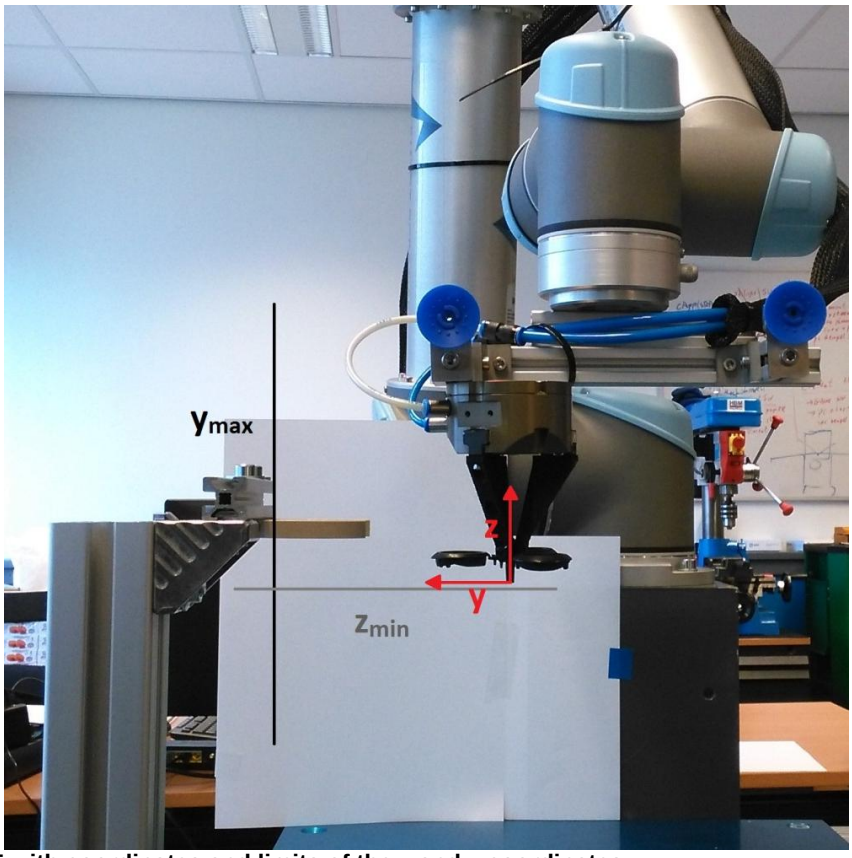


Figure 5-7 - UR5 with coordinates and limits of the y and z coordinates.

Secondary measurements

Smallest execution time: With DMPs the trajectory is parameterised with a fixed duration. Therefore, all demonstrations have in theory the same duration. To be able to evaluate the demonstration based on the difference in execution time another definition is used. The execution time is the time where the robot is actually moving. To determine this, the start time and the finish time of the movement are needed. The start time is where the robot starts its movement and finish time when the robot is in its final position. The start time t_s can be defined through Equation 5-1 and Equation 5-2. When the robot is not moving yet, the robot has no change in the y or z coordinate and no velocity. In Equation 5-2 can be seen that when the algorithm is one time step further than the start time t_s the robot is moving. Now either it has a change in the y or z coordinate or in the velocity.

$$(y(t_s) - y_0) + (z(t_s) - z_0) + \dot{y}(t_s) + \dot{z}(t_s) = 0\ . \quad 5\text{-}1$$

$$(y(t_s + 1) - y_0) + (z(t_s + 1) - z_0) + \dot{y}(t_s + 1) + \dot{z}(t_s + 1) \neq 0\ . \quad 5\text{-}2$$

Where $y(t)$ and $z(t)$ are the y and z coordinates at time step t. The initial coordinates of y and z are given through $y(0) = y_0$ and $z(0) = z_0$, and the velocities of coordinates y and z at time step t are $\dot{y}(t)$ and $\dot{z}(t)$ respectively.
For the finish time t_f, the time when the movement is finished, a similar description can be used, as is shown in Equation 5-3 and Equation 5-6.

$$\big(y(t_f) - y_{end}\big) + \big(z(t_f) - z_{end}\big) + \dot{y}\big(t_f\big) + \dot{z}\big(t_f\big) = 0. \quad 5\text{-}3$$

$$\big(y(t_f - 1) - y_{end}\big) + \big(z(t_f - 1) - z_{end}\big) + \dot{y}\big(t_f - 1\big) + \dot{z}\big(t_f - 1\big) \neq 0 \quad 5\text{-}4$$

Here is the final coordinate of y and z described as $y(t_{end}) = y_{end}$ and $z(t_{end}) = z_{end}$. Where t_{end} is the last time step recorded.

The execution time is then the finish time minus the starting time, as is shown in Equation 5-5:

$$p_{ExecutionTime} = (t_f - t_s) * dt. \quad 5\text{-}5$$

where dt is the step size in seconds.
With this definition a low value is seen as good performance, while a high value would be a bad performing demonstration.

Smooth movement: The movement is considered jerky when the system has very high accelerations. The jerkiness, or in this thesis referred to as inverse smoothness, is described as the absolute accelerations. Equation 5-6 shows the inverse smoothness at time step t:

$$p_{smoothness}(t) = |\ddot{y}(t)| + |\ddot{z}(t)| \quad 5\text{-}6$$

Equation 5-7 shows the definition of the inverse smoothness of the total trajectory:

$$p_{OverallSmoothness} = \sum_{t=t_0}^{t=t_{end}} p_{smoothness}(t) \quad 5\text{-}7$$

With this definition a low value is seen as good performance, while a high value would be a bad performing demonstration.

Length of the path taken: At each time step (t) the robot moves by its absolute velocity times the size of the time step. When summing this over the whole trajectory the length of the path taken is obtained, as shown in 5-8.

$$p_{PathTaken} = \sum_{t=t_0}^{t=t_{end}} (|\dot{y}(t) \cdot dt| + |\dot{z}(t) \cdot dt|) \quad 5\text{-}8$$

Another way of measuring the path would be to see if there are only positive velocities. A backwards motion is not a logical direction for the movement and, can, therefore be seen as a non-needed lengthening of the path. So when there is a negative velocity the absolute value of this velocity is used for calculating this unwanted movement. For the y coordinate this unwanted backward motion $p_{backwardsmotioniny}$ is defined as shown in Equation 5-9. For the z coordinate this unwanted backward motion $p_{backwardsmotioninz}$ is defined through Equation 5-10.

If $\dot{y}(t) < 0$
$p_{BackwardsMotionInY}(t) = -(\dot{y}(t))$
Else
$p_{BackwardsMotionInY}(t) = 0$ 5-9
End

$$p_{OverallBackwardsMotionInY}(t) = \sum_{t=t_0}^{t_{end}} p_{BackwardsMotionInY}(t)$$

If $\dot{z}(t) < 0$
$p_{BackwardsMotionInZ}(t) = -(\dot{z}(t))$
Else
$p_{BackwardsMotionInZ}(t) = 0$ 5-10
End
$p_{OverallBackwardsMotionInZ}(t) = \sum_{t=t_0}^{t_{end}} p_{BackwardsMotionInZ}(t)$

With those definitions, a low value is seen as good performance, while a high value would be a bad performing demonstration.

An overview of all the performance measurements which are previously discussed is given in Table 5-2. This table also overviews how they are measured and a quick description of the measurement is given.

Table 5-2 - Summary of Performance Measurements

Level of measurement	Which Measurement	How measured?	Description
Primary	Floes Broken	Real life	$0,1,2,3$ floes are broken off.
		Checked through formula.	Does it pass the point where the floes are in contact with the breaking tool? $p_{SimulationOfBrokenFloes}$ Calculates the distance to the breaking point. When the breaking point is passed, the distance to the breaking point is zero.
	Collision	Checked through formula.	Does the gripper of the robot come into contact with the breaking tool? Yes or No (No is wanted)
Secondary	Execution time	Calculated	$p_{ExecutionTime}$ (smaller is better)
	Smooth movement	Calculated	$p_{OverallSmoothness}$ (smaller is better)
	Path Taken	Calculated	$p_{PathTaken}$ (smaller is better)
		Calculated	$p_{OverallBackwardsMotionInY}$ and $p_{OverallBackwardsMotionInZ}$ (smaller is better)

Important though is that the above defined performance measurements correspond with what humans perceive as good performance, as is mentioned Section 2.2. This since for this research the demonstrations are ranked on how well they perform by a non-robotic expert.

6. Experimental set-up

In Section 4 the research method is discussed in general. This section describes the translation from a generic case to the specific situation for this research.

6.1 Gathering of demonstration data

As mentioned in Section 4.1.1 the gathering of the demonstration data can be divided into three parts; record, align, parameterise and replay. The following section discusses each step during the execution of the experiments.

Record

The kinaesthetic teaching can be done with the UR5 in the so called teach mode or freedrive mode (Universal-Robots, 2015). During the kinaesthetic teaching the Cartesian coordinates of the end effector are sampled with a fixed time-step through Robot Operating System (ROS). ROS is a set of open source software libraries and tools focussed on robotic applications (ROS, 2016). ROS is used for the connection between the robot and the computer hosting the experimental framework. The output of these recordings is given through a vector of the Cartesian coordinates ($\boldsymbol{y}$ and $\boldsymbol{z}$) and a corresponding time vector $\boldsymbol{t}$, shown in Equation 6-1. In those vectors, t_0 is the first time step with the corresponding $\boldsymbol{y}$ coordinate y_0 and z coordinate z_0. The last time step is given through t_e, y_e, and z_e.

$$\boldsymbol{t} = \begin{bmatrix} t_0 \\ t_1 \\ \vdots \\ t_e \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_e \end{bmatrix} \quad \boldsymbol{z} = \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_e \end{bmatrix}$$
$$\boldsymbol{x} = [\boldsymbol{y}\ \boldsymbol{z}] \qquad \text{6-1}$$

Aligning

It was decided to take as the alignment point the maximum velocity of the $\boldsymbol{z}$ coordinate since this is an important moment of the movement and can also easily be detected for each demonstration. Afterwards, a unique time frame, 15.5 seconds, was set where all demonstrations would fit in. See Figure 6-1 for the plots of the $\boldsymbol{y}$ and $\boldsymbol{z}$ coordinates of the trajectories of the demonstrations before and after the alignment.

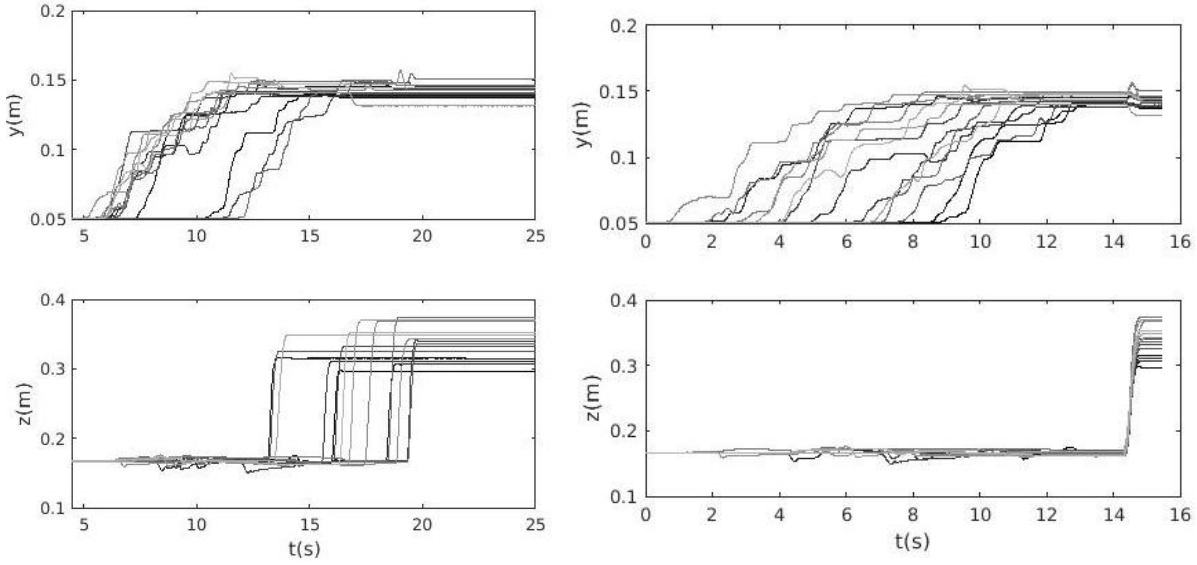


Figure 6-1 – The 14 recorded demonstrations before alignment are shown on the left and after alignment on the right. For the plots on the left before alignment, the demonstrations that were finished before t=25 s, were kept constant on the end state.

