Limits and Contraints of Reinforcement Learning in Affordances

A. Akhtar[a], H. Almaree[c], A. Busse[a], S. Hossain[a], S. Jahan[a], K. Kuribayashi[a], L. Matayeva[a], M. Naghavipour[a], S. Ojog[b], M. Rahman[a], A. Reveriuk[b], F. Schneekloth[b], J. Schramm[b], A. Stricker[c], and K. XU [a], R. Weller[*], H. Meißenhelter[*]G. Zachmann[*]

[a]M.Sc. Digital Media, University of Bremen
[b]M.Sc. Computer Science, University of Bremen
[c]B.Sc. Digital Media, University of Bremen
[*]Department of Computergraphics, University of Bremen

March 16, 2023

Abstract

Affordances are inherent properties of objects and environments that enable certain actions to be executed. It is a concept originating from psychology that has gained increased attention in robotics as it provides a way to interpret environments and optimise planning and execution of actions. At the same time, reinforcement learning has become a crucial part of problem solving in robotics as well. In this paper, we explore the limitations of reinforcement learning in the context of affordances. We specifically focus on the physical execution of affordances and show that they provide rather complex situations—proving to be quite difficult for reinforcement learning in certain cases. We also argue that the challenges can be overcome by splitting affordances into simpler components and propose a taxonomy of these components to accommodate this concept. Finally, we explore whether or not virtual environments are suited to simulate real world affordances in the first place. We come to the conclusion that, while they are not inherently ill-suited, virtual environments do encounter some limitations that need to be accounted for when it is employed for real world simulation.

1. Introduction

The concept of affordances, introduced by J. J. Gibson, describes properties of objects and environments which enable various actions to achieve certain effects. As the fields of robotics and artificial intelligence, and, specifically, reinforcement learning (RL) continue to expand, accurately describing an environment's properties, potential actions, and desired outcomes are essential for developing and training a successful AI system. In this paper, we explore what limits may be encountered when using reinforcement learning for affordances, how accurate simulations are in comparison to the real world, and how these challenges can be approached. Specifically, we focus on the physical motions required to execute an affordance from the perspective of the agent.

2. Related Work

The concept of affordances, which originated in psychology, has received various proposed formalisations over the years. In 2007, Sahin et. al. proposed a new formalisation based on the major ones and showed that there are three different perspectives to view affordances. [1] Later, in 2017, Zech et. al. have provided a high level taxonomy of computational models of affordances in robotics. [2] In the same year, Nguyen et. al. used convolutional neural networks, an object detector, and dense conditional random fields in combination to present a method for detecting object-affordances—a specific set of affordances that is derived from the physical properties of an object. [3] In 2018, object-affordances have been used by Zamani et. al. to recognise user intentions and mapping them to a goal for which a sequence of actions is generated. [4] Similarly, in 2018, Bhattacharyya and Hazarika used object-affordances in conjunction with a model of human intent to extract reward functions from high level demonstrations. [5] More recently, in 2020, Khimya et. al. used affordances to reduce the actions available to an agent at any given moment, increasing planning speed and also enabling more efficient and stable learning of partial models which can generalise better than full models—specifically when function approximations are required. [6] Also in 2020, Wu et. al. have used robotic imagination of the execution of affordances to classify objects, with the example of a chair. [7] In 2015, Kumar and Todorov developed a virtual reality system to collect hand manipulation data by combining the MuJoCo physics engine with a cyber glove for use in robotics. [8] Similarly, the bachelor project VR-Rat implemented a VR-Application to record affordances inside the Unreal Engine for usage in robotics. [9] Another contribution related to the context of affordances is the work done by Ostiategui et. al. in 2010, showing the potential of virtual reality as a tool to solve household tasks, such as gardening in a safe environment, specifically for disabled people. [10]. Lastly Raikwar et. al. have shown in 2019, that not only virtual reality can be used to simulate affordances, it can also provide completely new ones at the example of an architecture platform in VR. [11]. In regards to the comparisons of motions, in 2016, Neil Vaughan and Bogdan Gabrys proposed the use of the dynamic time warping (DTW) algorithm to analyse the similarity between trajectories in a manner insensitive to sampling rate and numerical results. [12]

3. Methodology

For the experiments, we selected various affordances and implemented them in two environments. The first is an OpenAI Gymnasium environment with Mujoco and PyBullet as the physics engine, along with an implementation of DQN and PPO as learning algorithms. The other is an Unreal Environment in virtual reality, as well as a motion tracking set-up facilitated by OptiTrack Technology. We used the OpenAI environment to explore what kind of affordance properties may cause issues regarding the reinforcement learning process itself,in relation to the physics engine used. Based on our observations, we propose a taxonomy for affordance components which are required to achieve the desired end-state for fulfilling an affordance. The Unreal Environment is used to assess the accuracy of the physics engine in comparison to real world scenarios. The evaluation aims to determine whether simulated environments are inherently ill-suited for certain tasks and determines if training a system without real world data would be feasible.

3.1. Selected Affordances

The affordances are selected based on properties, which we estimated may pose challenges on the reinforcement learning, the physics engine, or the execution in virtual environments. Some of the selected affordances have properties that are challenging for reinforcement learning and are only tested in that context—not in the virtual unreal environment.

Covering a Pot

This affordance involves taking an object, such as a lid, and using it to cover a cooking pot. It is a rather simple task without any special properties, and serves mainly as a reference task. The end-state can be described by means of either collision or positioning, which can be used as the reward function. More specifically, the agent gets rewarded for closing the distance between pot and lid, and gains additional reward for the amount of collision between the pot's opening and the lid—reaching maximum reward when the lid is perfectly covering the pot's opening.

Flipping Object in a Pan

This functionality is afforded by a spatula. The object to be flipped is an artificial toast, interpreted as a rigid object. The difficulty in this task lies in the fact that the toast can not be manipulated directly. Instead, only the spatula can be controlled and the toast needs to be flipped through the interaction between spatula and toast. The reward function is defined by the rotation of the toast, i.e. maximum reward is achieved when the up-axis of the toast is flipped 180 degrees.

Pouring Liquids

Liquids provide a unique challenge due to the agent's limited control over them. Other than solid objects, the agent can not grab a liquid and manipulate it. However, the agent can control the holding container, thus it can indirectly manipulate the liquid. Once the liquid leaves the container, it is no longer subject to the agent's control. The corresponding task involves pouring liquids from one container into a different one. Due to the missing native support of fluid dynamics in MuJoCo and PyBullet, we approximated the liquid under the use of spheres. Also, we did not use liquid for motion tracking and did not include liquid in virtual reality, as water cannot be tracked through OptiTrack. However, users should understand how liquids will move depending on their own movements. The reward function is a distance function between the spheres and target container, accompanied by a negative reward if the controlled container is put inside the target container. Additional rewards are given when the spheres collide with the target container, while a sphere landing outside the target container reduces the reward.

Insert card in mailbox

Inserting objects can be a rather simple task, as the required translation is straightforward and only the correct orientation needs to be found. The challenge lies in the precision required to solve the task. As the height of the mail slot decreases, the card must be aligned with increased accuracy, leaving less tolerance for the card's orientation. This task is a positioning task, with the reward function defined by the distance of the card to the inside of the mailbox. Corresponding to the generic approach, the card's orientation is not part of the reward function as it is not relevant once the card has entered the box.

Balancing objects on a tray

Balancing is a common example, often in the form of the cart pole environment. While it is usually not too challenging a task, it was reported that a previous student project [9] encountered difficulties with this task. To specify, when executing in virtual reality, the lack of haptic feedback seemed to cause issues, therefore, we decided to solely test it in reinforcement learning. In reinforcement learning, the reward function is described by the differences between the balanced state and the current state of the balanced objects (i.e. rotation and position), supported by the collision between the cubes.

