Chair of Astronautics
Prof. Prof. h.c. Dr. Dr. h.c.
Ulrich Walter

Bachelor's Thesis

Implementation and Evaluation of Reinforcement Learning for Depth Image Enhancement in Orbital Proximity Operations

RT-BA-2021/04

Author:
Daniel Gebhard

Supervisor:	Martin Dziura Chair of Astronautics Technische Universität München

Danksagung

Hiermit möchte ich allen Personen danken, die mich bei der Erstellung dieser Arbeit unterstützt haben.

Beginnen möchte ich mit einem großen Dank an meinen Betreuer Martin Dziura, der mir mit vielen wertvollen Hinweisen zur Seite stand und alle Fragen zügig beantwortete. Seine Erklärungen haben mir sehr dabei geholfen, diese erste wissenschaftliche Arbeit anzufertigen.

Außerdem möchte ich Gerhard Reinerth danken, der mich mit seinem Wissen zu Reinforcement Learning unterstützt hat.

Der Stiftung der Deutschen Wirtschaft möchte ich für die Förderung meines Studiums meinen Dank aussprechen.

Bei meinem Bruder möchte ich mich für seine Unterstützung bei der Korrektur dieser Arbeit bedanken.

Abschließend möchte ich meinen Eltern herzlich danken, die mir das Studium überhaupt erst ermöglicht haben. Mehrfach haben sie mir schon zu Schulzeiten gestattet, besondere Chancen wahrzunehmen, und dabei früh loslassen müssen. Die Selbstlosigkeit, bei diesen Entscheidungen ihre eigenen Interessen hinten anzustellen, kann ich gar nicht genug wertschätzen. Ihr Beistand, ihre Unterstützung und ihr Vertrauen haben mir bei der Erstellung dieser Arbeit sehr geholfen.

Zusammenfassung

Um das Eintreten des Kessler-Syndroms zu verhindern und die operationale Lebenszeit von Satelliten zu verlängern, sind Active Debris Removal- und On Orbit Servicing-Missionen geplant. Für beide ist ein unkooperatives Rendezvous-Manöver zwischen den Verfolger und dem Ziel notwendig. Häufig steht kein exaktes 3D-Modell des Ziels zur Verfügung, was die Aufgabe weiter erschwert. Viele Forschungsgruppen und Firmen aus der Weltraumindustrie entwickeln Systeme für unkooperative Rendezvous-Manöver.

Das RACOON-Lab an der TUM ist eine dieser Forschungsgruppen. In ihrem Hardware-in-the-loop-Teststand können Rendezvous-Missionen simuliert werden. In vorangegangenen Arbeiten wurde eine Toolchain entwickelt, die die Trajektorie des Verfolgers um das Ziel sowie eine 3D-Rekonstruktion des Ziels berechnet und dabei ausschließlich auf Daten von Sensoren, die auf dem Verfolger montiert sind, zurückgreift. Dafür wird der DIFODO-Algorithmus eingesetzt. Die Toolchain ist noch nicht für eine reale Mission geeignet, wobei Reflektionen in den verwendeten Bildern die meisten Probleme verursachen.

Diese Arbeit schlägt eine Toolchain zur Evaluation von Reinforcement Learning (RL) zur Verbesserung von Tiefenbildern, die von den auf dem Verfolger monierten Sensoren aufgezeichnet wurden, vor, um die Genauigkeit der 3D-Rekonstruktion und der Trajektorie zu verbessern. Der RL-Agent wählt einen Tiefenbildfilter aus, der auf die Eingabebilder angewendet wird, bevor diese vom DIFODO-Algorithmus verarbeitet werden. Für diese Toolchain wurden ein PPO-Algorithmus sowie eine dichte Belohnungsfunktion ausgewählt. Die RL-Umgebung wurde gemäß OpenAIs *Gym*-Schnittstelle implementiert. Sie kann daher unabhängig vom in dieser Arbeit genutzten RL-Agenten oder der in dieser Arbeit genutzten Implementierung desselben verwendet werden.

Es wurden aussagekräftige Diagrammarten entwickelt, die mit Funktionen, die als Teil dieser Arbeit implementiert wurden, erstellt werden können. Diese ermöglichen eine Evaluation der Performance des RL-Agenten und der Toolchain selbst. Die Aussagekraft der Diagrammarten wurde in drei Kategorien bewertet: als einzelnes Diagramm, im Vergleich mit anderen Diagrammen der gleichen Art und im Vergleich oder in Kombination mit Diagrammen anderer Art. Zusätzlich wurden Funktionen zur Optimierung zweier Parameter – der Anzahl an parallelen Umgebungen und der Lerngeschwindigkeit – implementiert und getestet. Diese ermöglichen eine schnelle Optimierung der Parameter. Die Kombination der Toolchain, der Diagramme und der Funktionen zur Parameteroptimierung bilden ein Werkzeug, das genutzt werden kann, um den Einsatz von RL in künftigen Studien des RACOON-Labs zu evaluieren.

Außerdem wurde die Belohnungsfunktion mit Hilfe einer zuvor im RACOON-Lab aufgezeichneten Trajektorie getestet. Dabei zeigten sich Probleme in allen Komponenten der Belohnungsfunktion, die vor allem von Schwankungen in den Koordinaten und Orientierungen, die der DIFODO-Algorithmus berechnet, hervorgerufen werden. Eine Möglichkeit, dieses Problem in künftigen Arbeiten zur Evaluierung von RL im RACOON-Lab anzugehen, wurde vorgeschlagen.

Abstract

In order to prevent the Kessler Syndrome and to extent the operational lifetime of satellites, Active Debris Removal and On Orbit Servicing missions are planned. Both rely on an uncooperative rendezvous between the chaser and the target. Often an exact 3D model of the target is not available which makes the task even more difficult. Many research groups as well as different space industry companies are developing systems for an uncooperative rendezvous.

The RACOON-Lab at TUM is one of those research groups. In its hardware-in-the-loop test stand, rendezvous missions can be simulated. In previous studies, a toolchain was developed that calculates the trajectory of the chaser around the target and a 3D reconstruction of the target using only the data coming from the sensors mounted on the chaser. For that purpose, a DIFODO algorithm is used. This toolchain is not suitable for a real mission yet with reflections in the input image causing the most problems.

This thesis proposes a toolchain to evaluate Reinforcement Learning (RL) to enhance the depth images recorded by the sensors of the chaser in order to improve the accuracy of the 3D reconstruction and the trajectory. The RL agent chooses depth image filters to apply to the input image before it is processed by the DIFODO algorithm. For this toolchain, a Proximal Policy Optimization algorithm and a dense reward function were chosen. The RL environment was implemented following OpenAI's *Gym* interface. It can thus be used independent of the RL algorithm or the implementation of the same used in this thesis.

Expressive figure types which can be created using functions implemented as part of this thesis were developed. These enable an evaluation of both the RL agent's performance and the toolchain itself. The expressiveness of the figures types were ranked in three categories: as an individual figure, when being compared to figures of the same type and when being compared to or combined with figures of other types. Additionally, functions to optimize two essential parameters – the number of parallel environments and the learning rate – were implemented and tested. These allow a quick tuning of the parameters. In combination the toolchain, the figures, and the parameter optimization functions form a tool which can be used to evaluate the usage of RL in future studies of the RACOON-Lab.

Furthermore, the reward function was tested using a trajectory previously calculated in the RACOON-Lab. These tests showed problems in all of the components of the reward function, most of which are caused by fluctuations in the coordinates and orientation angles calculated by the DIFODO algorithm. An option to address the problems in the reward function has been proposed for future evaluations of RL in the RACOON-Lab's toolchain.