Parameterise the demonstrations

The parameters for the DMPs of the demonstrations are shown in Table 6-1. To mimic the demonstration the amount of base functions used to initialize the DMPs should be large. The effect of the amount of base functions is shown in Figure 6-2. A disadvantage of using a too large amount of base functions is that there are more policy parameters and therefore when using the DMPs for a learning task, this task takes longer. The DMPs also need a start state and goal state; the start state is the initial position of the robot and is the same for each demonstration. The goal state is the final position.

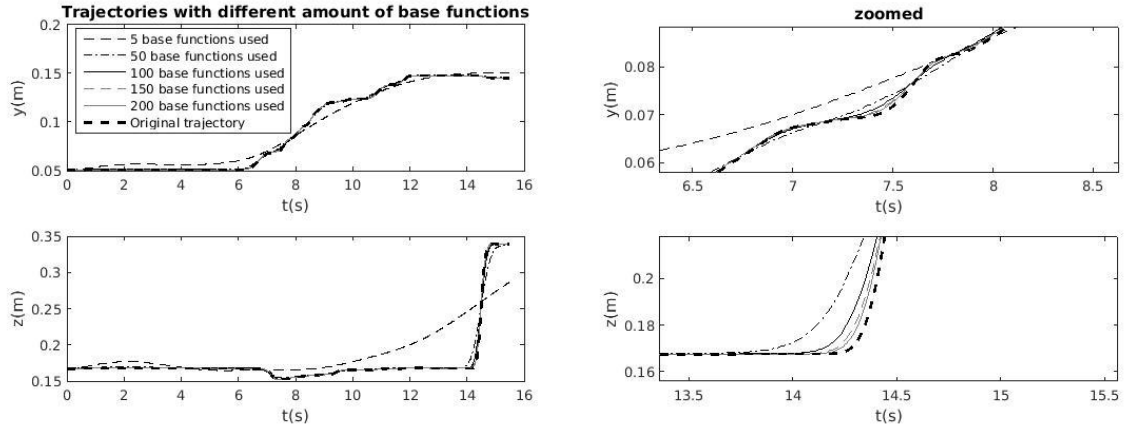


Figure 6-2 – Parameterised y and z coordinates of a demonstration with different amount of base functions, on the right the same plot is shown but zoomed in on a curve for better comparison.

To mimic the demonstration the amount of base functions should be large. With 150 base functions, the demonstrations are nicely imitated as is shown in Figure 6-2. The DMPs also need a start state and goal state; the start state is the initial position(y_0, z_0) of the robot and is the same for each demonstration. The goal state is different for each demonstration and is the final position (y_e, z_e) of the robot of that demonstration.

Table 6-1 - Parameters to parameterise the demonstrations through DMPs

Parameter	*Value*
Amount of base functions	150
Start state of y (first degree of freedom)	Starting state of the movement; which is $y_0 = 0.05$ metres.
Start state of z (second degree of freedom)	Starting state of the movement; which is $z_0 = 0.165$ metres
Goal state of y (first degree of freedom)	Depended on the final state of the specific demonstration, the value is in between $y_e = 0.13$ and 0.15 metres
Goal state of z (second degree of freedom)	Depended on the final state of the specific demonstration, the value is around $z_e = 0.3$ and 0.4 metres

Replay the DMPs of the demonstrations on the robot.

The goal is to obtain demonstrations that can be ranked by non-robotic experts. Playing back the DMPs of the demonstrations on the robot results in clearer and simpler demonstrations. This enables effective ranking by non-robotic experts. As mentioned in Section 4.1.1 the DMPs need to be fed back to the robot, because of the approximation of the trajectories through the parameterisation. Another important reason to feed back the DMPs of the demonstrations to the robot is that during the kinaesthetic teaching all degrees of freedom are manipulated through the demonstrator, while for the simplicity only the y and z coordinates of the end effector of the robot are chosen to control. This also simplifies the

movement significantly. The DMPs are calculated through the MATLAB code of Ijspeert et al. (2002a). Furthermore, it is necessary to count the amount of floes these trajectories are breaking off. For replaying the parameterized demonstrations the MATLAB driver from Aalamifar (2015) was used to make the connection with the UR5 and send the Cartesian coordinates.

6.1.1 Parameters for creating weight factors through the preferences

As mentioned in Section 4.1.2 the performance measurements are used to describe an overall performance measurement P. The performance measurements used for this experiment are described in Section 5.2. Since with all the gathered demonstrations no collision did occur, this performance measurement is left out. Also is decided to split the performance measurement smoothness into 5 time steps and the overall smoothness of the whole trajectory. To describe the overall performance measurement two cases are distinguished:

1. With the amount of floes broken off taken into account as one of the performance measurements, shown in Equation 6-2
2. Without the amount of floes broken off taken into account as one of the performance measurements, shown in Equation 6-3.

For this split is chosen, since for the learned trajectories the performance measurement, the amount of floes broken off, is hard to measure offline, and it is desirable to have a measurement for the learned trajectories without the need of feeding them back to the robot.

The performance of the learned trajectories can only be evaluated with the overall performance measurement P_2, unless the learned trajectory is fed back to the robot to measure the amount of floes broken off. Since a new learned trajectory can perform worse on a single performance measurement, such as execution time, then the input demonstrations it can receive a negative value for this measurement. This can lead to that the learned trajectory has a really low overall performance, whilst is only bad for one of the performance measurements. Therefore, it is chosen to when the single performance measurement has a negative value to bind this one to zero.

$$P_1 = v_1 \begin{bmatrix} p_{ActualAmountOfFloesBroken} \\ p_{SimulationOfBrokenFloes} \\ p_{ExecutionTime} \\ p_{smoothness1} \\ p_{smoothness2} \\ p_{smoothness3} \\ p_{smoothness4} \\ p_{smoothness5} \\ p_{OverallSmoothness} \\ p_{PathTaken} \\ p_{NoBackwardsMotionInY} \\ p_{NoBackwardsMotionInZ} \end{bmatrix} \quad \text{6-2}$$

$$P_2 = v_2 \begin{bmatrix} p_{SimulationOfBrokenFloes} \\ p_{ExecutionTime} \\ p_{smoothness1} \\ p_{smoothness2} \\ p_{smoothness3} \\ p_{smoothness4} \\ p_{smoothness5} \\ p_{OverallSmoothness} \\ p_{PathTaken} \\ p_{NoBackwardsMotionInY} \\ p_{NoBackwardsMotionInZ} \end{bmatrix} \quad \text{6-3}$$

6.2 Parameter selection of the learning algorithms

This section is split into the parameters for inverse PI^2 and PI^2.

6.2.1 Parameters chosen for inverse PI^2

For implementing the inverse PI^2 algorithm the weights $\boldsymbol{w}$ of Eqaution 3-23 are calculated through the *fmincon* function of MATLAB *(MathWorks, 2016)*. The *fmincon* function finds the value for the variable that minimises the nonlinear multivariable function that is given.

Cost features for inverse PI^2

As mentioned in Section 4.2.1 the cost features exist out of features based on the performance measurements and 'steering' features. In Table 1 an overview of all cost features is given, with from where they originate. The performance measurements can be found in Section 5.2.1.

The performance measurement smoothness is split into five timesteps, since it can differ per timestep whether the accelerations should be low. Furthermore, it was chosen to split the performance measurement execution time into eight features. First it was split into the start time and the end time, which were respectively split into the y and z coordinate afterwards. The feature to keep the y coordinate constant and therefore let the robot start as late as possible with the movement (start time), was also split into five time steps. The exact divisions can be found in Appendix A.

The steering features are determined through the movement, it needs to follow. The movement consists of three parts, as was explained in Section 5.1. First, the robot needs to bring the floes into place by moving along the y direction. Afterwards, the robot needs to accelerate quickly in z direction to obtain enough velocity for when the floes hit the breaking tool and break off the floes. Afterwards, the movement is ended.

There are two features that were formulated to steer the movement. The first feature was to bring the floe into place by changing the y coordinate close to the breaking tool, this feature is given in Equation 6-4. The second feature was to make sure the end effector was in place before accelerating in the z direction to break off the flow; this feature is given in Equation 6-5.

$$\phi_{bringfloesinplace} = \phi_y = |y - y_g| \tag{6-4}$$

$$\phi_{beforebreaking} = |y - y_{bb}| + |z - z_{bb}| \text{ for } t = [12{:}\,13] \tag{6-5}$$

The complete set of features can be found in Appendix A.

Exploration noise

The exploration noise ϵ has typically a zero mean, chosen for the variance (Σ) of the noise to use a value of 0.2. Furthermore, it is chosen to create ten noisy versions of the demonstrated trajectory, $K = 10$.

6.2.2 Parameters chosen for PI^2

As mentioned in Section 4.2.2 for the experiments a choice needs to be made on the amount of iterations and the exploration noise. After trying some different values for those parameters, the following ones were chosen because they created a search that nicely converged, for this task.

- In total, there were 300 iterations.
- The exploration noise is kept small; respectively 0.2 and it even further converge to zero as more iterations had been done. In total, at each iteration, 6 new trajectories were created and 6 were re-used from the previous iteration.

7. Results

In this section, the results of the experiments are shown. First, the input demonstrations are shown. Afterwards, the results of the crowdsourcing experiment are shown in Section 7.1. In Section 7.2 the input demonstrations are shown which relate to hypothesis 1. Followed by Section 7.3 where the input demonstrations and the preferences of the non-robotic expert are used to evaluate hypothesis 2. Finally, in Section 7.4 an interesting selection of the learned trajectories is fed back to the robot to see how the trajectories perform in real life.

For the experiments in total 14 demonstrations are recorded. The amount of demonstrations created is chosen since it is desired to have demonstrations with different amount of floes broken off and to have variation in terms of smoothness, execution time among others. The performance of the 14 recorded demonstrations, according to the performance measurements, can be found in Appendix B - Table 1. Although the demonstrations show differences in the performance measurements, it can be argued that the demonstrations are too similar when looking at **Figure** 6-1 at the right plot. In this figure for all 14 demonstrations the y and z coordinates over time are shown. What is seen that the demonstrations mainly differ in the y coordinate and more specifically when does the robot start to change this y coordinate. Subsequently, this led to a difference in the 'waiting' time in between that the y coordinate is in place and the z coordinate is changing.

7.1 Crowdsourcing experiment

To rank the recorded demonstrations the crowdsourcing experiment described in Section 4.1.2 is set up to gather the non-robotic expert's preferences. The results of this crowdsourcing experiment are shown in this section. To obtain a sequential order by the non-expert's preferences all 14 demonstrations need to be pairwise compared. As a result, in total 182 combinations needed to be ranked. In total 78 people participated in the crowdsourcing experiment. Each participant compared seven different combinations of the demonstrations, in total there were 546 preferences, which means that for each combination around 3 to 4 preferences were given.

The average participant is an 30.5 year old male, with a technical background. He has no to little experience in Robotics and the UR5, has worked long with Artificial Intelligence, although this is quite contradictory to the study direction. He does not have any experience with preference learning or Inverse Reinforcement learning. Reinforcement Learning is mostly familiar, but sometimes from the different field such as education. The demographic data and the link to the raw data can be found in Appendix F.

The answers that the participants gave are shown in Section 7.1.1, besides those preferences some of the participants mentioned in the comment section that they found it hard to determine on which criteria they should evaluate the performance. Since the participants said they could not always find the difference between the two demonstrations. In the next section more details are given on the answers the participants gave and the correlations with the performance measurements.

7.1.1 Correlations of the performance measurements

With the survey in total 546 preferences were given. The answer options were a>b a<b a=b or a?b, where the first two options are considered as clear preference and the second two as no clear preference. In Appendix B - Table 2 all given answers are shown. In Figure 7-1 is shown how many of the preferences were clear preferences, also a division is made on whether the combination of demonstrations rated had a difference in the amount of floes broken off. What can be seen is that the participants when there was a difference in the amount of floes broken off people found it much easier to give a clear preference, than when the same amount of floes were broken off.