Puzzle assembly

The focus here is dealing with a high number of degrees of freedom—rather than to solve the puzzle—especially when not every action affects all involved parts. This task is exclusive to reinforcement learning. The reward function is an accumulation of the distances and differences in rotation between the individual pieces and their target positions and rotations. The reward function is mostly punishing in behaviour, always giving negative reward which is reduced for productive actions (i.e. maximum reward is 0).

3.2. OpenAI

Due to the limitless amount of affordances, it would be unfeasible to define and train every affordance individually. Instead, we tried to keep the reward functions as general as possible, only using the successful end-state of the affordance (i.e. the pot is covered or the card is inside the box) to simulate a generic approach to affordance learning.)

There were a few more affordances taken into consideration for the experiments, however, most were dropped due to limited resources and time limitations on implementation. These affordances include: Stirring Objects in a Pot, Cutting a String, Cutting a Fruit and Wrapping a Chain around a Horizontal Pole. The chain affordance proved to be quite troublesome, in particular, as the handling of the chain by the physics engine turned out to be rather unreliable in both MuJoCo and PyBullet. While the chain by itself could be handled somewhat sufficiently, as soon as the agent gained control over any part of the chain, the behaviour became unpredictable and the joint connection seemed to be ineffective.

When comparing the two physics engines, MuJoCo had a lot of tunnelling between objects (i.e. the toast would simply drop through the pan). Using internal primitives instead of external objects solved these issues; although, in many cases, it came at the cost of precision. Furthermore, even small changes to mass seemed to have an unnaturally strong effect on the interaction between objects. While tunnelling could also be seen in PyBullet, it was only encountered at rather large simulation step sizes, which is expected. All-in-all, we found that PyBullet was a lot more stable and predictable, as well as more accessible thanks to more extensive documentation. Unfortunately, this stability came at the cost of speed, as the training took about 2-3 times as long when using PyBullet instead of MuJoCo. This resulted in PyBullet not making it into the final training of the agents, due to time limitations. Thus, all the results presented in chapter 4 have been achieved under the use of MuJoCo.

3.3. Virtual Environment

To compare the execution of affordances between virtual reality and the motion tracking, we conducted a study in which the participants would execute the selected affordances with different tool objects in both environments. The study consisted of 7 users and each user executed the affordance 3 times, with different tool objects. Afterwards, the trials for each user per environment and affordance have been aggregated and compared by use of the dynamic time warp algorithm, invented by Vaughan and Gabrys. [12] The DTW algorithm is particularly useful, as it is insensitive to different sampling rates and minor differences in distance. It generates an idealised curve based on two sources and provides a distance value to indicate the similarity between the trajectory shapes. In this paper, we call this the DTW distance.

Before the data can be processed, it needs to be collected. For this purpose, we used the SemLog plugin for the Unreal Engine, which provides a collection of logger applications to record the state of any scene at runtime and writes the results to either a MongoDB or NoSQL database.

While the SemLog plugin provides a variety of loggers with different abstraction levels, we only made use of the world state logger which records the location and rotation as a 3D vector and a quaternion. The results can then be exported to a csv or other file format for further processing.

Moreover, the integrated Take recorder functionality of the Unreal Engine enables the visual recording of the scene at runtime. The recordings can afterwards be replayed, including multiple at once. This type of visualisation is very helpful to provide a reference frame for the numerical DTW distances. While a DTW distance of 0 shows that the trajectories are identical, the numerical values by themselves are mostly descriptive in relation to each other. The take recordings provide the means to not just identify differences between trajectories, but also to identify whether the whole set of trajectory pairs is generally similar or rather different.

While recording to gather data with SemLog, we also used the Take Recorder and screen recorder to capture data and video simultaneously.

After recording each instance successfully, the data required to be preserved for evaluation. However, manually exporting all the data for each affordance was a very time-consuming task. To alleviate this issue, we devised an script file that generates a folder with the given name and automatically exports the most recent recordings into it.

3.3.1 OptiTrack

OptiTrack Motive is a motion capture software developed by OptiTrack, which is used for capturing and analysing the movement of objects or people in 3D space. Motive uses advanced algorithms and image processing techniques to track and record the movement of reflective markers placed on the subject or object being tracked. When combined with Unreal Engine, it offers a solution to create virtual environments that can simulate real-life scenarios with high precision.

For the experiments, we used OptiTrack Motive 2.0.1, which was operated using a combination of cameras including 13 Primex 13 and 4 Primex 41. The experiments were conducted on a computer system comprising an Intel i7-4790 processor with 4 cores clocked at 3.6 GHz, 32 GB of 3600 MHz DDR4 RAM, and a Nvidia Titan V 12 GB graphics card. The system was integrated with Unreal Engine 4.27.

The initial step in utilising the OptiTrack system involves the precise setup of the tracking area by positioning the cameras in appropriate locations and calibrating them to ensure accurate tracking.

This calibration process is critical to achieving reliable and reproducible tracking results.

Through our research, it has been determined that utilising cameras positioned at varying heights around a given scene yields superior tracking results in comparison to utilising cameras positioned at the same height. See

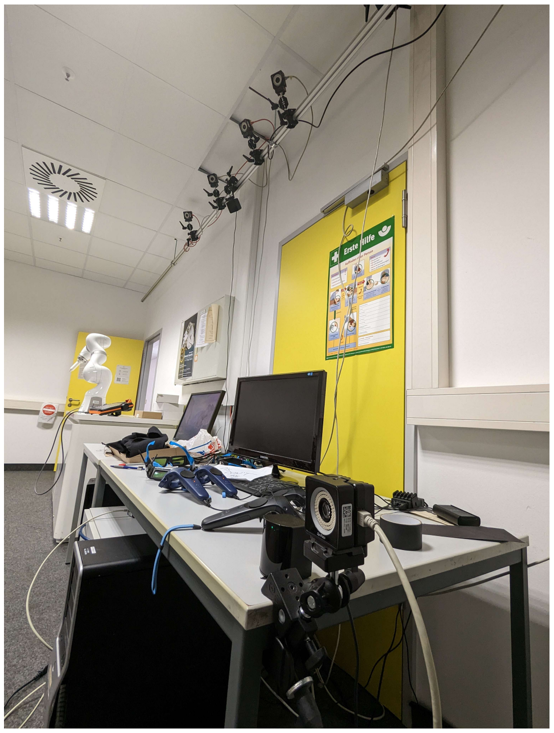


Figure 1: *The camera setup used for the motion tracking, highlight the height differences for better tracking performance.*

figure 1. However, it should be noted that as long as the markers are consistently visible to the cameras, increasing the number of cameras may not necessarily lead to improved tracking performance.

OptiTrack employs reflective markers for tracking movement, which are attached to specific points on the object. The accuracy and quality of data recorded during motion capture rely heavily on the proper preparation of the object or individual, which includes the precise attachment of markers. In addition to markers, rigid body sticks are used to track the movement of larger, more complex objects by tracking multiple points simultaneously, while markers are single-point objects used to track the movement of individual points.