Contents

List of Figures

List of Tables

Symbols and Formulas

Symbol	**Unit**	**Description**
S_t	-	state at timestep t
H_t	-	history at timestep t
O_t	-	observation at timestep t
R_t	-	reward at timestep t
A_t	-	action at timestep t
G_t	-	return at timestep t
γ	-	discount factor
ϵ	-	rate at which the agent explores
δ^{π_θ}	-	TD error of policy π_θ
γ_{ref}	rad	correct angle between two timesteps
u	rad	roll angle
v	rad	pitch angle
w	rad	yaw angle
t	s	time passed between two timesteps
R	m	distance between target and chaser in RACOON-Lab
$\boldsymbol{v}_{ref}$	m/s	correct translational velocity
$\boldsymbol{v}$	m/s	calculated translational velocity
$\boldsymbol{\omega}_{ref}$	rad/s	correct angular velocity
$\boldsymbol{\omega}$	rad/s	calculated angular velocity
$d_{translation,norm}$	-	normalized euclidian distance of the velocity vectors
$s_{translation,norm}$	-	normalized scalar product between velocity vectors
$d_{rotation,norm}$	-	normalized euclidian distance of the angular velocity vectors
$s_{rotation,norm}$	-	normalized scalar product between angular velocity vectors
E_t	-	absolute error at timestep t
$R(t)$	-	reward signal at timestep t
δ_i	-	distance of point i from the best-fit-circle
δ_{rl}	-	summed squared distance of all points from the best fit-circle for the toolchain using RL
δ_{cur}	-	summed squared distance of all points from the best-fit circle for the current toolchain

Acronyms

A2C	Advantage Actor Critic
A3C	Asynchronous Advantage Actor Critic
ACER	Sample efficient actor-critic with experience replay
ACKTR	Actor Critic using Kronecker-Factored Trust Region
ADR	Active Debris Removal
ALE	Arcade Learning Environment
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
CPU	Central Processing Unit
Deep RL	Deep Reinforcement Learning
DLR	German Aerospace Center
DQN	Deep Q-Network
EPOS	European Proximity Operations Simulator
ESA	European Space Agency
EVAs	Extravehicular Activities
FPS	frames per second
GEO	Geostationary Orbit
GPU	Graphical Processing Unit
JAXA	Japanese Aerospace Exploration Agency
K-FAC	Kronecker-factored approximate curvature
LEO	Low Earth Orbit
LRT	Institute of Astronautics
MDP	Markov Decision Process
ML	Machine Laerning
MPI	Message Passing Interface
MWN	Münchner Wissenschaftsnetz

NASDA Japanese National Space Developement Agency

OOS On Orbit Servicing

POMDP Partially Observable Markov Decision Process

PPO Proximal Policy Optimization

RACOON-Lab Real-Time Attitude Control and On-Orbit Navigation Laboratory

RL Reinforcement Learning

SB Stable Baselines

SB2 Stable Baselines 2

SB3 Stable Baselines 3

SGD Stochastic Gradient Descent

SLAM Simultaneous Localization and Mapping

TRPO Trust-region policy optimization

TUM Technische Universität München

1 Introduction

Since the first successful deployment of a satellite in space in 1957, human space activities have been increasing rapidly. More and more satellite applications for science, communication, and earth observation have been developed. In 1978, *Kessler* and *Court-Palais* published [1] in which they described the scenario which was later named ”Kessler Syndrome” by *Gabbard*. The Kessler Syndrome is a scenario in which objects orbiting the Earth collide with each other at a frequency that creates a belt of fragments around the Earth which threatens any human spaceflight activities. The prediction of *Kessler* and *Court-Palais* in 1978 was that this belt could begin to form around the year 2000 and that it will grow exponentially even if no further artificial objects are launched into space. In 2010, *Kessler et al.* published [2] in which they confirm the initial predictions of [1] and state that Active Debris Removal (ADR) is likely to become necessary to avoid the Kessler Syndrome from becoming a reality. Any artificial object orbiting Earth which is unused – for example a satellite which ran out of propellant – is called space debris.

Figure 1–1 shows the number of artificial objects in space over time. The number of objects is increasing rapidly, especially in the last decade. So far, four accidental collisions between cataloged objects have been reported, one of which – the Iridium-Cosmos-collision – *Kessler et al.* consider catastrophic [2]. Anti-satellite tests such as the very recent Russian one [3] come in addition and are also considered catastrophic by the authors.

On Orbit Servicing (OOS) describes missions in space where servicing, ,maintenance, refueling, repairs, or physical upgrades, is performed. They are also seen as an option to reduce the amount of new space debris since they extend the operational lifetime of a satellite. Any ADR or OOS mission requires a successful rendezvous between the chaser, i.e., the approaching spaceship, and the target, i.e., the spaceship to approach.

At the Institute of Astronautics (LRT) at Technische Universität München (TUM), the

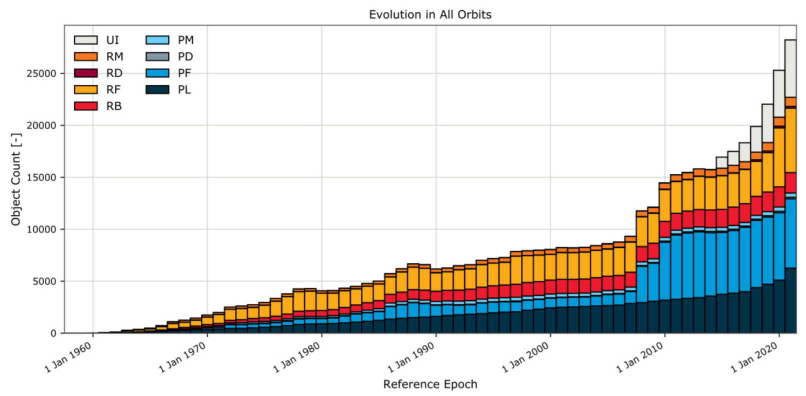


Fig. 1–1: Artificial objects in space [4]; all acronyms explained in table 1–1

Tab. 1–1: Explanation of acronyms used in figure 1–1

PL	Payload
PF	Payload Fragmentation Debris
PD	Payload Debris
PM	Payload Mission Related Object
RB	Rocket Body
RF	Rocket Fragmentation Debris
RD	Rocket Debris
RM	Rocket Mission Related Object
UI	Unidentified

Real-Time Attitude Control and On-Orbit Navigation Laboratory (RACOON-Lab) was established to simulate rendezvous missions. *Dziura et al.* introduced it in [5]. The RACOON-Lab's toolchain estimates the trajectory of the chaser around the target. The results are not sufficient for a real mission yet with reflections in the input image causing the most problems [6].

Following these findings from previous studies in the RACOON-Lab, the main goal of this thesis is to develop a tool for the evaluation of RL for depth image enhancement in the RACOON-Lab. The idea is to improve the toolchain's results and handle the reflections when occurring using depth image filters chosen by a RL agent. For the task at hand, no pairs of depth images and best-suited filters are available. The usage of supervised Machine Laerning (ML) for the task is thus not possible. RL in contrast is capable of learning without such a dataset using a reward function.

2 State of science and technology

2.1 Orbital Proximity Operations

Nowadays, the term OOS implies robotic OOS although first OOS missions were carried out by astronauts in Extravehicular Activities (EVAs) during the servicing of Intelsat VI in 1992 and the Hubble Space Telescope in 1993 [7]. Reasons to develop robotic OOS, although human servicing had already been successfully performed, were the high risks for the astronauts as well as the enormous costs of the missions [8]. First automated OOS experiments were conducted by the Japanese Aerospace Exploration Agency (JAXA) (formerly Japanese National Space Developement Agency (NASDA)) in 1999 with the ETS-VII mission, the results of which were reported in [9]. This mission was the first to successfully control the satellite attitude and a robotic arm cooperatively [7].