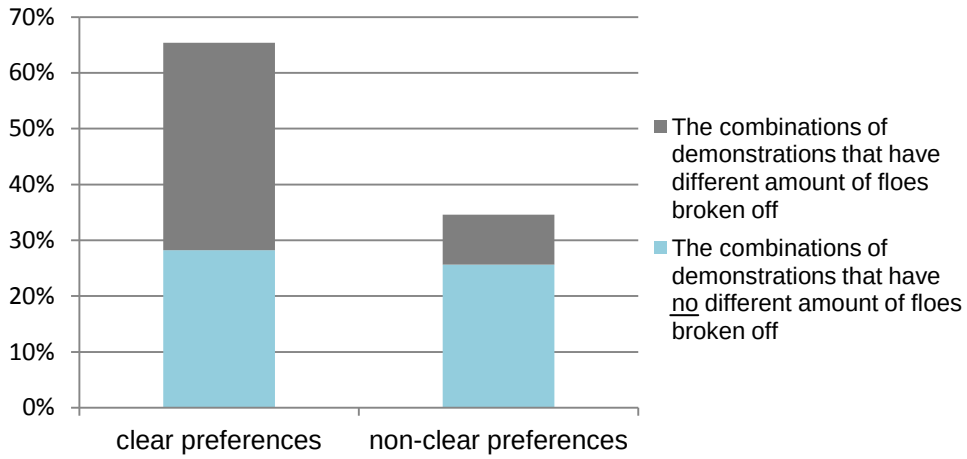


Figure 7-1 - All pairwise given preferences divided over clear preferences and non clear preferences. What can be seen is that when there is a difference in the amount of floes broken off between the two demonstrations shown, more clear preferences are given. In total of all preferences 65.4% are clear preferences.

In Table 7-1, all correlation values ($c(p)$) of the performance measurements with or without the difference in amount of broken floes can be found. A quick recap of the performance measurements can be found in Table 5-2. For the smoothness is chosen to split it into 5 time steps. Additionally, also the overall smoothness of the whole trajectory is shown. Three interesting results can be highlighted from Table 7-1. First is that a high correlation is found for the amount of floes broken off. Secondly, when only looking at the combinations where the same amounts of floes are broken off, the other performance measurements have higher correlations with the preferences of the non-robotic expert. For example, the execution time and the second time step of the smoothness have really high correlations. Finally, also negative correlations are found. Those negative correlations are found when there is was a different amount of floes broken off for the smoothness in the second and third time step, and for the path taken. Furthermore, negative correlations are found at the simulated break when only having the same amount of floes broken off. Those negative correlations can be explained through that all the performance measurements influence each other, and therefore if another performance measurement is optimal another one is less optimal. More explanation about this phenomenon can be found in Section 8.1.

Table 7-1 – The correlation values ($c(p)$) of the different performance measurements. The correlations higher than 0.2 or lower than -0.2 are highlighted. When looking at all the preferences the amount of floes is shown to have the highest correlations with the preferences given in the crowdsourcing experiment. When only looking at the combinations of demonstrations that have no difference in the amount of floes broken it can be seen that the other performance measurements are higher correlated.

	Amount of clear preferences	*Amount of floes broken off*	*Simulated break: Shortest distance to breaking point*	*Execution time*	*Smoothness Split into 5 time step and the overall smoothness*	*Path taken*	*No backward motion y*	*No backward motion z*
All preferences	357	**0.6256**	-0.0735	0.1877	0.0532 0.0588 -0.0756 -0.1429 0.1485 0.1541	0.0084	**0.3053**	0.1036
Different amount of floes broken off	203	**0.6256**	0.1214	-0.0936	-0.0936 **-0.2906** **-0.2217** 0.0739 **0.2315** -0.0640	**-0.2117**	**0.2512**	-0.1626
Same amount of floes broken off	154	N/A	**-0.3143**	**0.5584**	**0.2468** **0.5195** 0.1169 **-0.4286** 0.0390 **0.4416**	**0.3117**	**0.3766**	**0.4545**

Interesting would be to see what the important performance measurements are when the different amount of floes are broken off, but the preference is not given according to this performance measurement. For example, when demonstrations A breaks of 2 floes and demonstrations B 1 floe, but demonstrations B is preferred, it is interesting to see which of the other performance measurements is then chosen to rank the demonstrations on. The most interesting result with this division is shown in Table 7-2, the full table is shown in Appendix B - **Table** 3. The most important result is that the simulated break, through using the shortest distance to the breaking point is correlated with the amount of floes broken off. The other results found in Appendix B - **Table** 3 were as expected, namely when the preference did not agree with the amount floes broken off the other performance measurements became more important. Similar to when only looking at the preferences at the combinations which have the same amount of floes broken off.

Table 7-2 – the correlation values ($c(p)$) of the performance measurement, shortest distance to breaking point, which simulates the break of the floes divided in two categories is shown. The first category is when the preferences do not agree with the performance measurement the amount of floes broken off. The second category they do agree. It can be seen that they correspond to the performance measurement amount of floes broken off.

	Amount of preferences	*Amount of floes broken off*	*Shortest distance to breaking point 1.0e-04 m*
*Preference does **not** agree with the performance measurement the amount of floes broken off*	38	-1	-0.6970
Preference does agree with the performance measurement the amount of floes broken off	165	1	0.3143

7.1.2 Combining the performance measurements to obtain the overall performance measurement description

After the correlation of the performance measurements are known the next step is to create the overall performance measurement (P_1, P_2), shown in Equation 6-2 and 6-3. The optimization search of Equation 4-3 for the weights v_1 and v_2 led to the result shown in Table 7-3.

Table 7-3 – The learned weights v_1 and v_2 to create the overall performance measurements P_1 and P_2. For v_1 a high weight is given to the amount of floes broken, where v_2 has the highest weights more divided over the features; no backward movement in y, smoothness in the second and third time step, and the simulated break.

	Amount of floes broken off	*Simulated break*	*Execution time*	*Smoothness during t=[0 3]*	*Smoothness during t=[3 6]*	*Smoothness during t=[6 9]*	*Smoothness during t=[9 12]*	*Smoothness during t=[12 15.5]*	*Overall smoothness*	*Path taken*	*No backwards movement in y*	*No backwards movement in z*
v_1	**0.6667**	0.0201	0.0653	0.0451	0.0557	0.0266	0.0007	0.0149	0.0392	0.0019	0.0425	0.0214
v_2	N/A	**0.1283**	0.0902	0.0861	0.0590	**0.1873**	**0.1413**	0.0216	0.0173	0.0028	**0.1901**	0.0761

Which led to that the demonstrations could be ranked through P_1 and P_2, shown in Figure 7-2. P_1 has a correlation value $c(P_1) = 0.6078$ and P_2 has a correlation value of $c(P_2) = 0.3501$. In Table 7-3 can be seen that according to P_1 the floes broken off is the most important measurement for the overall performance measurement. For P_2, the weights v_2 are more spread over the simulated break, the second and third time step of the smoothness and no backwards movement in y. This leads to that for P_1 demonstration 10 is performing best, while for P_2 demonstration 2 is performing best. As mentioned in Section 6.1.1 P_2 is used to evaluate the learned trajectories when they are not fed back to the robot.

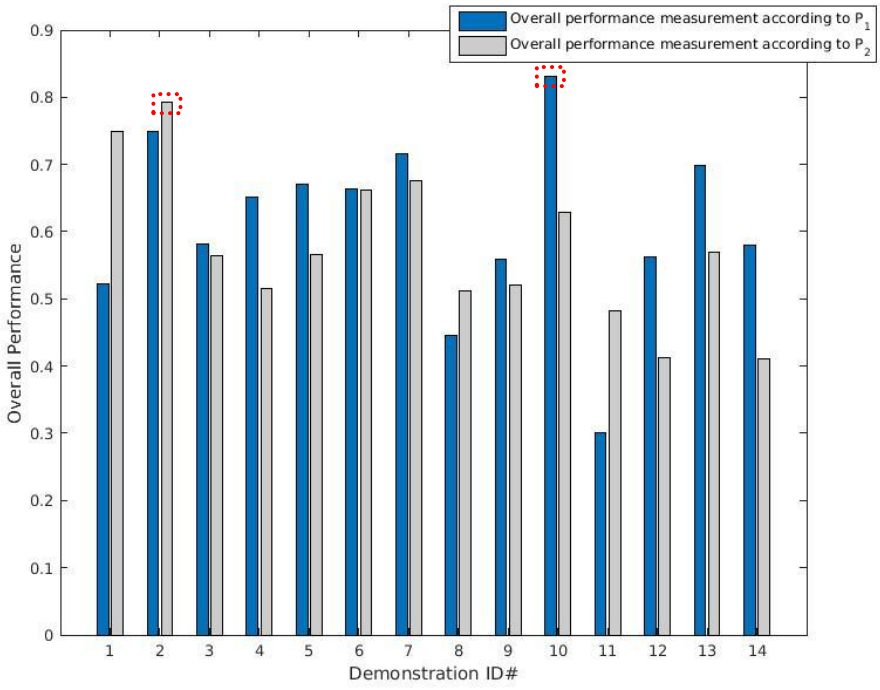


Figure 7-2 - The ranking of the input Demonstrations based on Performance measurement with floes broken (P_1) and without floes broken off P_2. All exact values and the full order can be found in Appendix B - Table 4. It can be seen that according to P_1 demonstrations 10 is performing the best, while according to P_2 demonstration 2 is performing best.

The mean trajectory has according to P_2 a performance of 0.8180, which is higher than all the demonstrations separately. This was being expected since it averages out some of the small bumps of the demonstrations, so a higher smoothness is found and less backward motion. As said the smoothness and less backward motion in y are having high weights v_2 for P_2.

7.2 Hypothesis 1: One or multiple demonstrations

This section contains the results for evaluating hypothesis 1. The hypothesis is that the performance of the learned trajectories is mainly based on the average performance of the input demonstrations. More specifically, is looked whether using multiple input demonstrations does change this.

For Figure 7-3 is decided to use the best demonstrations, according to the overall performance measurement P_2, for plotting. This means that when 1 demonstration is selected the trajectory is learned from only demonstration 2, and when 2 demonstrations are selected it learns from demonstration 2 and 1. Figure 7-4 shows a similar plot only now the worst demonstrations are selected. An overview of which input demonstrations are used for the figures can be found in Table 7-4.

Table 7-4 - An overview of which demonstrations ID are used for Figure 7-3, Figure 7-4, and Figure 7-5.

#amount of demonstrations	*Case 1 (ID#)*	*Case 2 (ID#)*	*Case 3 (ID#)*	*Case 4 (ID#)*	*Case 5 (ID#)*
Figure 7-3	2	2, 1	2, 1, 7	2, 1, 7, 6	2, 1, 7, 6, 10
Figure 7-4 / Figure 7-5	14	12, 14	11, 12, 14	8, 11, 12, 14	4, 8, 11, 12, 14

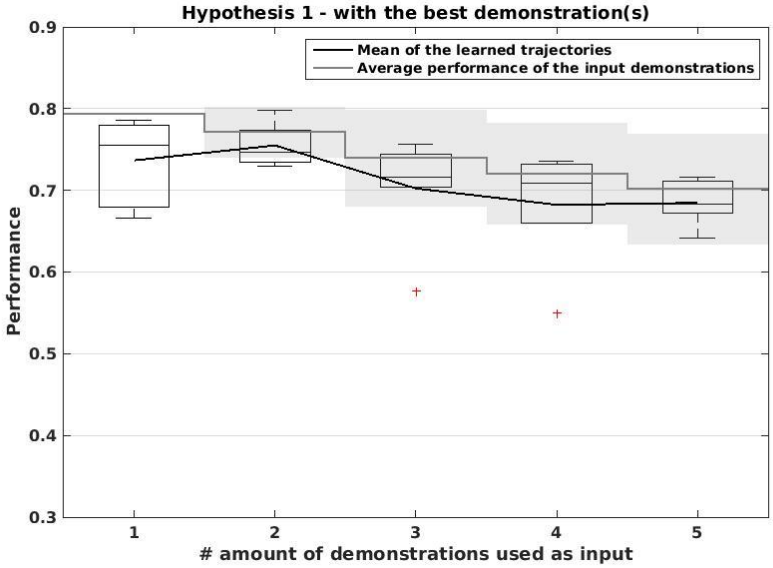


Figure 7-3 - The performance of the learned trajectories with 1,2,3,4 or 5 demonstrations used as input for IRL. The demonstrations are selected to be the best demonstrations according to P_2. Also the average performance of the input demonstrations is plotted. The grey area around the mean represents the standard deviation of the performance of the input demonstrations. The boxplots are represented the performance of 6 learned trajectories, the outliars are given as red crosses. It can be seen when 2 demonstrations are used the performance is increased a bit afterwards the performance is degrading in the same trend as the average performance of the input demonstrations is.