One of the initial issues encountered in the OptiTrack system was the random occurrence of unrecognised markers. We utilised black tape and paint to cover all the objects

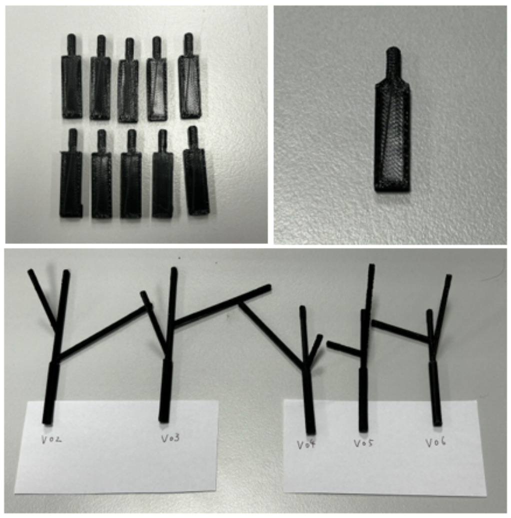


Figure 2: *Rigid Body Sticks developed with 3D printing technology, individual markers (top), and five unique patterns (bottom).*

in the scene, regardless of their reflexibility, and this technique reduced the amount of light reflected by the objects and minimised interference caused by reflections, resulting in improved overall performance and more precise motion tracking outcomes. Initially, we attached the markers directly to the items using various types of tapes in non-uniform positions. However, as a result of this method, the markers tended to move or come off easily when we moved the objects which led to the inaccurate recognition of items. For example, OptiTrack got confused which object is getting tracked or the objects in Unreal that were jittering or not moving at the same time as they should. Also, as we introduced more objects to track, we have found that using numerous markers for multiple objects in a single scene can lead to confusion in defining objects and how they are placed due to overlapping patterns, resulting in inaccurate motion tracking. Conversely, using fewer markers can be problematic for OptiTrack when objects are moving since it makes it difficult for the cameras to track at least 3 markers, which is the absolute minimum markers to recognise the object at the same time. To address this issue, we developed a solution using 3D printing technology. Specifically, we created 3D rigid body sticks with six unique patterns and screws for individual markers, which can be seen in figure 2. The unique patterns and closer proximity of the markers made the tracked objects' patterns more distinct and reduced spatial overlap of the markers, resulting in more stable and accurate performance. We were also able to disattach markers to reuse them for different items without worrying about dislocating the markers' positions by being able to tape the screws or the rigid body sticks, instead of taping individual markers directly. Moreover, by using a combination of rigid body sticks and individual markers, we were able to achieve significant improvements in tracking accuracy for multiple objects in a scene by creating unique marking patterns for each. Our approach provides a practical solution for enhancing motion tracking accuracy in complex multi-object scenarios.

3.3.2 Virtual Reality (VR)

The virtual reality setup is rather simple as it involves mainly the recreation of the motion tracking scene in the Unreal Engine. We experimented with a VR headset, the HTC Vive Pro Eye with its native controllers. To ensure that the environments of virtual reality scenes and OptiTrack scenes properly relate to each other, the scene's dimensions were measured beforehand and applied to the virtual scene. Likewise, the objects were modelled after the measurements of their real world counterparts. One major issue with the involved objects is that most of them were non-convex, as this property is quite common with household objects. Unfortunately, the PhysX Engine can, just like MuJoCo and PyBullet, only handle convex objects. Since a pure convex hull would not be accurate enough for many of our objects, convex decomposition was necessary. At first, we tried the V-HACD algorithm which generated too many colliders, resulting in bad performance. As a solution, we had to decompose the objects manually in blender, which resulted

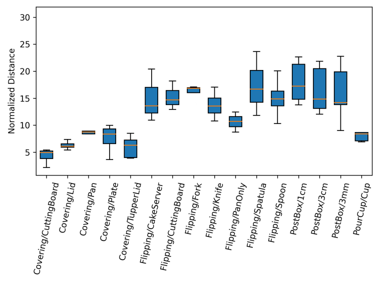


Figure 3: *The plot visualizes the mean DTW distances and standard deviation for all affordances. The orange lines represent the median. While some variation is visible the overall DTW distances are considered small.*

in more uniform colliders, improving not only the performance, but also the collision detection in the process. Finally, it had to be ensured that the starting position of the involved objects were consistent with the OptiTrack setup, which was achieved by marking key positions in the OptiTrack setup with tape—so that they can easily be recreated. Only basic interaction for the used virtual reality headset and controllers need to be implemented and accurately calibrated, and as long as these steps are followed, both environments are comparative.

4. Results

4.1. Virtual Environment

Figure 3 shows the mean DTW distance and standard deviation for various tested affordances and objects. The lower values indicate more similarity. When the recordings of the trajectories are taken into account, the DTW distances can be considered rather small overall. More detailed view can be found in Table 2 in the Appendix.

Regarding the affordances, we can make the following general observations based on the results of DTW analysis. For covering, the mean DTW distances for the Lid, Tupper-lid, and Cutting board are relatively low, whereas the DTW distances for the Plate and Pan are slightly higher. The standard deviation values for all objects are relatively small, indicating consistent performance across trials. We can also see that the standard deviation for pouring is relatively high compared to the other affordances.

It is notable that the DTW distances for the card insertion and flipping affordances are comparatively high, however, the standard deviation increases proportionally, indicating that the change is consistent across all users.

4.2. Reinforcement Learned Systems

Table 1 shows the learning results of the trained reinforcement learning agents in three different levels: successful, converging and unsuccessful. Successful agents converged and managed to execute the affordance in a sufficient manner. Converging agents made a clear learning process but failed to fully execute the affordance: we will discuss the reasons for that in section 5. Finally, unsuccessful agents did not show any learning progress at all, and conversely were not successful.

More detailed information can be found in the appendix. The appendix shows the reward per episode for all affordances, for both DQN and PPO. The DQN algorithm shows two curves called 50 decayrate and 80 decayrate. The decayrate determines how fast the agent would move from exploration to exploitation, starting at 95% exploration (i.e. 95% of actions are selected randomly) to 5% exploration at the end. A decayrate of 50 indicates that the 5% mark is reached after 50% of the training time, whereas a decayrate of 80 reaches the 5% mark after 80% of the training. There is no noticeable difference between the two decayrates.

While the pouring liquid tasks, the affordance shows the 50 decayrate as overall better performing, it should be noted that it performed better from the start while it was still in the exploration phase. The learning rate is not significantly better also.

Other notable features are that the DQN agent for the pot covering affordance shows more variation later in training, and the DQN agent for the flipping affordance even decreased in reward significantly. Lastly, the

Affordance	DQN 50 Decayrate	DQN 80 Decayrate	PPO
Covering a Pot	Unsuccessful	Unsuccessful	Successful
Covering a Pot - no collision	Successful	Successful	-
Flipping Object in Pan	Unsuccessful	Unsuccessful	Unsuccessful
Pouring Liquids	Converging	Converging	Successful
Insert Card in Mailbox	Converging	Converging	Converging
Balancing Objects on Tray	Unsuccessful	Unsuccessful	Unsuccessful
Puzzle Assembly	Unsuccessful	Unsuccessful	Converging

Table 1: *The table shows the success rate of the different agents with the different algorithms in three levels: Unsuccessful, converging but not successful and successful.*

DQN agents for the card insertion affordance showed some unusual drops in the second half of the training. Generally all of the DQN agents were unsuccessful with the only exception of a special case for covering a pot, where the collision criteria was taken out of the reward function.

All agents were trained in the MuJoCo environment.

5. Discussion

5.1. Virtual Environment

As described in chapter 4, we observed a rather high standard deviation in the pouring affordance, while the mean DTW distance being not particularly high. This indicates a lot of variation between users, which is most likely the result of the fact that cups are items which have variation in the way people hold them. At the same time, many people report issues with the precision of the grabbing functionality in VR as it was based on a button press and did not mimic real world finger motions. These factors combined explain the high variation among users for this affordance.