Cooperative describes the state in which not only the chaser's but also the target's attitude can be controlled, which is generally not the case for OOS and by definition not the case for ADR missions. Non-cooperative missions do not rely on that constraint, which is why a chaser capable of approaching the target non-cooperatively is desired for OOS and necessary for ADR missions. However, no non-cooperative system has yet been successfully demonstrated for targets without special adoptions like reflector plates. Therefore, many researchers and private companies are developing systems for the non-cooperative case. The following will give an overview about the most relevant projects in that area without a specific order.

The RemoveDEBRIS mission [10] was developed by a consortium of universities and companies from the space sector and launched in 2018. It successfully performed tests for a net and a harpoon capturing mechanism with the net targeting a CubeSat and the harpoon a target of the size of a table tennis bat. Besides that, the mission also performed tests for vision-based navigation using a digital camera and a flash imaging LiDAR system. The tests for a dragsail to reduce the deorbiting of the CubeSat mothership from Low Earth Orbit (LEO) by 90% compared to the duration of deorbiting without further actions were unsuccessful.

Astroscale, a company headquartered in Japan, has launched its ELSA-D mission [11] in March 2021. It aims to demonstrate a magnetic capturing mechanism of a non-tumbling and a tumbling target. Like the RemoveDEBRIS mission, ELSA-D also uses a CubeSat as the chaser and a smaller CubeSat as the target.

With its second Mission Extension Vehicle (MEV-2) [12], the US company Northrop Grumman has been successfully performing OOS in Geostationary Orbit (GEO) since docking MEV-2 with its target satellite IS-10-02 in April 2021. MEV-2 extends the operational lifespan of the target satellite, which was about to run out of propellant, by providing a propulsion system. However, as IS-10-02 was still fully functional, MEV-2 cannot be seen as demonstration of non-cooperative docking.

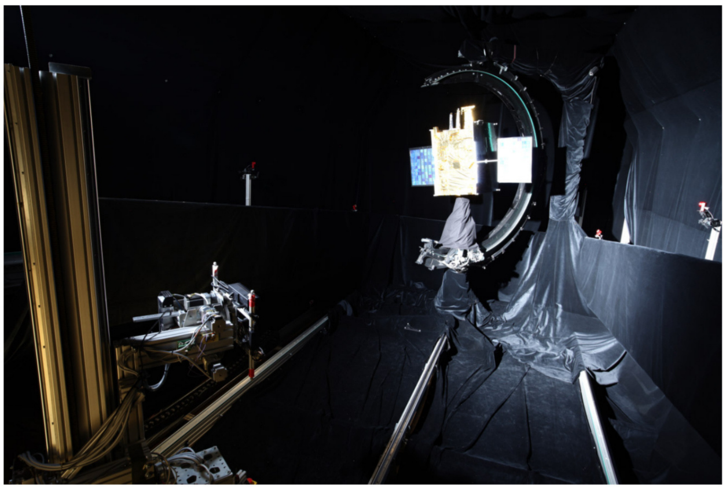


Fig. 2–1: The RACOON's test-stand with the chaser on the left and the target on the right [6]

Under its Clean Space Initiative, the European Space Agency (ESA) has planned the e.deorbit mission [13] which was meant to develop the technology necessary for ADR. In 2018, the funding for e.deorbit was stopped and instead the ClearSpace-1 mission [14] was financed with €86 million by ESA. The mission is scheduled for 2025 and aims to deorbit a VESPA upper part launched in 2013. The project is conducted by a consortium of European companies led by Swiss startup ClearSpace.

The German Aerospace Center (DLR)) operates the European Proximity Operations Simulator (EPOS) 2.0 [15]. It consists of two robotic manipulators, each having six degrees of freedom. One of the robotic manipulators can be moved on a linear slide which is 25m long. Thus, at EPOS 2.0 the last 25m of rendezvous missions can be simulated in real-time. Both hardware, especially sensors and cameras, and software can be tested at EPOS 2.0. To make the lightning conditions similar to the ones expected in a real mission, a Sun simulator is available and can be used to simulate front, side and back Sun illumination.

Another research group focusing on solving the challenges of non-cooperative docking is the RACOON-Lab. With its hardware-in-the-loop test-stand, it simulates the final stage of rendezvous missions and aims to develop software capable of calculating the correct trajectory of the chaser relative to the target. The test-stand is shown in figure 2–1. In the RACOON-Lab, the chaser which is equipped with a 3D camera can be moved around the target satellite which can be rotated around 4 axes. The RACOON-Lab can simulate both Sun and Earth Albedo illumination. Its test-stand was compared to EPOS 2.0 by *Rehn* in [16].

The images recorded by the 3D camera mounted on the chaser are fed into a toolchain which aims to calculate the trajectory of the chaser relative to the target. The current toolchain consists of three steps as developed by *Franceschini* in [6]. First, a box filter is applied to filter out the background of the lab which will not be present in a real space mission. After that, the recorded 3D images are processed by a DIFODO algorithm which estimates the trajectory of the chaser relative to the target. In the

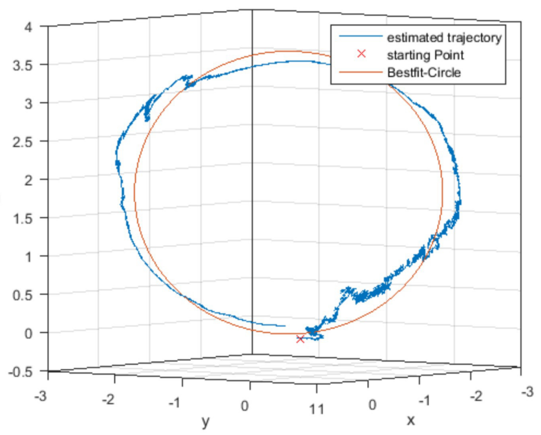


Fig. 2–2: Visualization of best-fit circle by *Rehn* [16]

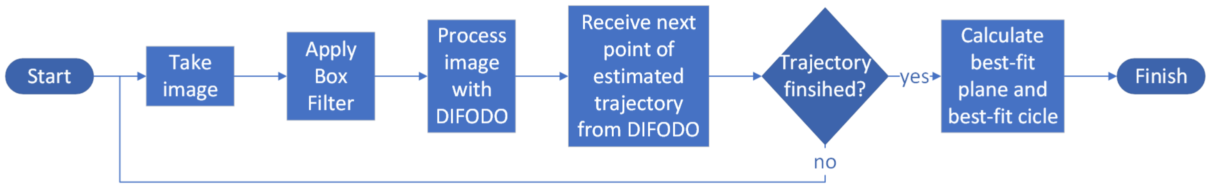

Start
Take image
Apply Box Filter
Process image with DIFODO
Receive next point of estimated trajectory from DIFODO
Trajectory finsihed?
yes
no
Calculate best-fit plane and best-fit cicle
Finish

Fig. 2–3: The current toolchain of the RACOON-Lab

RACOON-Lab, the real trajectory of the chaser is known to always be an exact circle. In [16], *Rehn* has proposed a metric to evaluate the toolchain. According to that metric, the points of the trajectory are projected on a best-fit plane on which a best-fit circle is calculated. Figure 2–2 visualizes the metric. The current toolchain of the RACOON-Lab is summarized in figure 2–3.