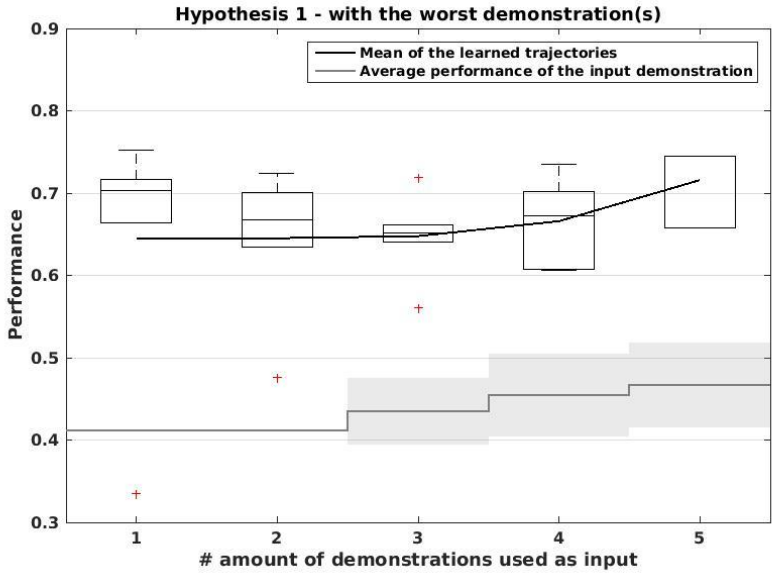


Figure 7-4 - The performance of the learned trajectories with 1,2,3,4 or 5 demonstrations used as input for IRL. The demonstrations are selected to be the worst demonstrations according to P_2. Also the average performance of the input demonstrations is plotted. The grey area around the mean represents the standard deviation of the performance of the input demonstrations. The boxplots are represented the performance of 6 learned trajectories, the outliars are given as red crosses. What is seen is that when the average performance of the input demonstrations is increasing, so is the performance of the learned trajectories. Further the learned trajectories are a performing lot better than the input demonstrations.

As expected, Figure 7-3 shows that the performances of the learned trajectories seem to follow the average performance of the input. When the average performance of the input demonstrations decreases also the performance of the learned trajectories decreases. Another interesting result from Figure 7-3 is that a little increase in the performance is seen when going from one input demonstrations to two demonstrations. This increase is not seen in Figure 7-4. As expected Figure 7-4 shows that when the performance of the input demonstrations increases also the performance of the learned trajectories increases. A remarkable result from Figure 7-4 is that the performance is higher than the average input demonstrations. This is probably due to the initialisation with the mean trajectory of all demonstrations, as explained in Section 4.2.2. Another run with RL is made with a different initialisation DMP, namely with the worst demonstration respectively demonstrations with ID 14, according to performance measurement 2, as initialisation. In Figure 7-5 this is shown and it can be seen that the performance of the learned trajectories is now closer to the average of the input demonstration with this different initialisation of the DMPs.

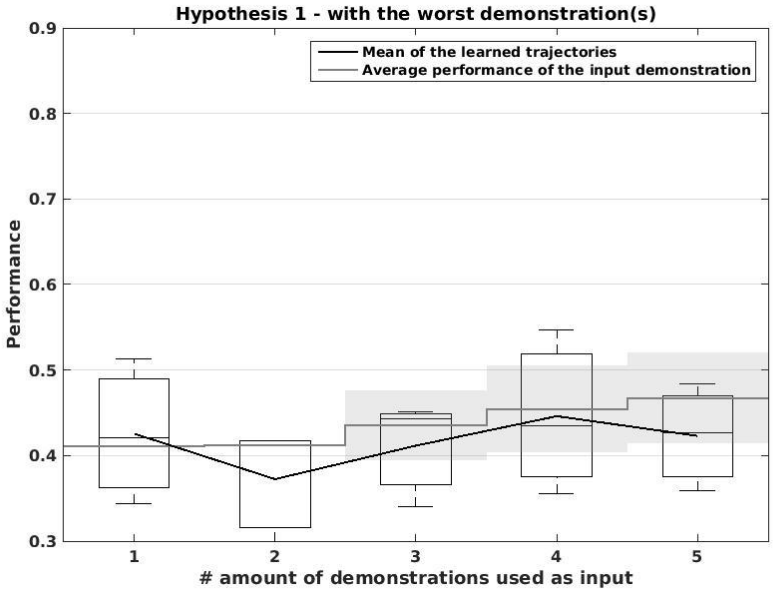


Figure 7-5 - Similar to Figure 7-4, but here for the initialization of the DMPs for RL demonstration 14 is used, which is the worst performing demonstrations according to P_2. Furthermore, the boxplots are now represented the performance of 3 learned trajectories. The performance of the learned trajectory is now closer to the average performance of the input demonstrations, than in Figure 7-4.

7.3 Hypothesis 2: Added preference

This section contains the results for Hypothesis 2, which is about the effect of adding preferences through weight factors of the preferences of the non-robotic expert as defined in 4.1.2.

Figure 7-6 shows for each of the four sets of selected input demonstrations, the three cases. The first case is without any added preferences; in this case, the weights are uniformly divided over the trajectories. The second case is learned with added preferences through weight factor 1. Finally, the third case the trajectories are learned by adding the preferences through weight factor 2. An overview of the weight factors is shown in Table 7-5

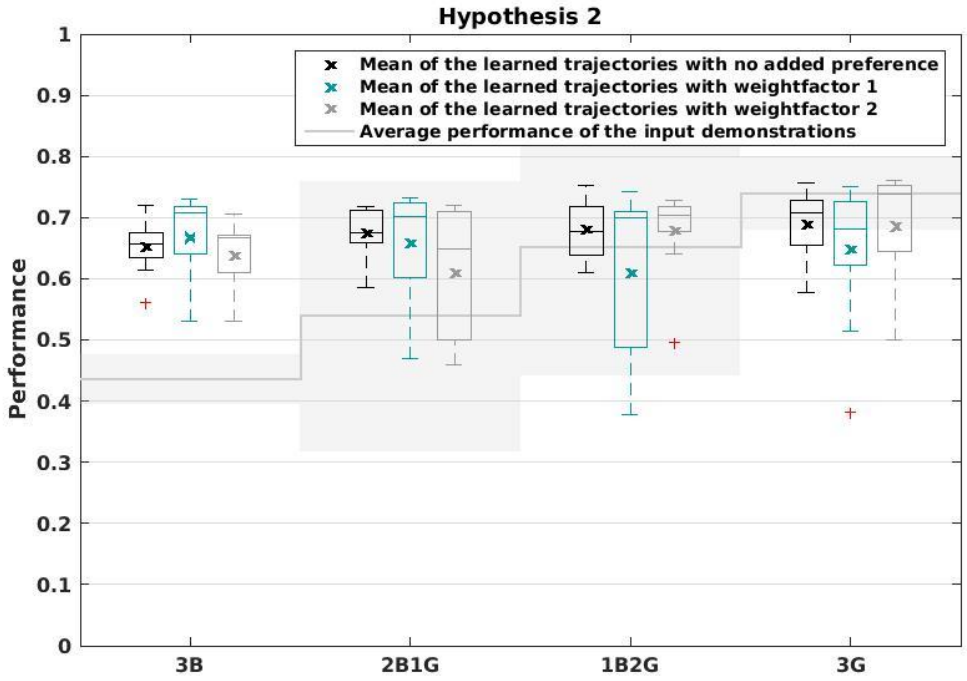


Figure 7-6 - Performance plotted of the 3 cases; without preferences, with preferences through weight factor 1 and with preferences added through weight factor 2. 3B means that 3 'bad' demonstrations are used as input, 2B1G means that 2 'bad' and 1 'good' demonstrations are used, 1B2G means that 1 'bad' and 2 'good' demonstrations are used, and 3G means that 3 'good' demonstrations are used. The above categorization is made through P_2. The boxplots are represented the performance of 9 learned trajectories, the outliars are given as red crosses. No clear difference in performance between the different cases for each of the four sets of input demonstrations.

Table 7-5 - The weight factors used for learning the trajectories from which the performance is plotted in Figure 7-6.

Situation:	*3B*	*2B1G*	*1B2G*	*3G*
Demonstrations ID #	*11 12 14*	*2 12 14*	*2 1 14*	*2 1 7*
No added weights	[0.33 0.33 0.33]	[0.33 0.33 0.33]	[0.33 0.33 0.33]	[0.33 0.33 0.33]
Weight Factor 1, (created through P_2)	[0.3692 0.3158 0.3150]	[0.4906,0.2550,0.2543]	[0.4061 0.3834 0.2105]	[0.3576 0.3376 0.3048]
Weight Factor 2	[0.5000 0.3333 0.1667]	[0.5000 0.3333 0.1667]	[0.5000 0.3333 0.1667]	[0.5000 0.3333 0.1667]

In all situations, it seems that the learned performance is very similar. Since earlier was seen that the performance was also highly influenced through the initialisation of the DMPs, decided was to replay 2 situations with 3 data points to see whether this would change the outcome of the learned trajectories with respect to each other. It was chosen to select random three learned cost function in the case no added preferences and for three learned trajectories with added preferences through weight factor 1 with as input '2B1G'. Afterwards was optimized to those cost function through PI^2 using two different initialisations of the DMPs for learning the trajectories. In the first case the mean trajectory of all demonstrations was used before as well is shown in Figure 7-7. The second case the worst demonstrations, according to P_2, were used, respectively demonstration 14. Although it can be seen that the performance of the case with initialisation through the worst demonstration is lower than shown in with the initialisation with the mean trajectory. There seems to be no difference in learning between the cases with or without preference. The performance is just lowered in both cases evenly.

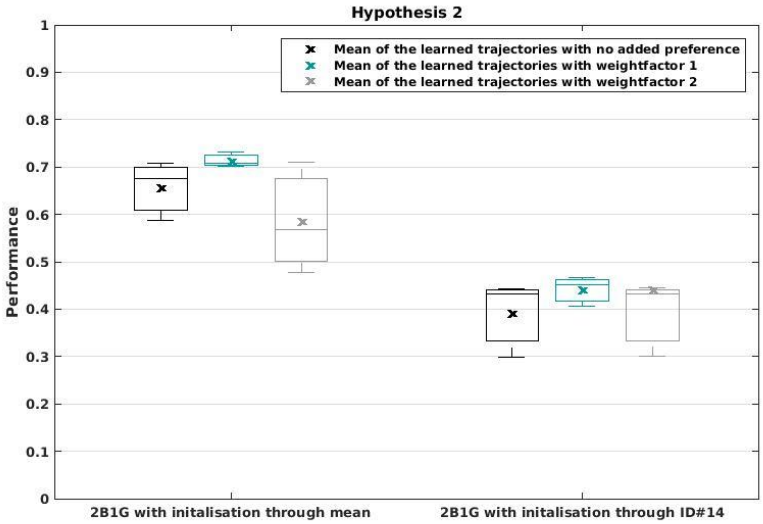


Figure 7-7 - The performance of the learned trajectories with the same learned cost function using a different initialization of the DMPs. The boxplots are represented the performance of 3 learned trajectories. The performance with the initialization through demonstrations 14 is just lowered in comparison to the performance of the learned trajectories which were using the initialization through the mean.

7.4 Performance on UR5

Finally, to see whether the learned trajectories are usable in the real life scenario, 15 of all the learned trajectories are selected to feed back to the UR5. The main observations are:

- When the robot follows the same learned trajectories twice, the amount of floes broken off is not necessarily the same. Therefore the performance according to P_1 is not repeatable.
- There are two batches of floes, which have a different coating. With the uncoated 'silver' batch the floes are much easier to break off compared to the coated 'black' batch.
- The speed at the breaking point is also an important factor for breaking off all floes. Sometimes the floes are touched and therefore bend, but not broken.
- The positioning of the breaking tool should be very precise. When the breaking tool is placed a few millimetres differently the floes were not broken off anymore.

Appendix D explains which trajectories are chosen and shows the amount of floes that the selected trajectory broke off. Since the amount of floes broken off depends on more factors than just the trajectory, it is hard to say something about the performance according to P_1. Therefore hypotheses 1 and 2 cannot be evaluated according to this overall performance measurement P_1. The conclusions made, with the overall performance measurement P_2, are still valid. This is because the input demonstrations are also evaluated according to P_2, without the amount of floes broken off.

8. Discussion and conclusion

This section starts with a recap of the research goal afterwards the results shown in Section 7 are discussed. The main goal of this thesis is to find the effect of the input demonstrations on the performance of the learned trajectories. Therefore, it is important to first determine what good performance is. This is done through the preferences of a non-robotic expert which are gathered through a crowdsourcing experiment. The results of this crowdsourcing experiment are discussed in Section 8.1. Afterwards, in Section 8.2 the effect of the amount input demonstrations used is discussed. According to Hypothesis 1 the average performance of the input demonstrations depends on the performance of the learned trajectories, the amount of input demonstrations has less influence. In Section 8.3, Hypothesis 2 is evaluated, namely, whether adding the preference knowledge of the non-robotic expert to the demonstrations is indeed improving performance of the learned trajectories. In Section 8.4 the conclusion on the selected learned trajectories which were fed back to the robot are given. Afterwards, in Section 8.5 the overall conclusion is given. Finally, Section 8.6 contains the recommendations for future work are given.