Furthermore, the card insertion and flipping affordances show a comparatively high DTW distance with a proportional change in standard deviation. In both affordances, we observed users struggling with the lack of haptic feedback and imprecise collision detection in VR. In the case of the card insertion task, users would accidentally move the box when missing the mail slot or simply choose to turn the box on its back for better access to the mail slot. The box could have been made a static object and block the postcard when the user misses the mail slot, thus avoiding a movement of the box. However, this would introduce a dissonance between the users actual movement and its virtual projection—trading one inaccuracy for another. In the case of the flipping affordance, there is one variant with an unusually low DTW distance among the variations in the form of the "pan only" variation. This difference indicates that the main issue for the users resulted, again, from the lack of haptic feedback and collision detection between objects, as the difference between VR and Optitrack decreased once the tool object was removed. Finally, the covering affordances show relatively low DTW distances, which is not surprising since they do not require interaction between objects while the user is still in control. Another issue that was encountered in VR is object movement introduced by the physics engines.

Due to the lack of haptic feedback, objects would sometimes be forced into other objects, causing the physics engine to be overwhelmed and fling objects through the scene. An example of such behaviour can be seen in figure 4. This additional movement can be filtered out as long as it was caused after the affordance has been executed, as we did for our results manually. However, if this behaviour were to occur during the main execution part of the affordance, it would make the data unusable for further processing; even if it does occur after the execution, filtering is required for further

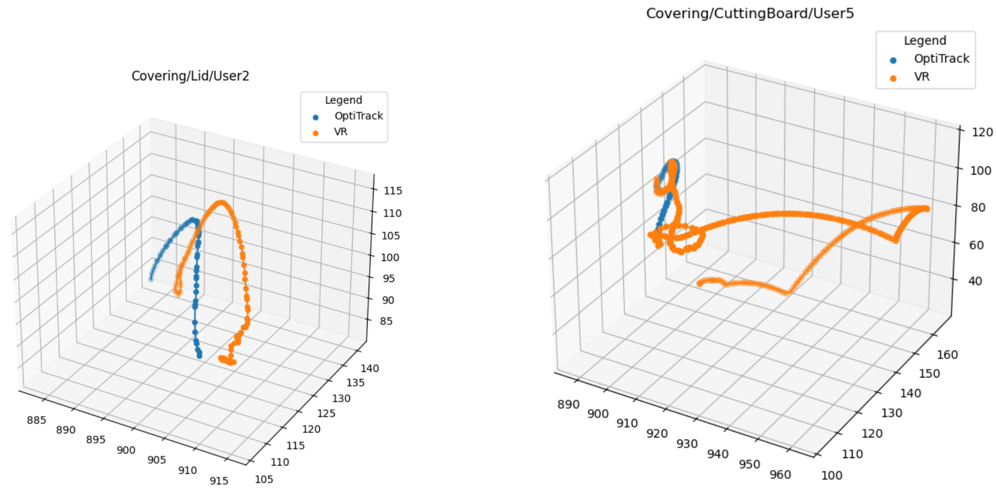


Figure 4: *The images show the VR (orange) and Optitrack (blue) trajectories for the covering affordance in comparison. On the left is stable example with the lid as object and on the right an example showing a case where the tool object, in this case a cutting board was thrown away by the physics engine.*

usage.

Despite the fact that the motion tracking setup was able to avoid collision issues due to the laws of nature, it still faced its own challenges. Most of these challenges have already been described in chapter 3, such as the problems with reflection and the visibility of markers. Another issue is that some affordance simply can not be executed completely when the objects involved have volumetric markers attached to them, as these may be too big to pass through openings or may make proper collision between objects impossible.

Overall, we concluded that virtual environments can theoretically simulate the real world in a sufficient manner. However, special care is to be taken as to what form the input takes; additionally, the recorded data needs to be validated and potentially processed before it can be used in the training of reinforcement learning agents or similar applications.

5.2. Reinforcement Learning

In this section, we will further discuss the results and also take into account agent behaviour that can not be shown in graphs or individual images.

We start with the consistently successful affordances, which include: Flipping object in a pan, Balancing objects on a tray, and Puzzle assembly, which can be found in Figure 8, 11 and 12 respectively.

The reward of the DQN agents does not converge in either the balancing or assembly task, and, while the PPO agent converges in the assembly task at least, the resulting behaviour was that the puzzle pieces would simply be thrown away from each other and not assembled.

The reasoning behind why the agent converged to this solution is not entirely clear, it might be that the calculation for comparing the current state to the desired state of all objects contains a mathematical glitch, which causes the reward to be better at very extreme states compared to the randomised starting states. Similarly, misleading is the tendency line of the PPO agent for balancing objects on a tray, since the curve overall shows no persistent improvement and never passes a reward threshold of about 1200. The actual behaviour of both the DQN and PPO agents mainly consisted of keeping the tray moving and to delay the collapse of the cubes, not showing any signs of actually counteracting gravity. The failure in both tasks can be explained by the environment having too many parameters. The puzzle assembly has 54 degrees of freedom (9 Objects, Location and Rotation) all of which can be controlled and, while the balancing affordances has only 30 degrees of freedom (DoF) of the cubes, those 30 are controlled by a further 6 degrees from the tray and also subject to gravity. The same can not be argued for the flipping task, as it only involves controllable 6 DoF for the spatula and a further 6 for the Toast, with the 6 DoF of the pan not being part of either the reward function or the agent's inputs, thus not having any influence at all.

Yet the DQN agents did not only not converge, but even decreased in performance after the first 200 episodes. The PPO agent, on the other hand, shows a slight tendency for improvement, which is mostly caused by sporadic peaks in reward, and are more likely the result of chance rather than actual learning—as no convergence is visible. It may be possible for the random good episodes to accumulate enough so that the PPO agent would be able to successfully converge; however, this would require an unreasonably large amount of training. The difficulties in the flipping tasks can be explained by the discrepancy between the available input actions and the reward function, which does not provide any clear feedback to the agent for most of its inputs—since only a very few input sequences will even affect the toast. This is supported by the resulting agent behaviour for both DQN and PPO, which did not show any sensible pattern, either just moving the spatula to infinity or simply dropping it—choosing not to do anything. To fulfil the task, the agent would need to identify the instrumental goal of touching the toast with the spatula.

A similar issue can be found in the card insertion task, which is shown in Figure 10. This time, all agents converge clearly but fail to actually insert the card into the box; instead the card just gets slapped to the outside of the box trying to go ever closer. Compared to the flipping affordance, the reward function in this case and the available input action both relate directly to the card, providing clear feedback for any given input, thus the clear convergence is not surprising.

Similarly to the flipping task, the agents fail to identify a required instrumental goal. The card first needs to be oriented correctly and then moved through the opening of the box—a property that is not reflected in the reward function—corresponding to the premise of keeping it as generic as possible. One additional feature to be noted is that the DQN agents have some very bad episodes even after they mostly converged. This is most likely the result of the agent using the general motion of moving the card closer to the box but slightly missing it and moving further away from the box as the motion continues, gathering negative reward.

Next, we have the pouring liquids affordance, of which the results can be found in figure 9.

All of the agents converge in this task, however, only the PPO agent does so successfully. The DQN agent also converges very slowly and tends to mainly rotate the cup violently over the pot before discarding it, resulting in some of the spheres to land in the pot and other's outside. The main issue seems to be in dealing with multiple reward criteria, as the reward function uses the sphere-pot distance as a base and provides a bonus for any sphere inside the pot, while giving negative reward for the spheres landing outside the pot and if the cup is inside the pot. While DQN seems to be particularly sensitive to this issue, it is also reflected in the PPO agent. The PPO agent manages to successfully put all the spheres in the pot, but it does so by putting the whole cup in the pot and then emptying it, before throwing it away. Despite the negative reward it gets for inserting the cup into the pot, this solution seems the easiest to get consistently good rewards. The fact that the cup is thrown away afterwards is simply a result of the cup's behaviour not being regulated by the reward function once the spheres are in the pot. Lastly, it should be noted that the seemingly better performance of the 50 decayrate DQN agent is most likely a result of chance, as it already starts a lot better early in the exploration phase and the actual behaviour did not change any noticeable difference in performance.