The results of the RACOON-Lab's toolchain are not sufficient for a real mission yet. Especially reflections from the illumination on the target seem to cause problems in estimating the correct trajectory. In her semester thesis [17], *Pregel Hoderlein* proposed an autoencoder, i.e., an architecture of neural networks, to enhance the images before being processed by the DIFODO algorithm. The results of the DIFODO algorithm were used to calculate the loss function for the neural networks since ground truth data, i.e., perfect images without reflections or noise, are not available.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a subcategory of Machine Laerning (ML) that is comparable to the human learning behavior. In RL, a so-called **agent** acts inside an **environment**, i.e., the task setting, in the following way: Given a **state** by the environment, i.e., the current setting, the agent chooses an **action** to apply to this state from a predefined set of actions. The action chosen by the agent is then applied to the state and evaluated by the environment. This evaluation results in a **reward** in form of a positive or negative scalar which is given to the agent. With that feedback, the agent learns which actions to apply to which state as it aims to maximize the rewards over the training. In most environments, a new state is entered after an action was applied. In such case, the previously described steps are executed until the environment stops, i.e., the **episode** ends. This can be the case either because a predefined termination condition has been met (for example by entering a specific state) or because a maximum number of steps has been executed. In the typical case of entering a new state after executing an action, the reward for the following states is often included with a discount into the reward for the action applied. This ensures that the global maximum reward is reached instead of a local maximum.

In contrast to supervised ML methods, to train a RL agent, no prelabeled data but only a reward function is necessary. However, finding an appropriate reward function for the respective environment is of great importance since a reward function with wrong incentives will lead to an agent with unintended behavior. Another important setting is the definition of a state for the respective environment. When working with images for example, intuitively it may seem reasonable to use one image as the state. However, moving objects cannot be replicated by a single image, which is why using multiple images as the state might lead to improved performance as for example shown in [18].

2.2.1 Deep Reinforcement Learning

Deep Reinforcement Learning (Deep RL) uses deep learning methods like Convolutional Neural Networks (CNNs) to learn successful policies. CNNs are Artificial Neural Networks (ANNs) especially suited for processing images. Such ANNs consist of three types of layers: input, hidden and output layers. ANNs with multiple hidden layers are called deep. After advancements in deep learning for image and speech recognition, *Mnih et al.* proposed Deep RL in [18]. They trained a CNN to play different Atari 2600 games with the last four down-scaled monochrome screen images as the states.

According to *Sutton and Barto* [19], the greatest achievement of *Mnih et al.* were not the outstanding results for the specific games, many of which beat previous RL approaches, but doing so without any game-specific modifications. Other RL approaches at that time used special feature-sets for each game [19]. These manually selected features-sets were the key factor for the agent's performance. Hence, training a new RL agent with good performance needed experience and time. With the Deep RL approach of *Mnih et al.* however, this task became much easier. Also, non-deep RL approaches can only process states of limited complexity. Deep RL techniques in contrast can process even complex states like one, or even multiple, images directly.

2.2.2 Reinforcement Learning for 2D image processing

For 2D image processing, RL has already been used successfully. *Furuta et al.* summarize the works in that area in [20]: [21] uses RL for image cropping. [22] implemented color enhancement using Deep RL. [23] performs image face hallucination, i.e., the generation of high-resolution faces from low-resolution input images, using Deep RL. [24] makes use of Deep RL for image restoration. In [25], multiple 2D image filters have been applied in parallel in a filter fusion by the RL agent which adjusted the weights used to represent each filtered image in the output image.

3 Objective and Methodology

The huge efforts to develop systems for a non-cooperative rendezvous undertaken by multiple research groups and private companies underline the great influence of the field. The RACOON-Lab is one of those research groups and focuses on the testing of sensor systems and the development of software for this purpose. Its current toolchain is not sufficient for a real mission yet, with illumination reflections causing the greatest problems [6].

In order to improve the toolchain's performance and to bring it nearer to a state where it is capable for a real mission, this thesis will evaluate the adoption of RL-based depth image enhancement to the toolchain. To the best of the author's knowledge, this is the first attempt to adapt RL for depth image enhancement in the context of orbit proximity operations. With recent developments in RL, especially Deep RL, and its proven ability to enhance 2D images as outlined in chapter 2.2.2, the author sees great potential in the use of RL for the described case. Thus, this thesis will propose strategies on how to use RL for depth image enhancement in the toolchain of the RACOON-Lab. The reward function will play a crucial role in the design decision due to its great influence on the agent's performance.

As this thesis focuses on the usage of RL for depth image enhancement, other applications of RL in the context of the RACOON-Lab will not be investigated. For example, one could use RL to build an improved box-filter or an alternative toolchain. When evaluating RL only for depth image enhancement, there is one point in the toolchain that is particularly qualified to apply depth image filters chosen by the RL agent. Therefore, it is decided that the actions chosen by the RL agent will be applied after the application of the box-filter and before providing the filtered image to the DIFODO algorithm as visualized in figure 3–1. This thesis will not evaluate other options where to apply RL for depth image enhancement in the toolchain of the RACOON-Lab.

This point is particularly qualified for implementing RL in the RACOON-Lab's toolchain as the depth image filter chosen by the RL agent must be applied before the depth image is provided to the DIFODO algorithm. Otherwise, the filter would have no influence on the trajectory calculated by the DIFODO algorithm. Also, it is no option to apply the filter chosen by the RL agent before the application of the box filter since in that case, the agent may be influenced by the Lab's background which is not present in an application in space.

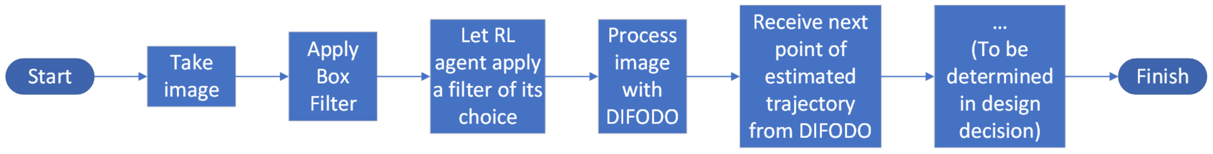

Start
Take image
Apply Box Filter
Let RL agent apply a filter of its choice
Process image with DIFODO
Receive next point of estimated trajectory from DIFODO
... (To be determined in design decision)
Finish

Fig. 3–1: New toolchain for the RACOON-Lab implementing RL

Additionally, as part of this thesis, one of the strategies will be implemented and evaluated. To train the RL agent, previously recorded data will be used.

The implementation of the DIFODO algorithm available for this thesis is yet untested and hence may not work as expected. Hence, this thesis will not finally evaluate the usage of RL in the RACOON-Lab's toolchain and will not compare the toolchain using RL to the existing one which does not. Instead, it aims to provide a toolset capable to evaluate the usage of RL in the RACOON-Lab's toolchain, once the implementation of the DIFODO algorithm was tested and verified to work as expected. It will not only implement software to include RL in the RACOON-Lab's toolchain but also implement software creating figures which can be used to evaluate the agent's performance and the toolchain. The resulting software should be easily useable and optimizable once the implementation of the DIFODO algorithm was tested and verified to work as expected. Therefore, the software will be tested and verified as far as possible with the untested implementation of the DIFODO algorithm.

Due to its great influence on the agent's performance, a special focus will be laid on the reward function. As far as this is possible despite the untested implementation of the DIFODO algorithm, the reward function will be analyzed and evaluated. Additionally, if applicable, changes to the reward function will be proposed.

The current toolchain of the RACOON-Lab is capable of running in real-time on hardware available in space. Since ML and hence RL are computationally intensive and often involve the usage of Graphical Processing Units (GPUs), this thesis does not aim to develop code which can run in real-time on such hardware. In a real-world application, the recorded data could be sent to a ground station and processed on capable hardware instead. The results of the toolchain could then be sent back to the chaser.

The created figures used to evaluate the toolchain will be ranked in three categories in the discussion, all of which will rate the figure on a scale from $--$ to $++$: the first category will rate the expressiveness of the individual figure, the second one the expressiveness of the figure when being compared to figures of the same type and the third one the expressiveness when being compared to or combined with figures of other types. When ranking the figures in the categories, the two purposes named above (evaluation of the agent's performance and evaluation of the toolchain) will be taken into account.