8.1 Crowdsourcing experiment

As expected the most important factor for defining good performance is the amount of floes broken off, as is shown in Figure 7-1 and Table 7-1. In other words, whether the robot is performing the given task as desired. In Figure 7-1 shows that the participants found it difficult to evaluate the trajectories, since only 65.4% of the preferences are clear preferences, which means that one of the demonstrations is ranked as better performing. This is in line with the comments of the participants, which stated the participants mentioned in the comment section they found it hard to find the differences between the demonstrations. When the participants need to rank the combinations of the demonstrations that have a difference in the amount of floes broken off, clearly more clear preferences are given.

In Table 7-1 shows that the amount of floes broken off has the highest correlation value with the preferences of the non-robotic expert. Furthermore is seen in this table that when the amount of floes broken off is not taken into account the other measurements become more important. An interesting result to mention shown in Table 7-1 is that the second time step of the smoothness is very highly correlated. A theory could be that this measurement is correlated with the execution time. When the execution time is small, the robot is not moving in the first and second time step, so likewise the accelerations are low. That the first time step of the smoothness is less correlated can be explained through that for that time step most of the demonstrations are not moving yet and therefore they all receive a high performance value for that performance measurement.

As can be seen in Table 7-2 the performance measurement simulated break seems to correlate with the amount of floes broken off as expected. Maybe an even higher correlation can be reached when adding the speed at that time since this is also an important factor for breaking as found during feeding back the selected learned trajectories on the UR5.

Overall performance function

As is shown in Figure 7-2 two overall performance measurements were created. One is with the amount of floes broken off taken into account P_1, the other one is without P_2. The one with the amount of floes broken off taken into account had a much higher correlation. Unfortunately, the amount of floes broken off is hard to determine without actually replaying the trajectory on the robot. Furthermore, it is seen in Section 7.4 when replaying the amount of floes broken off can differ within a learned trajectory. Therefore is chosen to use the overall performance measurement P_2, that does not take the amount of floes broken off into account for analysing the results.

In Section 7.1.2 is mentioned that the mean of all demonstrations is having a high overall performance P_2. This can be due to that the mean of all demonstrations is having low accelerations and therefore a high smoothness. Furthermore, the mean trajectory is having almost no backwards velocities in the y and z direction.

It can be questioned whether smoothness is not getting a too high influence on the overall performance P_2 since it has multiple weigths v in the overall performance measurement as is seen in Table 7-3. While for the execution time there is only one weight v. This can also be a reason why the mean is scoring higher than expected.

8.2 Hypothesis 1

As expected is shown in Figure 7-3, Figure 7-4, and Figure 7-5 that the performance of the learned trajectories depends on the performance of the input demonstrations. Simply adding more demonstrations with the idea of adding more information, is therefore not beneficial. Important is that the demonstrations that are added are of high quality, to improve the performance of the learned trajectory. Furthermore, the quality of the trajectory that is used as initialisation of the DMPs for RL is having a big influence on the performance as can be seen in Figure 7-5.

In theory, it should be hard to learn from one demonstration, since there is a high chance of optimising to a bad behaviour of this single demonstration. When using more than one demonstration this is averaged out. Although this effect can be seen in Figure 7-3, it is a smaller effect than expected. Furthermore, Figure 7-4 does not confirm this theory and therefore could not be said that it learns better from one demonstration than from two demonstrations which have a lower average performance.

8.3 Hypothesis 2

As mentioned in the previous section, the average performance of the input demonstrations is determining the performance of the learned trajectory. With hypothesis 2 is looked whether the performance can be increased when adding the non-robotic expert's preferences in the learning process. In theory the good demonstrations would be used more in this case than the bad demonstrations which should lead to a better performance than when assuming that all the demonstrations are equally important. Through this ranking the goal is to be able to also use imperfect demonstrations, which also contain a lot of information on the task that needs to be performed but have a low performance.

Unfortunately, this hypothesis is not confirmed as can be seen in Figure 7-6. For the three different cases; without added preferences, directly add the information of P_2, or through only giving the order of the demonstrations the learned trajectories perform similarly. Also looking at different combinations of input demonstrations did not change the performance.

For the situations where three similar performing demonstrations were used. This is logical since when the demonstrations are quite similar in performance the weigh factors are also very close to each other, as can be seen in Table 7-5. Also with the third case with weight factors 2, there are not that many differences since the demonstrations are close to each other in terms of performance and probably therefore also in their shape.

For the situations where there is a high spread in the performance of the input demonstrations an increase of the performance when using the added preferences is expected. Unfortunately, also no differences in performance are found when learning from 1 good and 2 bad demonstrations or from 1 bad and 2 good demonstrations when adding preferences. This can be caused through that the movements are too similar to each other. Another reason for this could be that the cost features which are used for inverse PI^2 are not sufficient for learning the difference.

This leaves two options; either the learned cost functions are very similar or PI^2 is not working as intended with the learned cost function. In Appendix C the learned cost function and the according learned trajectories are analysed. This analysis led to the conclusion that

PI^2 is working as expected. Therefore, it can be said that the learned trajectories do not show the clear differences in performance since the learned cost functions are similar.

8.4 Replay on robot

When replaying the learned trajectories on the robot, the conclusion is made that the demonstrations are hard to evaluate according to P_1. With the overall performance measurement P_1 the amount of floes broken off is needed. Unfortunately, this measurement was influenced through more factors are taken into account for learning. The exact placement of the breaking tool for breaking all the floes should be correct according to millimetres, this is hard to evaluate through the eye.

Furthermore, a big difference in the results seems to be due to which batch of floes is taken. The black series were much harder to break off the floes, than for the silver batch. Since the input demonstrations were ranked based on the black series and for the results the silver batch has only been available it is hard to say something about the results. What is seen during testing:

- Exact placement really important
- Speed when passing the breaking point is important, especially for the black batch. For the silver batch, this was less important. This is why it is desired want to start as low as possible with accelerating, only in the setting that was used there was a minimum for the z coordinate since otherwise the gripper would come into collision with the breaking tool.

8.5 Conclusion

The results confirm hypothesis one, that not the amount of unranked demonstrations is important, but the average performance of those demonstrations. Another important issue to solve is how the initialisation of the DMPs used as input for RL should be done since this has a big influence on the learned trajectories.

The second hypothesis can not be confirmed with the results of this research. So further adding weight factors to the input demonstrations based on the performance of the non-robotic expert is not improving the performance of the learned trajectories. It can be argued whether this is always the case or whether the variations between the demonstrations are not large enough or the features are not specific enough. For example, more steering features could be added.

8.6 Recommendations

In order to obtain statistical evidence for the above conclusions more data points are needed. This thesis is based on only a few, at maximum nine, learned trajectory per test case, because of the limited time available. To fully discard that the ranking the demonstrations through the preferences is not improving the performance of the learned trajectory, more learned trajectories are needed.

However, at the moment, the results suggest that ranking the demonstrations before using IRL is not improving the results. And therefore this does not seem to be the limiting factor for the performance. The limiting factor seems to be the cost features and the gathering demonstrations with enough variation.

As it turned out, it is not a trivial task to create the cost features for inverse PI^2 to make the algorithm learn the optimal cost function from which it can learn better than the input demonstrations or the average of the input demonstrations. Therefore, further research on what good elements would be for the cost function is necessary. Also, another look should be taken in what features are that tune the amount of floes broken off.

Another approach would be to record new demonstrations with more variation in the performance measurements such as execution time, smoothness, among others. This would lead to more difference in the input demonstrations and therefore, it is more likely to see the effect of adding the different weight factors to the demonstrations. This would also help with

the normalization of the features since the minimum and maximum values of the different features are then more apart from each other. Only this can lead to the paradox, that on one hand the system needs really bad demonstrations to normalize the features, while the input demonstrations should be performing relatively well.

Another disadvantage of the approach of creating different types of demonstrations is that it is quite hard to manipulate the UR5 in the teach mode. Therefore, the input trajectories always look a bit jerky and have a higher execution time than when the robot would have been programmed. This means that it is hard to create really fast demonstrations or a very smooth demonstration, or a combination of both. Most of the time when recording the demonstrations it was not the choice to perform faster or slower, it was more depended on the compliance of the robot.

Besides the above mentioned comments it would be interesting to look at more generic cost features. For this research as mentioned in 4.2.1 the cost features were chosen to be task specific. It would actually be a more useful algorithm when those features are generic for more tasks. So there is no need of a robotic expert to create those features per task. Also, it is interesting to check whether the learned trajectories are directly applicable to other robots. Moreover, it would be interesting to look at different tasks if the same results are found.

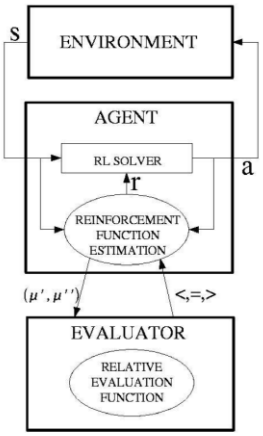


Figure 8-1 – Using an evaluator to update the reward function (r) (Silva et al., 2006)

For future work, when the above mentioned recommendations are fixed it could be interesting to look at another approach to avoid the problems with kinaesthetic teaching. The approach of Silva et al. (2006) as is shown in Figure 8-1. Here the non-robotic expert is the evaluator, which needs to evaluate after each learning step between the new and old trajectories which one is better performing. With this approach, the robot can be used to its full capabilities and having no influence on the limitations of kinaesthetic teaching. A disadvantage of this method can be that between two trials the difference in performance is very hard to notice. Another disadvantage is that the non-robotic expert has to be there during the time-expensive learning process.

Appendix A – Cost features for IRL

Table 1 - All cost features used for inverse PI^2 in the order the appear in the code. The first column shows the formulation used, the second column what the purpose of the feature is.