Finally, we have the affordance of covering a pot with a lid, with the results visualised in figure 6. The previously mentioned difficulty of dealing with multiple reward criteria can again be seen in the DQN agents for this task.

While its reward continues to improve, it fails to converge and interestingly, it also shows increasing variation in reward, especially for the 50 decayrate agent. This change in variance can be explained by the fact that the collision criteria only has an effect on the reward later during the task, once collision actually occurs. As a result, the agent starts to learn to satisfy the distance criteria but fails to then actually put the lid on the pot.

This is supported by the fact that the spread is less pronounced in the 80 decayrate agent as it spends more time exploring and, therefore, is more adjustable in its learned behaviour for a longer period of training. It is also reflected in the actual behaviour of the agents as they tend to move the lid towards the pot but then crash into it. The PPO agent, on the other hand, does not only converge well, it also solves the task perfectly—since, like with the pouring task, PPO is more capable of dealing with multiple reward criteria, with the task being quite simple. The simplicity is due to a rather direct motion requirement, a clear relation between reward function and input as well as only 6 DoF involved. However, the covering affordance has an additional two special agents trained, of which the results can be seen in figure 7. These special agents use only the positioning criteria of the reward function, but they are fully equal to the other DQN agents in every other aspect. While it does not finish the task by actually connecting the lid with the pot, it is successful in positioning it and does so after just 300 episodes of training. Additionally, most of the learning is reached after an incredibly low 50 episodes, even vastly outperforming the PPO agent in regards to convergence rate—despite PPO generally converging faster than DQN. These special agents show how simplification can have a very strong impact on performance and training time—even potentially better than just using a different algorithm. Of course different algorithms and simplification are not exclusive to each other and can be combined for further effect.

Based on these observations we identify the following three key issues:

1. Instrumental goals can not be derived purely from the affordance's fulfilment state.

2. Too many criteria in a single reward function can confuse the agent.

3. Too many parameters involved at once can overwhelm the agent.

As shown with the specialised agent for the covering task, and in a similar manner by the works of Khimya et. al. [6] , we argue that all of these issues can be solved or reduced by simplifying the task to be learned. More specifically, we propose that an affordance should be split into multiple components, which can be kept rather simple and can be trained faster as multiple agents, compared to a single agent being trained on the sum of the components. These component agents would require an additional managing agent that is trained to identify the components involved and delegate work to the corresponding component agents. Such a decomposition would also provide more flexibility and would require less re-training of similar—or even rather different affordances—as smaller components generalise better and can be easily reused, a property quite essential when considering the near uncountable amount of possible affordances.

To support further research on this concept, we propose a taxonomy for the aforementioned components, which is described in the following section.

5.3. Taxonomy

Based on our observations, we propose the taxonomy for components, which are required to describe the physical motions to fulfil an affordance in figure 5. An affordance can have one or multiple of these components, which may be fulfilled at the same time or sequentially. While not all classes of components could be included in the experiments, we evaluated the taxonomy by referencing a list of over hundred affordances and testing whether we were able to describe the affordances based on this taxonomy. We did not find any exceptions to the taxonomy, therefore, we consider it to be sufficiently complete for future use, though further verification is recommended.

5.3.1 Positioning

As the name implies, positioning components describe the need of an object to be placed in a certain position. There are two major subclasses, Space-based and Collision-based, which—in an instanced context—can theoretically be converted to each other if the environment is well known and fluids (including air) are taken into consideration for collisions (i.e. placing a balloon in a specific position can either be described via a target volume or a collision with specific parts of the airspace). However, the practicability of this conversion is highly dependent on the model used to describe the environment and may in many cases lead to unnecessary restrictions, resulting in worse performance.

- **Space based:** In space-based components, we only care about the position itself, not so much about the functional connection between objects. A few examples would be moving a piece of a board game onto a specific field. This class can be further divided into the Translational and Rotational subclasses. Depending on the situation, many components will involve both of these classes; in these cases, their reward criteria can simply be combined, potentially at different steps in the solving process. An example of such a combined task would be the insertion of an object into a box, such as a postcard into a mailbox or a toast in the toaster. In both cases, it is unlikely that the object is already aligned such that it can be directly inserted, requiring rotation. Furthermore, the actual end-state only requires the object to be in the target space defined by the box.
  - **Translational:** This subclass describes tasks in which the orientation of the object is either irrelevant or already satisfactory. Therefore, only a translation of the position needs to be applied to fulfil the affordance. The aforementioned board game piece would describe such a component, as the piece is already standing up as intended most of the time. The evaluation criteria for this component can be described as the inverse of the distance between object and target position, and a complete fulfilment after a certain threshold has been passed or the object is fully contained in the target space volume.
  - **Rotational:** As the counterpart to the translational subclass, components in this subclass only care about the rotation of an object and not the location. This could be because the location is fixed anyway. Turning a dial on a stove or similar would be described by such a component.
- **Collision based:** Contrary to the positioning components, collision based components do not care about the position itself, but rather the contact between two objects. While a location and rotation can be derived from a collision, there may be various valid combinations of values, all being satisfactory. For example, cooking a piece of toast in a pan requires one side of the toast to be in contact with the functional part of the pan—the specific location in the pan and also the rotation around the up-axis do not matter for the cooking process.

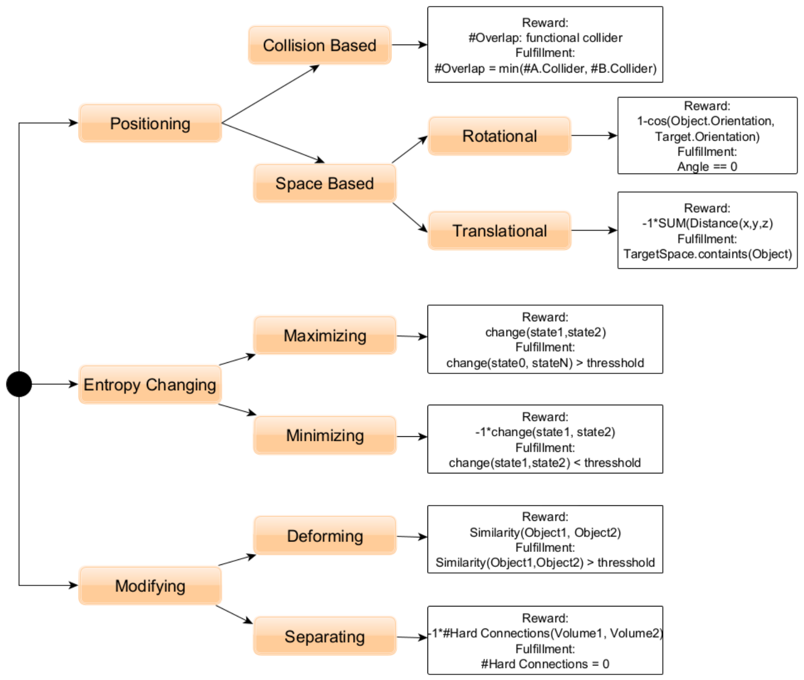

Collision Based
Reward:
#Overlap: functional collider
Fulfillment:
#Overlap = min(#A.Collider, #B.Collider)
Positioning
Rotational
Reward:
1-cos(Object.Orientation, Target.Orientation)
Fulfillment:
Angle == 0
Space Based
Translational
Reward:
-1*SUM(Distance(x,y,z)
Fulfillment:
TargetSpace.containts(Object)
Maximizing
Reward:
change(state1,state2)
Fulfillment:
change(state0, stateN) > thresshold
Entropy Changing
Minimizing
Reward:
-1*change(state1, state2)
Fulfillment:
change(state1,state2) < thresshold
Deforming
Reward:
Similarity(Object1, Object2)
Fulfillment:
Similarity(Object1,Object2) > thresshold
Modifying
Separating
Reward:
-1*#Hard Connections(Volume1, Volume2)
Fulfillment:
#Hard Connections = 0

Figure 5: *A taxonomy of affordance components, with proposed evaluation criteria for each class. The evaluation criteria of a parent class can be described as the combination of the criteria of all sub classes.*

Additionally, a space based description would also require taking the target object into account, as a smaller object occupies a smaller space and is more restrictive in the target space to fulfil the collision requirements. The evaluation criteria for this class can be described as the overlap between the functional collision areas of both objects involved. The maximum reward can be derived from the smaller functional area of both objects.