Table 3–1 defines the criteria used to classify the figures' expressiveness. Cells with a list name multiple options. If at least one of these criteria is fulfilled, the figure type receives the rating in this category. Some terms of the table need a definition which will be given in the following: *Less relevant* information refers to an overview for example while *relevant* information could be showing that the agent is improving over the training or how actions and rewards correlate. For categories 2 and 3, a distinction is made between *single-time* and *multi-time* figures. *Single-time* figures can be created only once per training and a comparison needs two trainings. *Multi-time* figures in contrast can be created and compared multiple times per training and between trainings, for example because they refer to a single episode. The highest in this category remains $++$. In case a figure type meets more than one criterion in category 2 or 3 and the combination of these criteria is not explicitly stated in table 3–1, it receives the highest rating of the individual criteria.

Tab. 3–1: Criteria for the ranking the figures' expressiveness

Rating	Category 1	Categories 2 and 3
🟥 $--$	• No information can be observed • Hint to a less relevant information	• No additional information can be observed • One single-time, less relevant information
🟧 $-$	• Clearly observable, less relevant information • Hint to relevant information	• One single-time, relevant information • Two or more single-time, less relevant information • One multi-time, less relevant information
🟨 o	Clearly observable, relevant information	• Two or more single-time, relevant information • Two or more multi-time, less relevant information • One multi-time, relevant information • One single-time, relevant information and one multi-time, less relevant information
🟩 $+$	Clearly observable, relevant information and hint to a less relevant information	• Two or more single-time, relevant information and one multi-time, relevant information • Two or more single-time, relevant information and two or more multi-time less relevant information • Two or more multi-time less relevant information and one multi-time relevant information • Two or more multi-time, relevant information
🟩 $++$	Clearly observable, relevant information and a hint to a relevant information	• Two or more single-time, relevant information and two or more multi-time relevant information • Two or more multi-time, relevant information and two or more multi-time, less relevant information

4 Design Decision

In this chapter, the most important design decisions for the implementation of RL in the toolchain of the RACOON-Lab are made. First, it decides on the way RL is implemented in the RACOON-Lab's toolchain. Then, an appropriate (Deep) RL algorithm is chosen. Afterwards, the reward function is designed. Next, the actions for the RL agent, i.e., the filters, are selected. The final subchapter then decides on the way of implementation, i.e., the programming language and the software packages.

In each subchapter, multiple options are outlined first. Then they are discussed and one of them is selected.

4.1 Training toolchain

Before deciding on a (Deep) RL technique and a reward mechanism, it needs to be decided how RL will be implemented in the RACOON-Lab's toolchain. This does not include a decision on when to apply the filters chosen by the RL agent since this point was fixed in chapter 3. Instead, a decision is made on the toolchain used for training the RL agent.

When training a RL agent, it is of great importance to ensure an useful reward because the training of the agent relies on that. Thus, the reward needs to incentive the expected behavior. For this thesis, this means to choose actions that ensure a circular trajectory.

To achieve this goal, an appropriate training toolchain is key. Various options are conceivable. The following will discuss three options: training on one, on multiple and on all images of an experiment. It is worth mentioning that the trajectory of the DIFODO algorithm is influenced by the previous images. Therefore, performing an action influences not only the point calculated by the DIFODO algorithm for the particular image, but also all the following ones. However, since the input image is not totally changed by the actions, the influence of the particular action on the points calculated for the following images is considered low. The following will use the term *evaluation toolchain*. In the context of this thesis, the term refers to the toolchain used in a real world application, i.e., the RL agent chooses an action for all images.

4.1.1 Training on one image per iteration

The first option is to train the RL agent only on one single image per iteration. In this toolchain, the reward can be clearly assigned to the action. However, applying a filter to a single image may result in changes so small that they cannot be adequately measured.

Also, in a real world application, filters chosen by the RL agent would be applied to all images recorded. Training and evaluating the agent on only one image per iteration might lead to worse performance in such applications. Since the final goal of the implementation of RL in the RACOON-Lab's toolchain is to improve its performance in real world applications, this point is of great importance.

4.1.2 Training on multiple images per iteration

Within this category, there are two options to perform the training. The images the RL agent is trained on can be either sequential or distributed over all the images from the experiment. Both options will be evaluated in the following.

Sequential

When training the RL agent on multiple sequential images per iteration, the results can most likely be measured adequately since they are of sufficient size. However, the performance of a single action is slightly harder to track compared to training on a single image because the reward given to an action could be influenced by previous actions. Keep in mind that a dense reward for an action can include the discounted reward for the state entered by the action. Using this, the influence of previous actions on the result of following ones could be made assessable if necessary.

Training on multiple images in sequential order would partially address the problem of different toolchains for training and evaluation which could lead to worse performance in the evaluation. To fully address the problem, the training iteration needs to involve all images of an experiment.

Distributed

Training on multiple images per iteration, which are distributed over all images of the experiment, would most likely suffer from the same problems as training on one single image per iteration while also lowering the traceability of each action's performance. Anyway, the training might run a little faster compared to the training on one single image.

4.1.3 Training on all images of an experiment per iteration

The last option for a training toolchain that will be evaluated in this chapter is to train the RL agent on all images of an experiment. This can be seen as the extended way of the sequential case described previously. Hence, the traceability of each action's performance is most likely harder compared to training on only one image per iteration. Nonetheless, the training and evaluation toolchain would be quite similar, so good training results would most likely lead to good evaluation results.

4.1.4 Decision for a toolchain

The decision for the right training toolchain is a trade-off between the traceability of each action and the similarity of the training and evaluation toolchain. The former would increase the chances of good training performance while the latter would increase the chances of good evaluation performance and ensure that the agent does not learn something unintended.

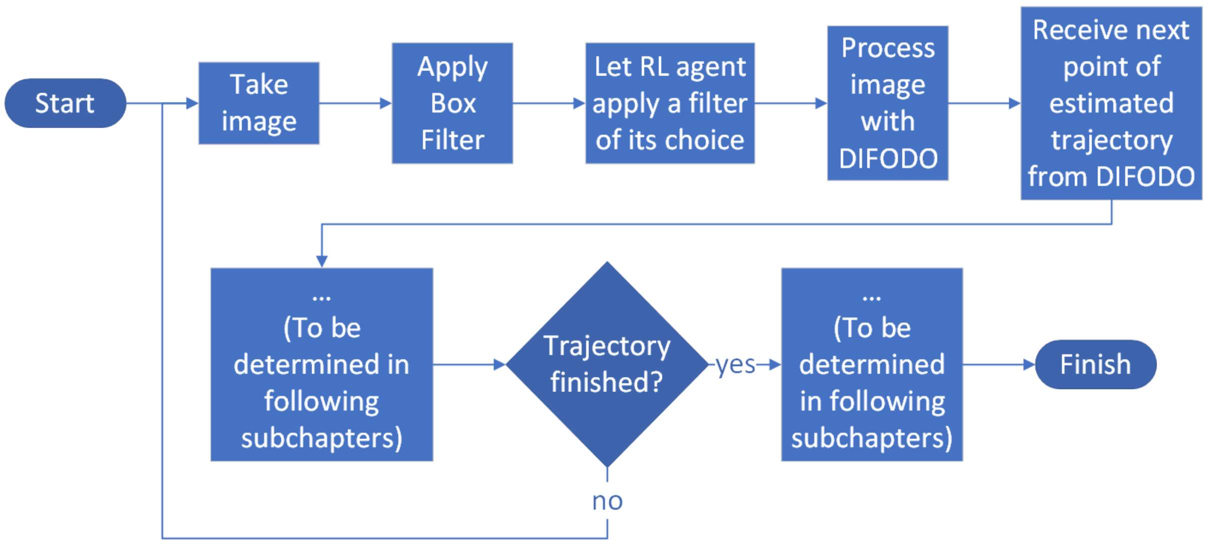

Start
Take image
Apply Box Filter
Let RL agent apply a filter of its choice
Process image with DIFODO
Receive next point of estimated trajectory from DIFODO
... (To be determined in following subchapters)
Trajectory finished?
yes
no
... (To be determined in following subchapters)
Finish

Fig. 4–1: Current state of training toolchain visualized

Most applications of RL perform actions continuously and cannot track the performance of an individual action completely. The problem of traceability is thus a typical problem in RL applications. Regardless of this problem, functional RL applications have been developed. Hence, RL algorithms are able to deal with the problem of traceability. The similarity of training and evaluation toolchain therefore weights higher than the traceability of each action. Consequently, the training will be performed on all images of an experiment per iteration. The resulting toolchain is visualized in figure 4–1.