	Description of the feature	*The feature origin*
1	$\phi_1(\theta,t) = \frac{1}{2}\left(\theta + M_t\epsilon_{k,t}\right)^T L(\theta + M_t\epsilon_{k,t})$	Cost of Policy Parameters
2	For the first time step, this feature is trying to keep the y coordinate constant $\phi_2(y,t) = \lvert y - y_0\rvert + \lvert\dot{y}\rvert$ for $t = [\,0{:}\,3]$ else $\phi_2 = 0$	Performance: Execution Time
3	$\phi_3(y,t) = \lvert y - y_0\rvert + \lvert\dot{y}\rvert$ for $t = [3{:}\,6]$ else $\phi_3 = 0$	Performance: Execution Time
4	$\phi_4(y,t) = \lvert y - y_0\rvert + \lvert\dot{y}\rvert$ for $t = [6{:}\,9]$ else $\phi_4 = 0$	Performance: Execution Time
5	$\phi_5(y,t) = \lvert y - y_0\rvert + \lvert\dot{y}\rvert$for $t = [9{:}\,12]$ else $\phi_5 = 0$	Performance: Execution Time
6	$\phi_6(y,t) = \lvert y - y_0\rvert + \lvert\dot{y}\rvert$ for $t = [12{:}\,14]$ else $\phi_6 = 0$	Performance: Execution Time
7	This feature keeps the z coordinate constant during the time period where the y coordinate is moving. $\phi_7(z,t) = \lvert z - z_0\rvert + \dot{z}$ for $t = [0{:}\,14]$ else $\phi_7 = 0$	Performance: Execution Time
8	$\phi_8(y,t) = \lvert y - y_e\rvert + \dot{y}$ for $t = [12.5 : 15.5]$ else $\phi_8 = 0$	Performance: Execution Time
9	$\phi_9(z,t) = \lvert z - z_0\rvert + \dot{z}$ for $t = [14.5 : 15.5]$ else $\phi_9 = 0$	Performance: Execution Time
10	$\phi_{10}(x,t) = \lvert\ddot{y}\rvert + \lvert\ddot{z}\rvert$ for $t = [0{:}\,3]$ else $\phi_{10} = 0$	Performance: Smoothness
11	$\phi_{11}(x,t) = \lvert\ddot{y}\rvert + \lvert\ddot{z}\rvert$ for $t = [3{:}\,6]$ else $\phi_{11} = 0$	Performance: Smoothness
12	$\phi_{12}(x,t) = \lvert\ddot{y}\rvert + \lvert\ddot{z}\rvert$ for $t = [6{:}\,9]$ else $\phi_{12} = 0$	Performance: Smoothness
13	$\phi_{13}(x,t) = \lvert\ddot{y}\rvert + \lvert\ddot{z}\rvert$ for $t = [9{:}\,12]$ else $\phi_{13} = 0$	Performance: Smoothness
14	$\phi_{14}(x,t) = \lvert\ddot{y}\rvert + \lvert\ddot{z}\rvert$ for $t = [12{:}\,15.5]$ else $\phi_{14} = 0$	Performance: Smoothness
15	$\phi_{15te} = \sum_{t=0}^{t_e} \lvert\dot{y}\cdot dt\rvert + \lvert\dot{z}\cdot dt\rvert$ (for $t = t_e$)	Performance: Path taken
16	If $\dot{y}(t) < 0$ then $\phi_{16} = -(\dot{y}(t))$ else $\phi_{16} = 0$	Performance: Path taken, no backwards motion
17	If $\dot{z}(t) < 0$ then $\phi_{17} = -(\dot{z}(t))$ else $\phi_{17} = 0$	Performance: Path taken, no backwards motion
18	$\phi_{18}(y,t) = \left\lvert y(t) - y_g\right\rvert$ for $t = [\,0{:}\,3]$ else $\phi_{18} = 0$	Steering: Bring y in position
19	$\phi_{19}(y,t) = \left\lvert y(t) - y_g\right\rvert$for $t = [3{:}\,6]$ else $\phi_{19} = 0$	Steering: Bring y in position
20	$\phi_{20}(y,t) = \left\lvert y(t) - y_g\right\rvert$ for $t = [6{:}\,9]$ else $\phi_{20} = 0$	Steering: Bring y in position
21	$\phi_{21}(y,t) = \left\lvert y(t) - y_g\right\rvert$ for $t = [9{:}\,12]$ else $\phi_{21} = 0$	Steering: Bring y in position
22	$\phi_{22}(y,t) = \left\lvert y(t) - y_g\right\rvert$ for $t = [12{:}\,14.5]$ else $\phi_{22} = 0$	Steering: Bring y in position
23	$\phi_{23}(x,t) = \left\lvert y(t) - y_{beforebreaking}\right\rvert + \lvert z(t) - z_{beforebreaking}\rvert$ for $t = [12{:}\,14]$ else $\phi_{23} = 0$	Steering: Before breaking the floes
24	$\phi_{24}(x,t) = \lvert y(t) - y_{break}\rvert + \lvert z(t) - z_{break}\rvert$ for $t = [14{:}\,15]$ else $\phi_{24} = 0$	Performance: Breaking off floes simulation
25	If $y(t) > y_{max}$ then $\phi_{25} = \lvert y(t) - y_{max}\rvert$ else $\phi_{25} = 0$	Performance: Collision
26	If $z(t) < z_{min}$ then $\phi_{26} = \lvert z(t) - z_{min}\rvert$ else $\phi_{26} = 0$	Performance: Collision

Appendix B – Input demonstrations and the corresponding performance

Table 1 – Performance of the recorded demonstrations

Demonstration ID #	*Amount of floes broken off*	*Shortest distance to breaking point 1.0e-04 m*	*Execution Time (seconds)*	*Overall Smoothness (m/s^2)*	*Path taken (m/s)*	*No backward motion y (m/s)*	*No backward motion z (m/s)*
1	1	0.1600	5.58	127.6211	24.5422	0.3398	1.1402
2	2	0.0900	6.21	128.2390	26.2778	0.2033	1.1218
3	2	0.0400	12.43	155.3608	29.5149	0.4477	2.6927
4	2	0	9.63	147.4194	28.7608	0.6932	2.1204
5	2	0.1600	8.26	153.7105	30.5306	0.2717	1.6356
6	2	0.0400	10.56	142.8810	29.4426	0.2283	1.7488
7	2	0.0900	6.93	150.1551	27.6728	0.1189	1.3074
8	1	0.2500	7.73	153.6194	29.7748	0.6351	0.8043
9	2	0.2500	12.77	171.9735	33.5464	0.4836	1.1704
10	3	0	11.92	157.5166	34.1211	0.4635	1.9706
11	1	0	14.03	168.4038	32.2831	1.1412	1.6510
12	2	0.1600	11.36	171.2847	34.7210	0.9419	2.2199
13	2	0.2500	7.97	135.1845	29.9549	0.4771	0.7824
14	2	0.0900	10.73	169.9748	33.8322	1.1463	1.5900
Mean trajectory	N/A	0	12.22	120.1540	28.2340	0.2604	0.8056

Table 2 – An overview of the answers given at the survey which are categorized by different groups of combinations.

	a>b	*a<b*	*a=b*	*a?b*	*# clear preferences (a</>b)*
All preferences	189	168	128	61	357 (65.4%)
Different amount of floes	102	101	28	21	203 (81.0%)
Same amount of floes broken off	87	67	100	40	154 (52.4%)

Table 3 – the correlation values ($c(p)$) of the performance measurements divided in two categories. The first category is when the preferences do not agree with the performance measurement the amount of floes broken off. The second category they do agree.

	Amount of preferences	*Amount of floes broken off*	*Shortest distance to breaking point 0.0001 m*	*Execution time*	*Smoothness*	*Path taken*	*No backward motion y*	*No backward motion z*
*Preference does **not** agree with the performance measurement the amount of floes broken off*	38	-1	-0.6970	0.5263	0.2632 0.7895 0.3158 -0.4737 -0.3684 0.2632	0.2632	-0.1579	0.6316
Preference does agree with the performance measurement the amount of floes broken off	165	1	0.3143	-0.2364	-0.1758 -0.5394 -0.3455 0.2000 0.3697 -0.1394	-0.3333	0.3455	-0.3455

Table 4 - Performance of the demonstrations, according to P_1 and P_2, also the performance according to P_2 of the mean is presented.

Demonstration ID #	P_1	*Order according to P_1*	P_2	*Order according to P_2*
1	0.5231	12	0.7491	2
2	0.7493	2	0.7934	1
3	0.5817	8	0.5641	8
4	0.6507	7	0.5161	10
5	0.6709	5	0.5652	7
6	0.6628	6	0.6615	4
7	0.7158	3	0.6762	3
8	0.4462	13	0.5121	11
9	0.5590	11	0.5199	9
10	0.8305	1	0.6281	5
11	0.3015	14	0.4820	12
12	0.5621	10	0.4123	13
13	0.6988	4	0.5694	6
14	0.5808	9	0.4112	14
Mean trajectory	N/A	N/A	0.8180	N/A

Appendix C – Learned trajectories and their corresponding weights

To better understand what is happening during the learning process, Figure 2, Figure 3, and Figure 4 are showing the learned trajectories and Figure 1 the input demonstrations. What can be seen is that although the demonstrations differ in performance, they look very similar. The main difference can be found in when the y coordinate is moving towards 'the breaking tool'.Table 1, Table 2, and Table 3
Table 3 shows the corresponding weights of the cost function which is used to create the learned trajectories.

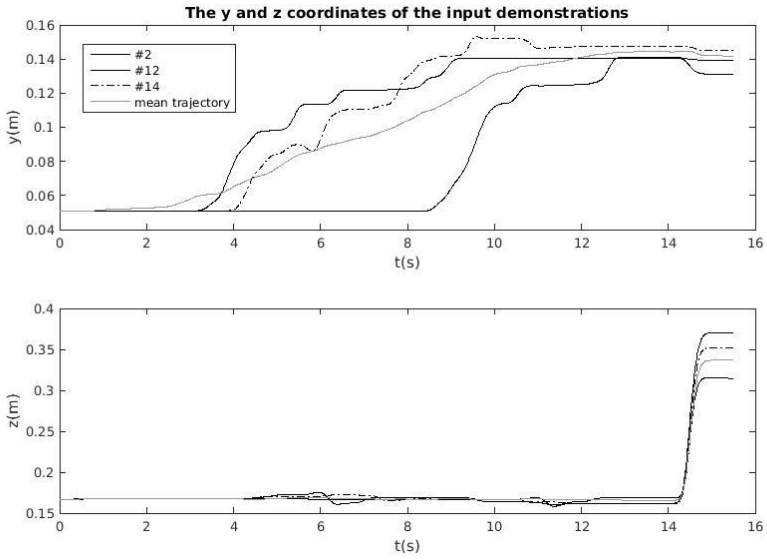


Figure 1 –The input demonstrations of the learned demonstrations shown in Figure 7-7, Figure 2, Figure 3, and Figure 4.

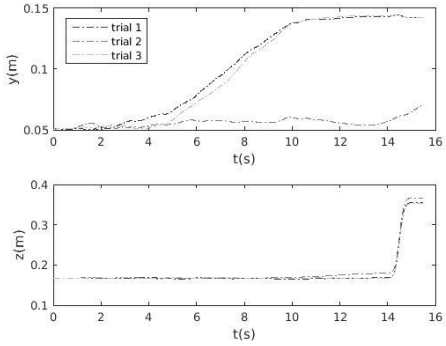


Figure 2 - Learned trajectories without weight factors

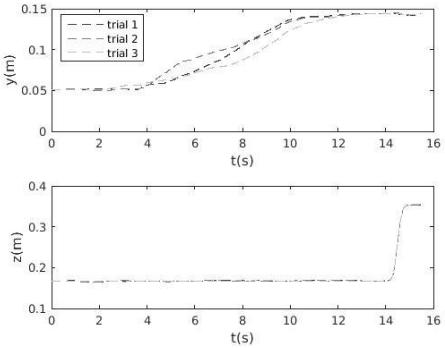


Figure 3 - Learned trajectories with weight factor 1

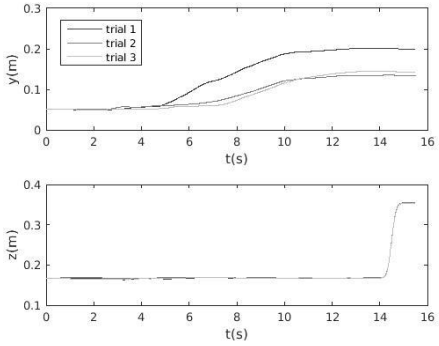


Figure 4 - Learned trajectories with weight factor 2

What can be seen that sometimes the weights are focussing on the wrong feature, e.g. in Figure 2 the trajectory of trial 2 the y coordinate is tried to keep constant. What can be explained through the negative weights w_{23} and w_{24}, which is corresponding to the features before breaking and during breaking shown in Table 1.
Furthermore, do all the learned trajectories look quite similar, unless, for Figure 4 where it seems that the y coordinate is kept constant until 6 seconds, this can be explained through the weight w_2, w_3, w_{18}, w_{19}, and a little bit through w_{20}, which are shown in
Table 3. It shows that at one hand the y coordinate is tried to keep constant for the first 2/3 time steps, and on the other hand the y coordinate is should not be close to breaking point.

Table 1 – weights of the cost function used to create the learned trajectories in Figure 2.

Trial	w_1	$w_2 - w_9$	$w_{10} - w_{14}$	$w_{15} - w_{17}$	$w_{18} - w_{22}$	w_{23}	w_{24}	$w_{25} - w_{26}$
Without added preferences Trial 1	-0.0380	1.0000 0.9744 0.7565 -0.5795 0.6489 1.0000 0.7933 0.8204	1.0000 1.0000 1.0000 0.8779 0.5937	1.0000 1.0000 1.0000	-0.5458 -0.3513 0.3558 0.7693 0.3906	1.0000	-0.9077	0.7268 1.0000
Without added preferences Trial 2	0.0516	1.0000 0.6251 0.6496 -0.6872 0.7778 1.0000 0.7550 0.1680	1.0000 1.0000 1.0000 0.9441 0.6454	1.0000 1.0000 1.0000	0.6441 0.9312 0.8731 0.9856 -0.4524	-0.0448	-0.5917	0.8583 1.0000
Without added preferences Trial 3	-0.0371	0.9355 0.8907 0.6773 -0.6454 0.2312 1.0000 0.6839 0.5542	1.0000 1.0000 1.0000 0.9018 0.6246	1.0000 1.0000 1.0000	0.1982 -0.2577 0.4963 0.9573 1.0000	1.0000	0.7734	0.8114 1.0000

Table 2 - weights of the cost function used to create the learned trajectories in Figure 3.