5.3.2 Entropy changing

As the name implies, these components require the entropy of observed values to be adjusted, either over time (change between steps) or over space (entropy for the observed space at a specific point in time). Entropy can either be minimised or maximised.

- **Minimising:** An example of a minimising task would be any balancing task. When balancing, the goal is to reduce change between angles and potentially positions over time. However, balancing usually also involves other components, such as positioning, as the minimal entropy is usually supposed to be reached ,while the objects are stacked or similar and not lying on the ground. The evaluation criteria would be the inverse to the entropy.
- **Maximising:** A maximising task would, for example, be the stirring of cooking ingredients in a pot or shuffling of playing cards. In both cases, the goal is to mix up the objects as much as possible. In the case of the cooking pot, the result would describe the most even heat distribution, which is a change over time. Therefore, with a different perspective, the stirring can also be described as minimising entropy over space if the heat distribution is measured directly. In the case of the playing cards, the maximum amount of entropy over space described the most fair shuffling for a game. Analogue to the minimising component, the evaluation criteria

in this case would be the entropy itself.

5.3.3 Modifying

The modifying class is the most complex one to evaluate. They fall into two major subclasses; Separating and Deforming.

- **Separating:** Separating components describe tasks in which an object needs to be split into two or more sub objects. A typical task of this class would be the cutting of fruit or other objects. For evaluation, it is required to identify the various volumes which need to be separated. Once these volumes have been identified, the evaluation can be described as the inverse of the amount of physical connections between all identified volumes.
- **Deforming:** The deforming class can theoretically be subdivided into the manipulation of the object by itself (i.e. deforming a chunk of clay) and adding additional things to the object (i.e. adding more clay to the chunk). Nonetheless, the evaluation criteria in both cases would be the similarity between the current state of the object and the desired reference state.

6. Conclusion

We conclude that affordances can prove to be rather complex situations, which can be quite challenging for reinforcement learning when tackling them as a whole. However, splitting them into simpler components can provide an efficient solution to this problem. To support research in this field we propose a taxonomy, which describes the physical motion components that may be encountered in an affordance, along with potential reward functions to train agents for these components. However, evaluating the quality of a reward function without empirical testing is, at the point of writing, basically impossible. Thus, further testing and validation of the taxonomy is required, for both the included reward functions as well as the component classes themselves, regarding the completeness and accuracy. Lastly, we consider virtual environments to be generally suited to simulate real world affordances, but more emphasis on faster and more stable collision detection needs to be made and special care taken when user data is to be used, for imitation learning or similar techniques.

References

[1] E. Şahin, M. Çakmak, M. R. Doğar, E. Uğur, and G. Üçoluk, "To afford or not to afford: A new formalization of affordances toward affordance-based robot control," *Adaptive Behavior*, vol. 15, no. 4, 2007.

[2] P. Zech, S. Haller, S. R. Lakani, B. Ridge, E. Ugur, and J. Piater, "Computational models of affordance in robotics: A taxonomy and systematic classification," *Adaptive Behavior*, vol. 25, no. 5, 2017.

[3] A. Nguyen, D. Kanoulas, D. G. Caldwell, and N. G. Tsagarakis, "Object-based affordances detection with convolutional neural networks and dense conditional random fields," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[4] M. A. Zamani, S. Magg, C. Weber, S. Wermter, and D. Fu, "Deep reinforcement learning using compositional representations for performing instructions," *Paladyn, Journal of Behavioral Robotics*, vol. 9, no. 1, 2018.

[5] R. Bhattacharyya and S. M. Hazarika, "Object affordance driven inverse reinforcement learning through conceptual abstraction and advice," *Paladyn, Journal of Behavioral Robotics*, vol. 9, no. 1, 2018.

[6] K. Khetarpal, Z. Ahmed, G. Comanici, D. Abel, and D. Precup, *What can i do here? a theory of affordances in reinforcement learning*, 2020.

[7] H. Wu, D. Misra, and G. S. Chirikjian, "Is that a chair? imagining affordances using simulations of an articulated human body," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 7240–7246.

[8] V. Kumar and E. Todorov, "Mujoco haptix: A virtual reality system for hand manipulation," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 2015, pp. 657–663.

[9] F. Busch, C. Heathcote, R. Jakob, B. Tegetmeier, and A. Vakili, "Vr-based human-robot-affordance transfer," 2022. [Online]. Available: `https://www.cgvr.cs.uni-bremen.de/teaching/studentprojects/vrrat` (visited on 02/24/2023).

[10] F. Ostiategui, A. Amundarain, A. Lozano-Rodero, and L. Matey, "Gardening work simulation tool in virtual reality for disabled people tutorial," *Proceedings of Integrated Design and Manufacturing-Virtual Concept (IDMME'10)*, 2010.

[11] A. Raikwar, N. D'Souza, C. Rogers, *et al.*, "Cubevr: Digital affordances for architecture undergraduate education using virtual reality," in *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2019, pp. 1623–1626.

[12] N. Vaughan and B. Gabrys, "Comparing and combining time series trajectories using dynamic time warping," *Procedia Computer Science*, vol. 96, 2016. DOI: `https://doi.org/10.1016/j.procs.2016.08.106`.

[13] F. Pardo, A. Tavakoli, V. Levdik, and P. Kormushev, "Time limits in reinforcement learning," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80, Jul. 2018.

[14] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: A survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*.

[15] S. Munikoti, D. Agarwal, L. Das, M. Halappanavar, and B. Natarajan, *Challenges and opportunities in deep reinforcement learning with graph neural networks: A comprehensive review of algorithms and applications*, 2022.

[16] R. Bhattacharyya and S. Hazarika, "Object affordance driven inverse reinforcement learning through conceptual abstraction and advice," *Paladyn, Journal of Behavioral Robotics*, vol. 9, Sep. 2018.

[17] Y.-C. Liao, K. Todi, A. Acharya, A. Keurulainen, A. Howes, and A. Oulasvirta, "Rediscovering affordance: A reinforcement learning perspective," in *CHI Conference on Human Factors in Computing Systems*, Apr. 2022.

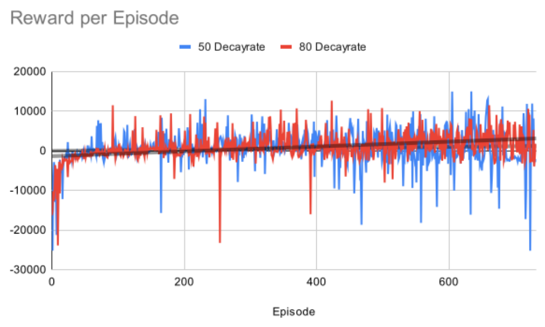
[18] R. Traoré, H. Caselles-Dupré, T. Lesort, T. Sun, N. Díaz-Rodríguez, and D. Filliat, *Continual reinforcement learning deployed in real-life using policy distillation and sim2real transfer*, 2019.