4.2 (Deep) Reinforcement Learning Algorithm

4.2.1 Fundamentals of Reinforcement Learning

Before outlining options for a RL algorithm and afterwards choosing one of them, some fundamentals of RL will be explained. The following explanations are mainly drawn from a lecture held by *David Silver*, one of the authors of [18] proposing Deep RL, at University College London in 2015 [26]. The explanations follow the notation of the standard textbook [19] which used capital letters for random variables and lower case letters for values of random variables and scalar functions. The scope of this explanation is not to summarize the complete field of RL but to introduce the reader to the most basic ideas and to make the reader familiar with the most common terms.

RL is typically applied to a Markov Decision Process (MDP) which is defined as a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$. $\mathcal{S}$ is a finite set of states. [27, p. 24] Formally, a state S_t is a function of the history H_t ($S_t = f(H_t)$). The history H_t is the sequence of all observations, actions and rewards seen so far

$$H_t = O_1, R_1, A_1, \ldots, A_{t-1}, O_t, R_t \,. \text{ [28, p. 18]} \tag{4–1}$$

Depending on the problem to solve, the environment may be only partially observable, meaning that the agent only knows a part of what defines the environment. Think of a card game where the agent only knows the public and its own cards but not the cards of its opponents. This setting is called Partially Observable Markov Decision Process (POMDP). [28, p. 24] In a POMDP, an Observation O_t is the information given to the agent by the environment at a certain time step.

$\mathcal{A}$ is a finite set of actions. $\mathcal{P}$ is a state transition probability matrix where

$$\mathcal{P}^a_{ss'} = \mathbb{P}[s_{t+1} = s' \mid S_t = s, A_t = a] \text{. [27, p. 24]} \tag{4–2}$$

$\mathcal{R}$ is a reward function where

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] \text{. [27, p. 24]} \tag{4–3}$$

$\gamma \in [0, 1]$ is a discount factor that reflects how much future rewards are influencing the return G_t which is calculated by

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \text{. [27, p. 12]} \tag{4–4}$$

The goal of the RL agent is to maximize the return at each state [28, p. 15].

A policy π describes the agent's behavior and can be understood as a map from state s to action a. It is either deterministic ($a = \pi(s)$) or stochastic ($\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]$). [28, p. 26] A stationary policy is time-independent, i.e., $A_t \sim \pi(\cdot \mid S_t), \forall t > 0$ [27, p. 26]. An agent searches an optimal policy which is defined as follows:

$$\pi_* \geq \pi, \forall \pi \text{ [27, p. 40]} \tag{4–5}$$

Whether a policy is grater than another, i.e., $\pi \geq \pi'$, is determined using the state value function $v(s)$:

$$\pi \geq \pi' \text{, if } v_\pi(s) \geq v_{\pi'}(s), \forall s \text{ [27, p. 40]} \tag{4–6}$$

For any MDP, a deterministic optimal policy exists [27, p. 41].

In order to solve MDPs, numerous types of agents have been proposed. They can be categorized in *Value-Based*, *Policy-Based* and *Actor-Critic* approaches, all of which are either *Model-Free* or *Model-Based* [28, p. 34f.].

4.2.1.1 Value-Based RL

Value-Based RL makes use of a value function. A value function determines the expected return from that state onwards. There are two types of value functions: The *state-value function* $v_\pi(s)$ tells us how much return we can expect starting in state s and following policy π from that point onwards.

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] \text{ [27, p. 28]} \tag{4–7}$$

The *action-value function* $q_\pi(s, a)$ in contrast also takes the action taken by the agent into account when predicting the value.

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \text{ [27, p. 26]} \tag{4–8}$$

Both the state- and the action-value function can be expressed recursively using the Bellman Expectation Equation

$$v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi \text{ [27, p. 36]} \tag{4–9}$$

in the following way:

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1} \mid S_t = s)] \tag{4–10}$$

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \text{ [27, p. 30]} \tag{4–11}$$

A MDP can be solved by finding the optimal value function which is the maximum of all value functions.

$$v_*(s) = \max_\pi(v_\pi(s)) \tag{4–12}$$

$$q_*(s,a) = \max_\pi(q_\pi(s,a)) \text{ [27, p. 37]} \tag{4–13}$$

The optimal value functions are connected by the Bellman Optimality Equation

$$v_*(s) = \max_a(\mathcal{R}_s^a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \text{ [27, p. 45]} \tag{4–14}$$

in the following way for a deterministic policy:

$$v_*(s) = \max_a(q_*(s,a)) \text{ [27, p. 43]} \tag{4–15}$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a v_*(s') \text{ [27, p. 44]} \tag{4–16}$$

When the optimal action-value function is found, the return can be maximized by acting greedy, meaning that the action with the highest value function is chosen at each state by a deterministic optimal policy π_*. This deterministic optimal policy is defined using a conditional probability, i.e., a probability is assigned to an action a given a state s.

$$\pi_*(a \mid s) = \begin{cases} 1 & \text{if } a = \arg\max\limits_{a \in \mathcal{A}}(q_*(s,a)) \\ 0 & \text{otherwise} \end{cases} \text{ [27, p. 41]} \tag{4–17}$$

Value-Based agents learn the value function and thus are able to solve the MDP once they found the correct value function. When searching for the optimal value function in a large state space, the agent follows an implicit policy, for example the ϵ-greedy policy. The parameter $\epsilon \in [0,1]$ of the ϵ-greedy policy determines the rate between *exploration* and *exploitation*. *Exploration* refers to the agent exploring unseen or uncertain terrain, meaning it chooses actions even though they seem not optimal with the current knowledge. By this *exploration*, the agent may find better ways than the ones it has explored previously. In contrast, *exploitation* refers to the agent maximizing the immediate return by choosing the best action known so far for the respective state. The ϵ-greedy policy is a common way to deal with the problem of finding a balance between *exploration* and *exploitation*. At a rate of ϵ, the agent explores, i.e., chooses an action randomly. At a rate of $1 - \epsilon$ the agent exploits, i.e., chooses the action of which it expects the highest return to receive from when starting in the current state. In other words: it acts greedy.