Trial	w_1	$w_2 - w_9$	$w_{10} - w_{14}$	$w_{15} - w_{17}$	$w_{18} - w_{22}$	w_{23}	w_{24}	$w_{25} - w_{26}$
With added preferences through weight factor 1 Trial 1	0.0253	0.6707 0.6064 0.5132 -0.4431 0.0521 1.0000 -0.3891 -0.1056	0.9756 1.0000 0.9455 0.7188 0.4407	1.0000 1.0000 1.0000	-0.2988 -0.3921 -0.2690 0.5323 0.2388	1.0000	0.5631	0.5572 0.8028
With added preferences through weight factor 1 Trial 2	0.0304	0.7073 0.6937 0.6672 0.2350 0.8373 1.0000 0.8025 0.3035	0.9611 0.9909 0.9634 0.5862 0.4146	1.0000 1.0000 1.0000	-0.3592 0.3042 0.4161 0.2529 0.5275	0.9365	-0.4942	0.5264 0.7456
With added preferences through weight factor 1 Trial 3	-0.0252	0.6512 0.7055 0.7277 0.3890 0.5945 1.0000 0.6908 0.2140	0.9557 0.9992 0.9587 0.6501 0.4112	1.0000 1.0000 1.0000	-0.3390 -0.4032 -0.3973 0.5910 0.7684	1.0000	0.4582	0.5826 0.8409

Table 3 - weights of the cost function used to create the learned trajectories in Figure 4.

Trial	w_1	$w_2 - w_9$	$w_{10} - w_{14}$	$w_{15} - w_{17}$	$w_{18} - w_{22}$	w_{23}	w_{24}	$w_{25} - w_{26}$
With added preferences through weight factor 2	0.0273	0.6799 0.7128 0.6204 -0.5386 0.5985	0.9533 0.9899 0.9798 0.6578 0.4204	1.0000 1.0000 1.0000	-0.3451 -0.2861 0.5230 0.4735 -0.8718	0.7325	-0.7276	-0.3898 0.9284
Trial 1		1.0000 0.5754 0.3256						
With added preferences through weight factor 2	0.0124	0.6508 0.7592 0.8078 0.5567 0.8158	0.9534 0.9687 0.9720 0.6747 0.4481	1.0000 1.0000 1.0000	-0.0840 -0.2925 -0.4655 0.5114 -0.4380	0.9512	0.3168	0.5781 0.8069
Trial 2		1.0000 0.8582 0.3271						
With added preferences through weight factor 2	0.0151	0.6204 0.7634 0.7314 0.1424 0.7180	0.9718 1.0000 0.9741 0.6194 0.4271	1.0000 1.0000 1.0000	-0.1985 -0.4939 -0.4231 -0.3730 0.5410	-0.4623	0.4582	0.4328 0.7281
Trial 3		1.0000 -0.5406 0.4390						

Appendix D – Selected trajectories to be fed back to the robot

To know how the learned trajectories look in real life some trajectories are selected. The first trajectory that is selected to be fed back to the robot is the mean trajectory. This trajectory is chosen because it is used as starting point for RL and furthermore it has a very high performance according to P_2.

The mean trajectory looks as shown in Figure 1. What can be seen that it is a quite smooth movement; this is because sudden changes in position are equalized through taking the average of all demonstrations.

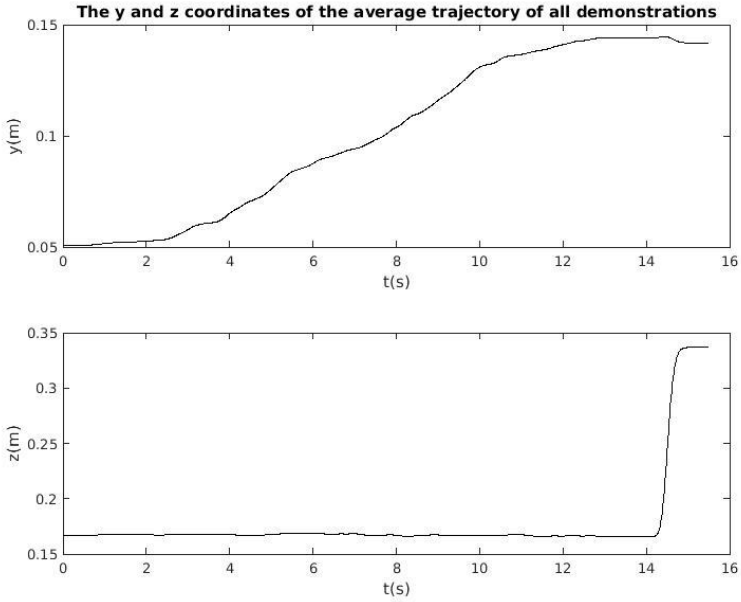


Figure 1 – the y and z coordinates over time of the average trajectory of all demonstrations, which is also used as initialization of the DMPs for PI^2.

Decided is to replay the best evaluated trajectory through P_2 that has learned from 2 demonstrations. Chosen for the situation of 2 demonstrations, because also in Figure 7-3 this was the situation where the highest performance was reached.

Furthermore was decided to feed the following trajectories back to the robot:

- The best performing trajectory according to P_2 from the case 2B1G and from the case 1B2G, where the demonstrations that were selected through P_1.
- The best performing trajectory according to P_2 from the case 2B1G and from the case 1B2G, where the demonstrations that were selected through P_2.

Furthermore was decided to replay the input trajectories that were used to create those demonstrations to check the test environment.

Table 1 and Table 2 shows the amount of floes that were broken when those demonstrations were fed back to the robot. For the new test, only a different batch of floes was used, the 'silver' batch. These floes were a lot easier to break off than the other batch, the black batch which was used in the testing can be seen in Table 2. There were not enough floes of the black batch set to test all trajectories with.

In the tables can be seen that the amount of floes broken off varies even between trials and also the amount of floes broken off do not seem to improve when adding preferences.

What was noticed when the mean was fed back to the robot with the 'black' batch that no of the floes were broken off, but all the floes were bent and therefore had been in contact with the 'breaking' tool. This is likely due to the slow speed at which it passed the breaking tool.

Table 1 - The amount of floes broken off for the selected learned trajectories

Trajectory	*Silver*	*Black*	*Performance according to P_2*
Mean	3,3	0	0.8180
With 2 good demonstrations selected through P_1	3,3		
With 2 good demonstrations selected through P_2	3,3		
1g2b no added preferences selected through P_1	3,3		0.6976
1g2b added preferences trough weight factor 2 selected through P_1	2,2		0.6931
1g2b added preferences trough weight factor 3 selected through P_1	3,2		0.7457
2g1b no added preferences selected through P_1	2,3,3		0.7133
2g1b added preferences trough weight factor 2 selected through P_1	3,3		0.7524
2g1b added preferences trough weight factor 3 selected through P_1	3,3		0.7252
1g2b no added preferences selected through P_2	2,3		0.7188
1g2b added preferences trough weight factor 2 selected through P_2	3,2		0.7267
1g2b added preferences trough weight factor 3 selected through P_2	3,2		0.7191
2g1b no added preferences selected through P_2	3,2		0.7516
2g1b added preferences trough weight factor 2 selected through P_2	2,2		0.7430
2g1b added preferences trough weight factor 3 selected through P_2	3,3		0.7282

Table 2 - The amount of floes broken off with the different batches for a couple of input demonstrations

Demonstration ID#	*Silver*	*Black*	*(recorded before)*
1	3,3		1
2	3		2
4	3		2
6	3		2
7	3		2
8	2		1
10	3,3	2	3
11	2	1	1
12	3		2
14	3 (but collision)		2

Appendix E – Questionnaire

soSci
ofb - der onlineFragebogen

Hello,

For my thesis I am looking into how to teach a Universal Robot 5 (UR5)* a production task. More specifically I am looking into the possibilities of using pair-wise preferences to gain more information about specific trials, to speed up the learning process.

Your data will be kept anonymous, but I do need to know the amount of experience you have in robotics or with similar production tasks in factories. Therefore the experiment will start with a few questions about your experiences and your background.

Afterwards you will be given 7 questions, where two trials of the production task are shown. You are asked which of the two trials is performing better than the other one. The questionnaire will take around 15 minutes.

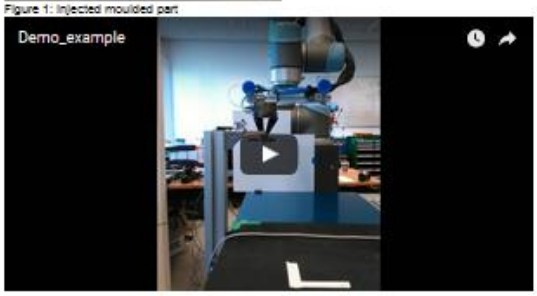
Task
The production task, which the UR5 is aiming to perform, is to break off the three circles of the injected moulded part of Figure 1. The task needs to be executed as quickly as possible, so the production can be maximized and smoothly as possible to prevent wear and tear to the robot.

In the video below you can see an example of how a trial looks like.

Kind regards,
Renée

*(www.universal-robots.com/products/ur5-robot/)

Figure 1: Injected moulded part



Video: An example of how a trial looks like.

Figure 2: The injected moulded part under the breaking tool.

Figure 3: The set up of the UR5 with the breaking tool

Next

Renee van der Wijden, TU Delft – 2016

0% completed

soSci
oFb - der onlineFragebogen

Please answer the following questions on your experiences and background:

1. How much experience do you have in robotics?

- 0 year (I have never worked with it)
- 0-1 year (I have worked with it)
- 1-3 years
- 3 or more years

2. How much experience do you have with an UR5 robot?

- 0 year (No experience, I have never heard from it)
- 0-0.5 year (I have heard from it, but I have never worked with it)
- 0.5-1 year (I have worked with it)
- 1-3 years
- 3 or more years

3. How much experience do you have with artificial intelligence or machine learning?

- 0 year (No experience, I have never worked with it)
- 0-0.5 year (I have heard from it, never worked with it)
- 0.5-1 year (I have worked with it)
- 1-3 years
- 3 or more years

4. How much experience do you have with preference learning or pair-wise preference reporting?

- 0 year (No experience, I have never heard from it)
- 0-0.5 (I have heard from it, but I have never worked with it)
- 0.5-1 year (I have worked with it)
- 1-3 years
- 3 or more years

5. How much experience do you have with reinforcement learning?

- 0 year (No experience, I never heard from it)
- 0-0.5 year (I have heard from it, I have never worked with it)
- 0.5-1 year (I have worked with it)
- 1-3 years
- 3 or more years

6. Are you familiar with inverse reinforcement learning?

- Yes, how:
- No

7. Gender:

- Man
- Woman

8. Age:

years

9. Education Level

[Please choose]

10. Educational Background

- Computer Science Engineering
- Electrical Engineering
- Mechanical Engineering
- Psychology
- Others, please specify:

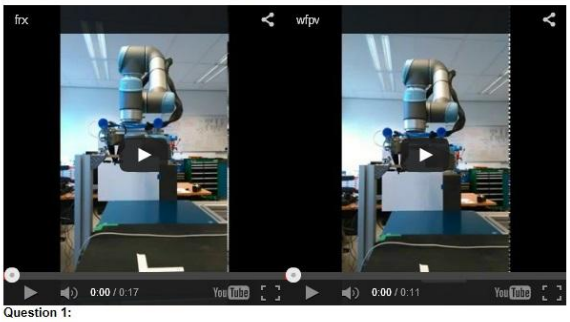
Additional comments (optional)

soSci
oFb - der onlineFragebogen

Please watch the videos and answer the question.

"Video A" is on the left and "video B" is on the right.

You can watch (and pause) the videos as many times as you want



Question 1:

- A performs better than B (A>B)
- A performs worse than B (A<B)
- A and B perform equally good or bad (A=B)
- I don't know which one is performing better (A?B)

Additional comments (optional)

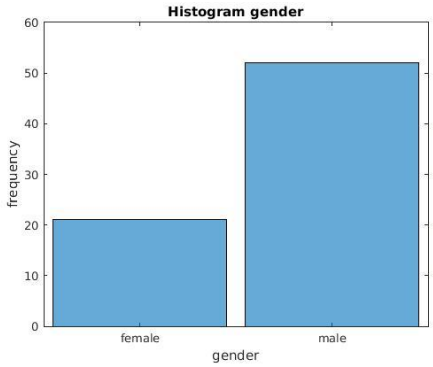
Next

Appendix F – Demographic data of the crowdsourcing experiment

The raw data of the crowdsourcing experiments can be found via: *https://docs.google.com/spreadsheets/d/1o6OpbDK5aaCIFiKqxDrDP_J3VEi0lwRCjCp8QXerZ9E/edit?usp=sharing*

Below the demographic data is shown.