[19] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, *Overcoming exploration in reinforcement learning with demonstrations*, 2017.

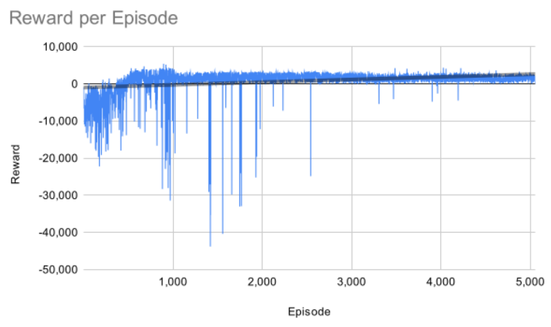
[20] T. C. Stephen Adams and P. A. Beling, "A survey of inverse reinforcement learning," *Artificial Intelligence Review*, vol. 55, 2022.

Appendix

Covering a Pot



(a) *DQN*



(b) *PPO*

Figure 6: *The rewards of per Episode for DQN on the left and PPO on the right for covering a pot.*

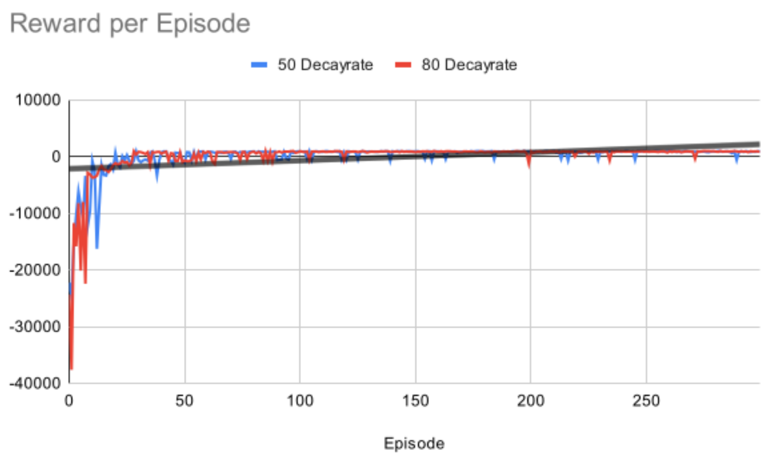
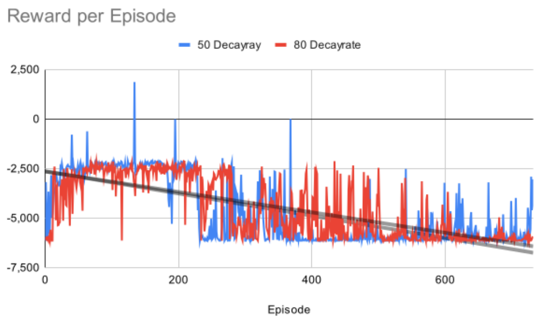
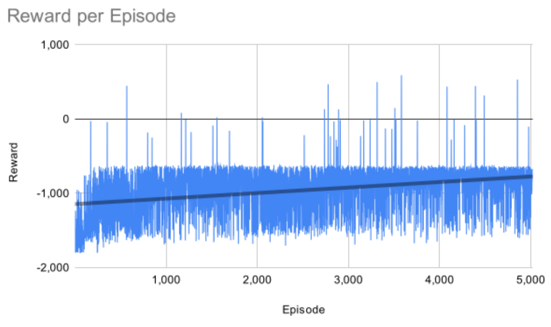


Figure 7: *The rewards of per Episode for DQN when the collision is not taken into account for the reward function.*

Flipping Object in a Pan



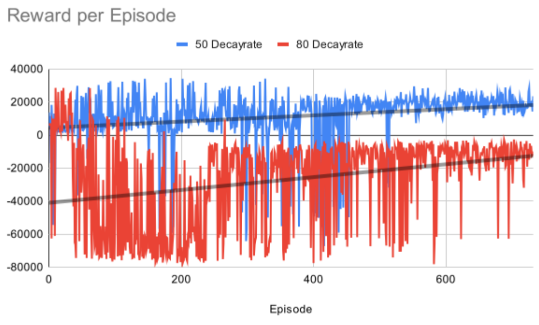
(a) *DQN*



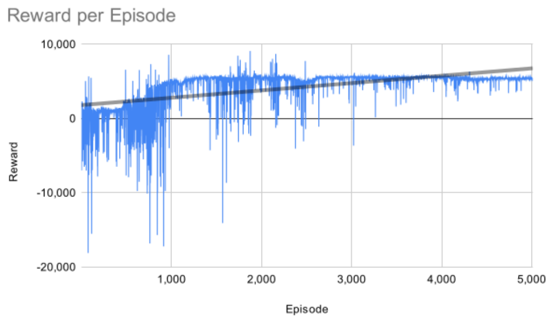
(b) *PPO*

Figure 8: *The rewards of per Episode for DQN on the left and PPO on the right, for flipping an object in a pan.*

Pouring Liquids



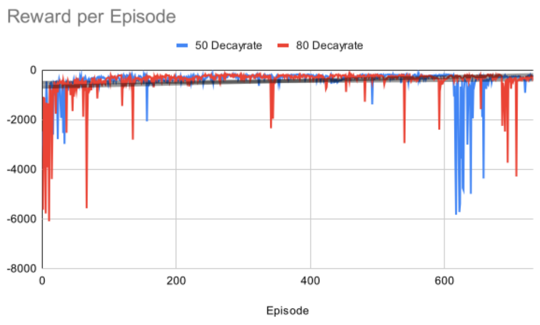
(a) *DQN*



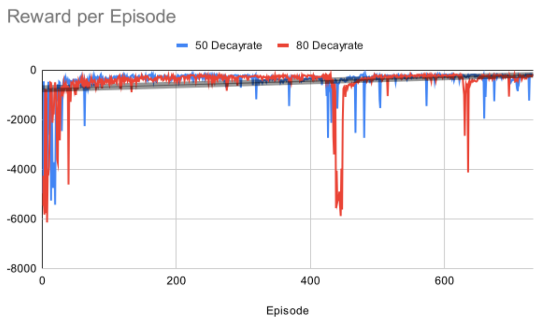
(b) *PPO*

Figure 9: *The rewards of per Episode for DQN on the left and PPO on the right, for pouring liquids into a container.*