There are also variants of the ϵ-greedy policy which do not use a fixed ϵ but decrease it over time as the agent is expected to know its environment better. [29, p. 11]

When working with many states and actions, the value function has to be approximated because a table for all states and actions is too large to store and learning the value of each state and space is too slow. To approximate the value function, the parameters $\boldsymbol{w}$ are used. Any type of function approximator can be chosen. Typical choices are linear combinations of features and ANNs. When using differentiable function approximators such as the two mentioned, the parameters $\boldsymbol{w}$ can be updated using gradient descent or other optimization methods. [30, p. 6]

Value-Based methods are usually working off-policy what means that they can learn from actions of others. Working on-policy in comparison means that the agent has to be in control of the policy in order to learn. [29, p. 5]

Since Value-Based methods are working off-policy they have a high sample efficiency, meaning they can exploit the data observed and therefore need less data in comparison to other methods. However, they highly suffer from poor convergence and are less stable than other methods, especially when using function approximators. This is the case because Value-Based methods operate in the value space but the ultimate goal is to find an optimal policy. A small change in the value space might lead to a bigger change in the policy space. The last point is the main reason for the disadvantages of Value-Based methods. [31, p. 18]

4.2.1.2 Policy-Based RL

Policy-Based agents learn a policy without using a value function. Therefore, the policy is parametrized with the parameters θ:

$$\pi_\theta(s,a) = (P)[a \mid s, \theta] \text{ [32, p. 3]} \tag{4–18}$$

Using this formulization, Policy-Based RL can be understood as an optimization problem. One can formulate a function $J(\theta)$ like the *average value*

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s) \tag{4–19}$$

or the *average reward per time-step*

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s,a) \mathcal{R}_s^a \tag{4–20}$$

where $d^{\pi_\theta}(s)$ is the stationary distribution of the Markov chain for π_θ. Using this function $J(\theta)$, one can optimize the policy by maximizing $J(\theta)$ using optimization methods such as gradient descent. [32, p. 10f.]

The main advantages of Policy-Based methods are their better convergence properties compared to Value-Based methods and their effectiveness in high-dimensional or continuous action spaces. Also, stochastic policies can be learned. The main disadvantage of these methods is that Policy-Based agents typically converge to local rather than global optima. Additionally, the evaluation of a policy is often inefficient and suffers under high variance. [32, p. 5]

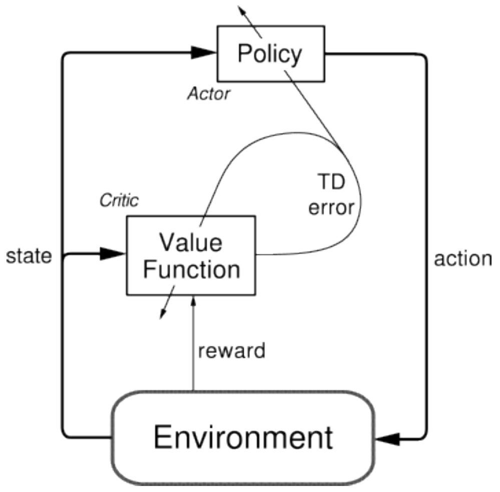


Fig. 4–2: The actor-critic architecture [33, p. 151]

4.2.1.3 Actor-Critic RL

Actor-Critic agents combine value and Policy-Based approaches by learning both the value function and the policy. As visualized in figure 4–2, the policy serves as the actor with the parameters θ and the value function as its critic with the parameters w. [32, p. 23] Both are provided with the state, but only the value function receives the reward from the environment. After an action was performed and a reward was received, the value function computes the TD error

$$\delta^{\pi_\theta} = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{4–21}$$

which is used to update both the value function and the policy. [33, p. 151]

Advantage actor-critic methods approximate the advantage of an action

$$A(s, a) = Q_w(s, a) - V_v(s) \,. \tag{4–22}$$

The advantage A is then used to learn the parameters θ of the policy without the need for the critic to approximate two functions ($Q(s, a)$ and $V(s)$). The usage of the advantage function can significantly reduce the variance of policy gradient. [32, p. 30]

As the idea of combining value and Policy-Based approaches already suggests, actor-critic methods combine the best of both. They are sample efficient, have lower variance compared to Policy-Based methods and perform well in continuous state-action spaces. [32, p. 26]

4.2.1.4 Model-free and Model-based RL

A model is a representation of a MDP with the parameters η which the agent learns [34, p. 12]. In model-free RL, the agent learns the value function, the policy, or both from experience. In model-based RL, the agent learns a model from experience and plans on the value function, the policy, or both from that model. [34, p. 6] Figure 4–3 visualizes that. Direct RL is an extension of model-based RL that lets the agent also learn from experience. [34, p. 25]

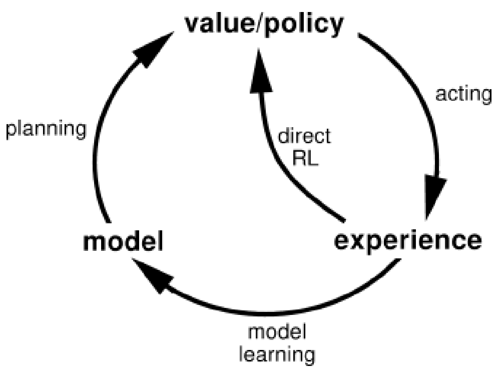


Fig. 4–3: Model-based RL visualized [33, p. 231]

The advantages of model-based RL are that it can use supervised learning methods to efficiently learn the model and that it can evaluate the model's uncertainty. The main disadvantage is that if both the model and the value function are approximated with parameters, there are two approximation errors in the feedback to the agent. [34, p. 11]

4.2.2 Definition of selection criteria

Now that the basic terms and ideas of RL have been introduced, options for a RL algorithm will be outlined. Afterwards, a decision is made for one of the options. However, first the selection criteria will be defined.

The algorithm to implement in the RACOON-Lab's toolchain should be capable of handling one or even multiple depth images, possibly downscaled, as an input. Otherwise, one would have to manually select features which represent the depth image well enough to choose suitable actions. Since the introduction of deep RL by [18] in 2013, this error-prone task is no longer necessary, as deep RL algorithms are capable of processing large inputs such as multiple images.

The next criterion for an appropriate algorithm is a good performance. A common way to measure the performance of RL algorithms is to compare their cumulative reward, i.e., the sum of all rewards received so far, over the number of steps. In order to fully optimize the algorithm's hyperparameters such as the learning rate α or ϵ for the ϵ-greedy policy, it has to be run multiple times for this comparison. One algorithm is superior to another algorithm if its curve is consistently above the other's. [35, chapter 12.6] Alternatively, the reward for every episode can be compared. Once again, one algorithm is superior to another if its curve is consistently above the other's. Since deep RL was proposed in [18] to play Atari games, this setting has become a standard comparison for RL algorithms in discrete action spaces. In discrete action spaces, the agent chooses one out of several available options. In continuous action spaces, the agent controls one or multiple continuous parameters The first could be to choose one out of several available filters, all of which would have a fixed intense while the latter could be to control the intense of a filter that is applied to an image. The Arcade Learning Environment (ALE) by *Bellemare et al.* [36] for example offers an environment for evaluating RL algorithms' performance in Atari games. Since the implementation will also use a discrete action space, comparison between algorithms will be done using

ALE as a metric.

Good performance should be paired with high scalability, which is the third criterion. Great performance which cannot be transferred to the intended application is not useful. The ALE metric compares RL algorithms in a variety of applications with different types of games and thus also checks the scalability of an algorithm. In combination with the second criterion, the best algorithm is not necessarily the one performing best in terms of cumulative score across all games since this one may perform bad in some games and compensate this with very good performance in other games. Since it is uncertain which Atari games do relate the most to the desired application in the RACOON-Lab's toolchain, a suitable algorithm should work well for most games, not just very well for some.

Finally, an appropriate algorithm should be well-documented and available in a major programming library which is optimized to run the calculations as quickly as possible, for example on a GPU. This final criterion will drastically reduce the duration of the implementation and training of the algorithm as well as the duration of the computation in a real-world usage if the state is large, e.g., an image.

Now that the criteria for the algorithm have been set, multiple options will be outlined.