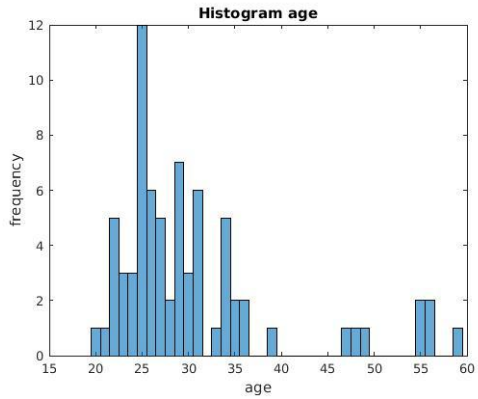
Gender:



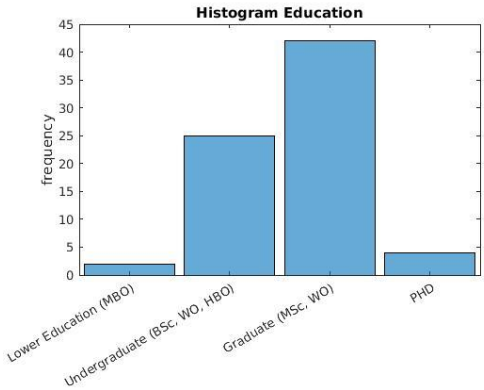
There are 21 female and 53 male that filled in the survey.

Age:

The average age of the people who filled in the survey is 30.51 years old with a standard deviation of 9,06.

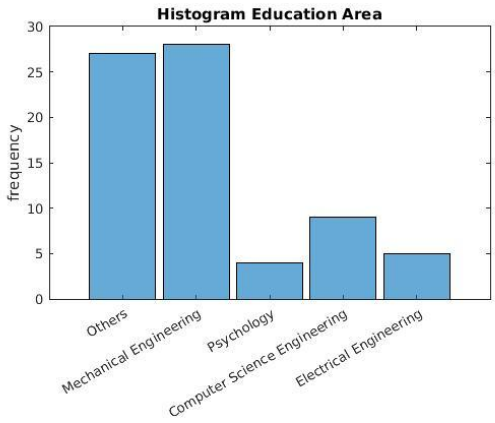


Education Level:



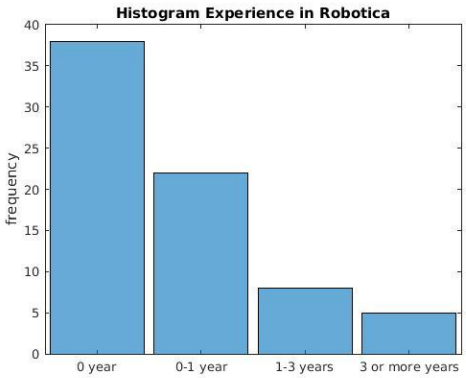
2 people did lower Education, 25 are undergraduate, 52 graduate and 4 phd'ers.

Educational Field:



Others were:
Aerospace, agriculture, Animal Husbandry, Applied Science, Architecture, Artificial Intelligence, BioMechatronics, Biomedical Engineering, Civil Engineering, Computer Science, Education, Geosciences, Industrial Design Engineering, Management, Marine Technology, MBA, Mechatronics, Medical, Physiotherapy, Political Science, System & Control.
The main conclusion that can be drawn, is that most people had a technical background and were higher educated.

Experiences:

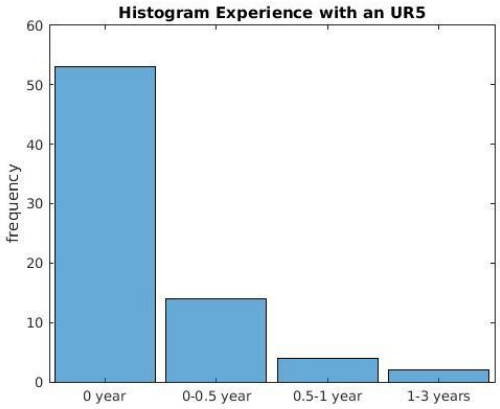


Robotics:

0 year(I have never worked with it)
0-1 year (I have worked with it)
1-3 years
3 or more years

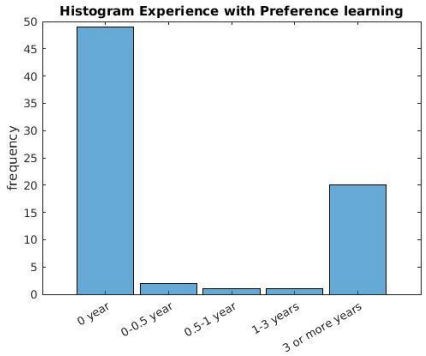
Some people commented that they did a course on it, but never worked with it.

Experience with an UR5:



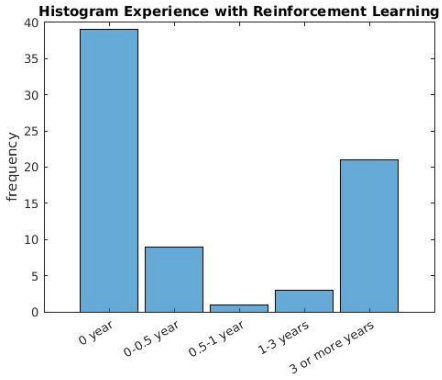
0 year (No experience, I have never heard from it)
0-0.5 year (I have heard from it, but I have never worked with it)
0.5-1 year (I have worked with it)
1-3 years
3 or more years.

PL



0 year (No experience, I have never heard from it)
0-0.5 year (I have heard from it, but I have never worked with it)
0.5-1 year (I have worked with it)
1-3 years
3 or more years

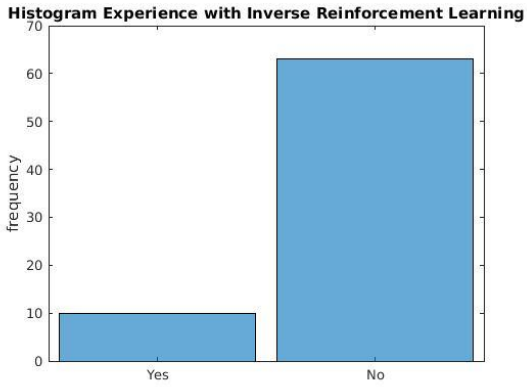
Experience RL:



0 year (No experience, I have never heard from it)
0-0.5 year (I have heard from it, but I have never worked with it)
0.5-1 year (I have worked with it)
1-3 years
3 or more years

What I say during scrolling through the results is that some people, who were not technical educated also said they knew it for over more then 3 years. I have the feeling they meant a different kind of reinforcement learning, the one of teaching little children.

Experience IRL:



People who said to have heard about it, 4 knew it from my stories. 5 people knew it from studies and 1 wrote a paper about it.

Bibliography

Aalamifar, F. (2015). UR5 Control Using Matlab. 15 December 2015

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. Paper presented at the Proceedings of the twenty-first international conference on Machine learning.

Andrew, G., & Gao, J. (2007). Scalable training of L 1-regularized log-linear models. Paper presented at the Proceedings of the 24th international conference on Machine learning.

Argall, B. D., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. Robotics and autonomous systems, 57(5), 469-483.

Bautista-Ballester, J., Vergés-Llahí, J., & Puig, D. (2014). Programming by Demonstration: A Taxonomy of Current Relevant Methods to Teach and Describe New Skills to Robots. Paper presented at the ROBOT2013: First Iberian Robotics Conference.

Cambridge-Dictionaries. (2016). Performance in British English.

Cheng, W., Fürnkranz, J., Hüllermeier, E., & Park, S.-H. (2011). Preference-based policy iteration: Leveraging preference learning for reinforcement learning Machine Learning and Knowledge Discovery in Databases (pp. 312-327): Springer.

Dembczyński, K., Kotłowski, W., Słowiński, R., & Szeląg, M. (2010). Learning of rule ensembles for multiple attribute ranking problems Preference Learning (pp. 217-247): Springer.

Doerr, A., Ratliff, N., Bohg, J., Toussaint, M., & Schaal, S. (2015). Direct loss minimization inverse optimal control. Proceedings of robotics: science and systems (R: SS), 1-9.

Englert, P., Paraschos, A., Deisenroth, M. P., & Peters, J. (2013). Probabilistic model-based imitation learning. Adaptive Behavior, 21(5), 388-403.

Fürnkranz, J., & Hüllermeier, E. (2010). Preference learning: Springer.

Grappiolo, C., Martínez, H. P., & Yannakakis, G. N. (2014). Validating Generic Metrics of Fairness in Game-based Resource Allocation Scenarios with Crowdsourced Annotations Transactions on Computational Intelligence XIII (pp. 176-200): Springer.

Hansen, N. (2006). The CMA evolution strategy: a comparing review Towards a new evolutionary computation (pp. 75-102): Springer.

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., & Schaal, S. (2013). Dynamical movement primitives: learning attractor models for motor behaviors. Neural computation, 25(2), 328-373.

Ijspeert, A. J., Nakanishi, J., & Schaal, S. (2002a). Learning attractor landscapes for learning motor primitives.

Ijspeert, A. J., Nakanishi, J., & Schaal, S. (2002b). Movement imitation with nonlinear dynamical systems in humanoid robots. Paper presented at the Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on.

Kalakrishnan, M., Pastor, P., Righetti, L., & Schaal, S. (2013). Learning objective functions for manipulation. Paper presented at the Robotics and Automation (ICRA), 2013 IEEE International Conference on.

Kober, J., & Peters, J. (2009). Learning motor primitives for robotics. Paper presented at the Robotics and Automation, 2009. ICRA'09. IEEE International Conference on.

Kober, J., & Peters, J. (2012). Reinforcement learning in robotics: A survey Reinforcement Learning (pp. 579-610): Springer.

Koenig, L. L., Lucero, J. C., & Perlman, E. (2008). Speech production variability in fricatives of children and adults: Results of functional data analysis. The Journal of the Acoustical Society of America, 124(5), 3158-3170.

Kormushev, P., Calinon, S., & Caldwell, D. G. (2013). Reinforcement learning in robotics: Applications and real-world challenges. Robotics, 2(3), 122-148.

MathWorks. (2016). fmincon.

Muelling, K., Boularias, A., Mohler, B., Schölkopf, B., & Peters, J. (2014). Learning strategies in table tennis using inverse reinforcement learning. Biological cybernetics, 108(5), 603-619.
Ng, A. Y., & Russell, S. J. (2000). Algorithms for inverse reinforcement learning. Paper presented at the Icml.
Peters, J., & Schaal, S. (2008a). Natural actor-critic. Neurocomputing, 71(7), 1180-1190.
Peters, J., & Schaal, S. (2008b). Reinforcement learning of motor skills with policy gradients. Neural networks, 21(4), 682-697.
Ramachandran, D., & Amir, E. (2007). Bayesian inverse reinforcement learning. Urbana, 51, 61801.
Ratliff, N. D., Bagnell, J. A., & Zinkevich, M. A. (2006). Maximum margin planning. Paper presented at the Proceedings of the 23rd international conference on Machine learning.
ROS. (2016).
Rubinstein, R. Y., & Kroese, D. P. (2013). The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning: Springer Science & Business Media.
Sakoe, H., & Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. Acoustics, Speech and Signal Processing, IEEE Transactions on, 26(1), 43-49.
Schaal, S. (1997). Learning from demonstration. Advances in neural information processing systems, 1040-1046.
Silva, V. F., Costa, A. H. R., & Lima, P. (2006). Inverse reinforcement learning with evaluation. Paper presented at the Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on.
Smith, A., & Anderson, J. (2014). AI, Robotics, and the Future of Jobs. Pew Research Center.
Stulp, F., & Sigaud, O. (2012). Path integral policy improvement with covariance matrix adaptation. arXiv preprint arXiv:1206.4621.
Sugiyama, H., Meguro, T., & Minami, Y. (2012). Preference-learning based Inverse Reinforcement Learning for Dialog Control. Paper presented at the INTERSPEECH.
Sutton, R. S., & Barto, A. G. (1998). Reinforcement learning: An introduction (Vol. 1): MIT press Cambridge.
Theodorou, E., Buchli, J., & Schaal, S. (2010). A generalized path integral control approach to reinforcement learning. The Journal of Machine Learning Research, 11, 3137-3181.
Thurstone, L. L. (1927). A law of comparative judgment. Psychological review, 34(4), 273.
Universal-Robots. (2015). UR5 Robot.
Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 8(3-4), 229-256.
Zhifei, S., & Joo, E. M. (2012). A review of inverse reinforcement learning theory and recent advances. Paper presented at the Evolutionary Computation (CEC), 2012 IEEE Congress on.