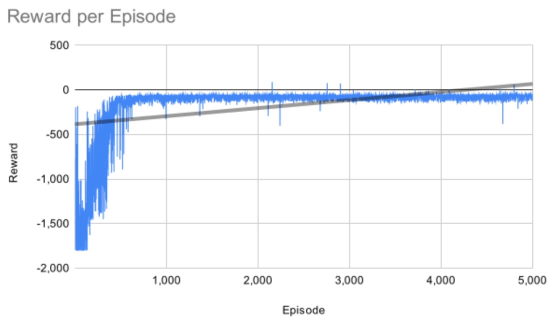
Insert card in mailbox



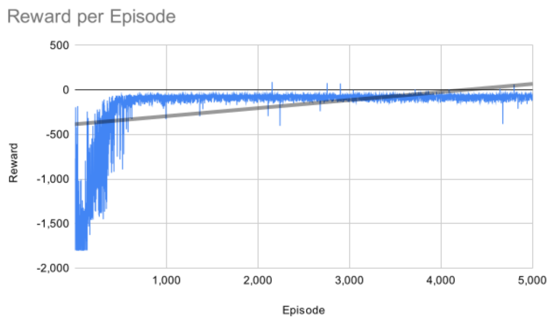
(a) *DQN 3 cm*



(b) *DQN 3 mm*



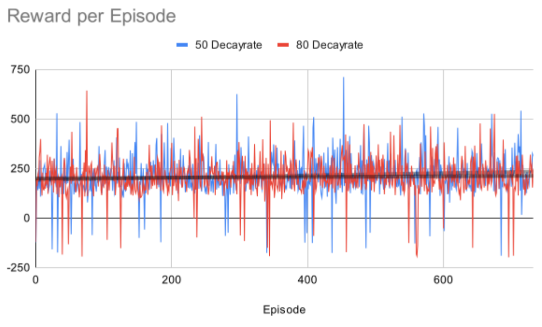
(c) *PPO 3 cm*



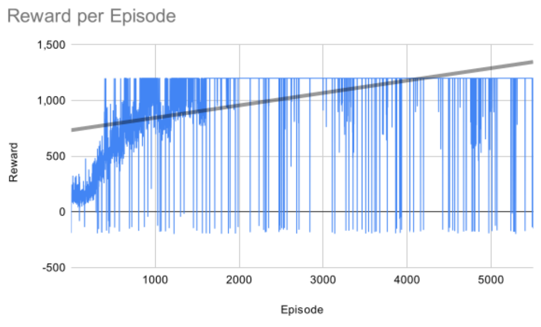
(d) *PPO 3 mm*

Figure 10: *The rewards of per Episode for DQN on the top and PPO on the bottom, inserting a card into a mailbox. The box was tested with opening sizes of 3 cm and 3 mm.*

Balancing objects on a tray



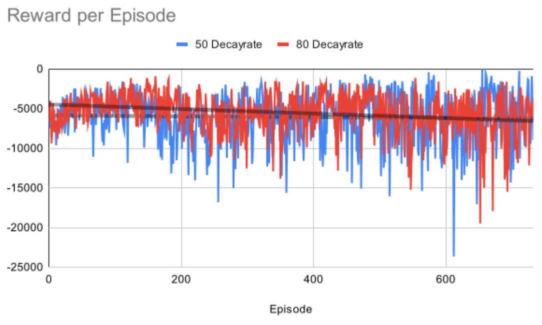
(a) *DQN*



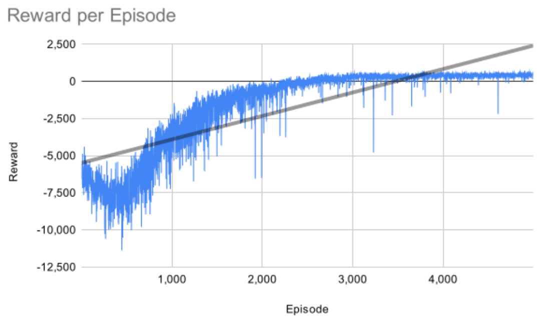
(b) *PPO*

Figure 11: *The rewards of per Episode for DQN on the left and PPO on the right, for balancing 5 cubes on a tray.*

Puzzle assembly



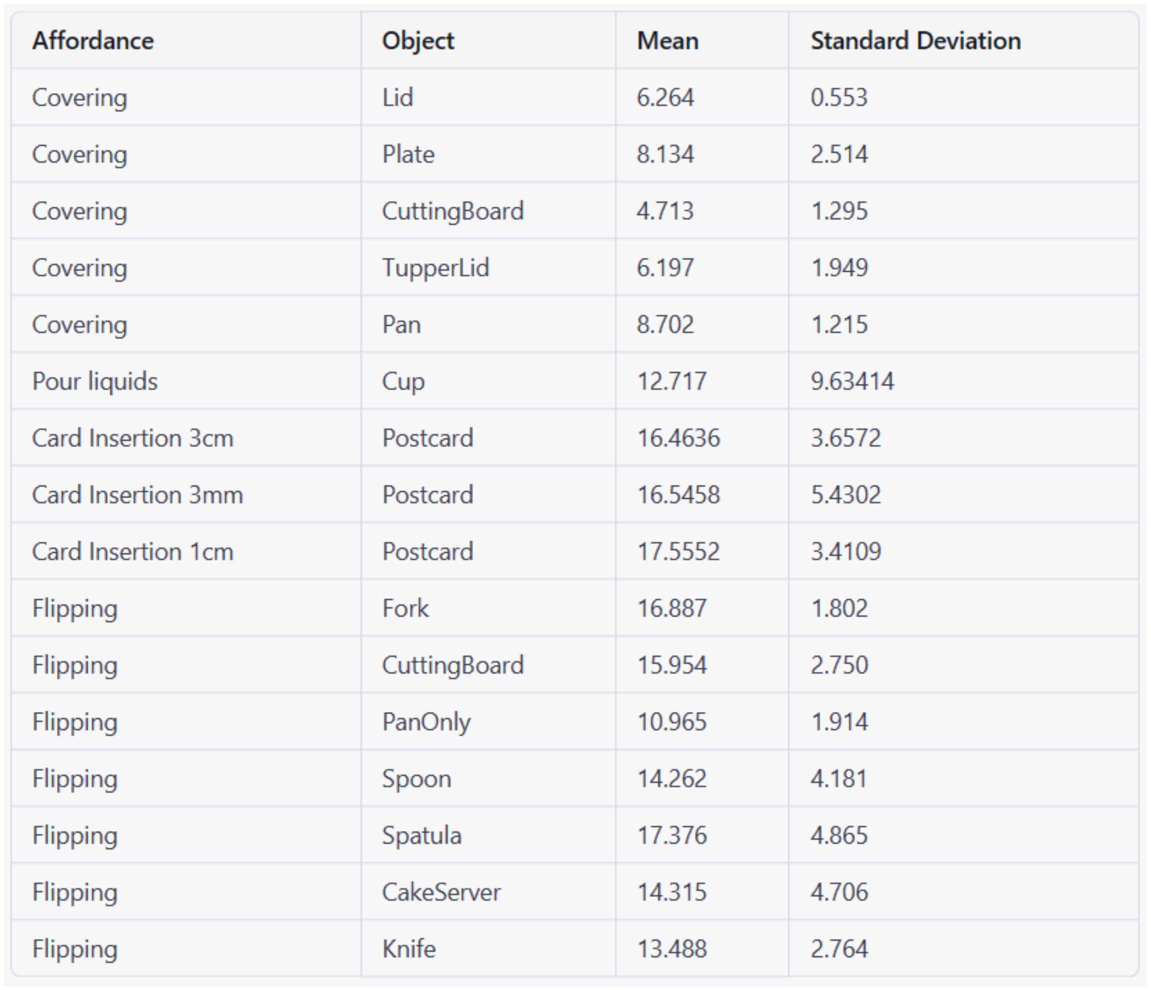
(a) *DQN*



(b) *PPO*

Figure 12: *The rewards of per Episode for DQN on the left and PPO on the right, for assembling a puzzle with 9 pieces.*

Trajectory comparison



Affordance	Object	Mean	Standard Deviation
Covering	Lid	6.264	0.553
Covering	Plate	8.134	2.514
Covering	CuttingBoard	4.713	1.295
Covering	TupperLid	6.197	1.949
Covering	Pan	8.702	1.215
Pour liquids	Cup	12.717	9.63414
Card Insertion 3cm	Postcard	16.4636	3.6572
Card Insertion 3mm	Postcard	16.5458	5.4302
Card Insertion 1cm	Postcard	17.5552	3.4109
Flipping	Fork	16.887	1.802
Flipping	CuttingBoard	15.954	2.750
Flipping	PanOnly	10.965	1.914
Flipping	Spoon	14.262	4.181
Flipping	Spatula	17.376	4.865
Flipping	CakeServer	14.315	4.706
Flipping	Knife	13.488	2.764

Table 2: *The table shows the mean DTW distances and corresponding standard deviation for between the VR and Optitrack trajectories for each tested affordance.*