Stable baselines is a common python library providing state-of-the-art RL algorithms including Advantage Actor Critic (A2C), Sample efficient actor-critic with experience replay (ACER), Actor Critic using Kronecker-Factored Trust Region (ACKTR), Deep Q-Network (DQN), Proximal Policy Optimization (PPO) and Trust-region policy optimization (TRPO). It was published on GitHub [37]. In its documentation [38], a guide to select an appropriate RL algorithm is provided [39]. It divides the algorithms by two criteria: *discrete* or *continuous* actions and *single process* or *multi-processed*. The multiprocessing criterion is further divided into multiprocessing with and without the usage of Message Passing Interface (MPI). The following will evaluate which criteria apply for our use-case.

As explained previously, the implementation of this thesis will operate in a discrete action space. The environment replicating the RACOON-Lab toolchain will be a custom environment. With the vectorized environments of stable baselines this custom environment can be used for multi-processing. To allow usage of the environment independent of MPI availability, this thesis restricts itself to multiprocessing without MPI. The documentation of stable baselines suggests PPO2, A2C and its successors ACKTR and ACER for this case. The following will now evaluate these algorithms according to the selection criteria.

Since all of the algorithms are deep, they all fulfill the first criterion. The stable baselines library is based on *Tensorflow*, a common ML library for python. With this well-documented implementation, all algorithms also fulfill the last criterion. The final subchapter of the design selection will however evaluate alternatives to the stable baselines library for the selected algorithm. The following will now briefly introduce all algorithms and then compare their performance and scalability using ALE as a metric.

4.2.3 Algorithm PPO2

PPO2 is a Proximal Policy Optimization (PPO) algorithm. The family of PPO algorithms was proposed by *Schulman et al.* in [40]. This family of algorithms combine the benefits of TRPO [41], namely their stability and reliability, with the multi-worker architecture of A2C. It uses multiple epochs of stochastic gradient ascent to perform each policy update. PPO algorithms are mainly based on the idea that a policy update should not move the policy too far from the old one.

The original paper includes a pseudo-code version of a PPO algorithm that uses fixed-length trajectory segments which is shown in algorithm 1. It uses

$$L_t^{CLOP+VS+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \tag{4–23}$$

where c_1 and c_2 are coefficients, S denotes an entropy bonus and

$$L_t^{VF} = (V_\theta(s_t) - V_t^{targ})^2 \,. \tag{4–24}$$

It also uses

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \ldots + \ldots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{4–25}$$

with

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \,. \tag{4–26}$$

Algorithm 1 PPO algorithm [40]

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

Schulman et al. name the following benefits of their family of algorithms over TRPO: simpler to implement, more general, and better overall performance.

4.2.4 Algorithm A2C

Advantage Actor Critic (A2C) is an adoption of *Asynchronous Advantage Actor Critic (A3C)* which was proposed by *Mnih et al.* in [42] but without the asynchrony. This means that in A2C the value function and the policy are optimized simultaneously whereas in A3C there is a delay which is called asynchrony. Both algorithms are advantage actor-critic algorithms. By using multiple workers, the usage of a replay buffer as used in the DQN can be avoided.

A2C has proven to be more effective than A3C which is why it is a common baseline to compare newly proposed algorithms against [43]. A3C however had already outperformed the DQN algorithm, the first deep RL algorithm, of *Mnih et al.* [18], a Value-Based approach. The comparison of A3C and DQN in Atari games reported in [42] is shown in figure 4–6. Since it was outperformed by A3C, DQN will not be considered here.

4.2.5 Algorithm ACKTR

Actor Critic using Kronecker-Factored Trust Region (ACKTR) is a deep actor-critic RL algorithm proposed by *Wu et al.* in [43] which is a successor of A2C. It uses *Kronecker-factored approximate curvature (K-FAC)* to update both the actor and the critic which are both deep ANNs. K-FAC [44] is an optimization algorithm that speeds up the training of various ANNs and requires only little more computational efforts compared to Stochastic Gradient Descent (SGD) (10-25% according to *Wu et al.*). A more detailed description of the principle of the ACKTR algorithm is out of scope of this introduction but can be found in the original paper [43].

4.2.6 Algorithm ACER

Sample efficient actor-critic with experience replay (ACER) is a deep actor-critic RL algorithm that combines several ideas from other algorithms. It was proposed by *Wang et al.* in [45] and uses experience replay like DQN, multiple workers like A2C and a trust region like TRPO. *Wang et al.* also proposed several new ideas such as a new trust region policy optimization method or stochastic dueling network architectures. The appendix of the original paper also offers a pseudo-code implementation for discrete actions which will not be presented in this thesis due to its length. *Wang et al.* report that ACER is stable, sample efficient and matches the performance of other RL methods on Atari (they compared to A3C and DQN in figure 4–6). However, they also report improvements over state-of-the-art RL algorithms in continuous action space.

4.2.7 Performance and stability comparison

For the Atari game Atlantis, *Wu et al.* report ACKTR to be 10 times more sample efficient than A2C [43]. Figure 4–4 compares the performance of ACKTR, A2C and TRPO in six different Atari games. The figure shows that ACKTR outperforms the other two algorithms in every game. Figure 4–5 compares ACKTR, A2C and PPO2 in five different Atari games. Once again, ACKTR outperforms the other two algorithms in every game. Figure 4–6 compares both the sample efficiency (left) and the computation efficiency (right) of ACER, A3C and DQN for 57 Atari games. The DQN algorithm performs worst by both measures. While A3C is the second-worst in terms of sample efficiency, it performs very well in terms of computation efficiency. The different versions of ACER perform well on both metrics. Figure 4–7 compares the performance of ACKTR, PPO, A2C and ACER in 49 Atari games. Please note that the hyperparameters of ACKTR were tuned using only one game, *Breakout*. In this figure, no algorithm performs best in all games. However, ACKTR performs better than A2C in almost all of them. For the comparison of ACER, PPO, and ACKTR, no clear result can be drawn from this figure.

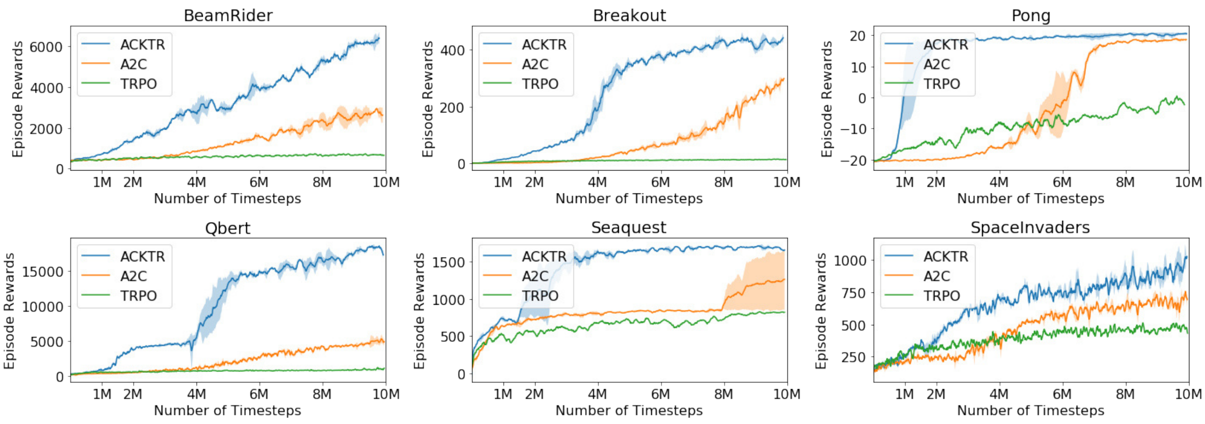


Fig. 4–4: Performance of ACKTR, A2C and TRPO in six Atari games compared. Standard deviation over two random seeds is denoted by shaded regions. [43]

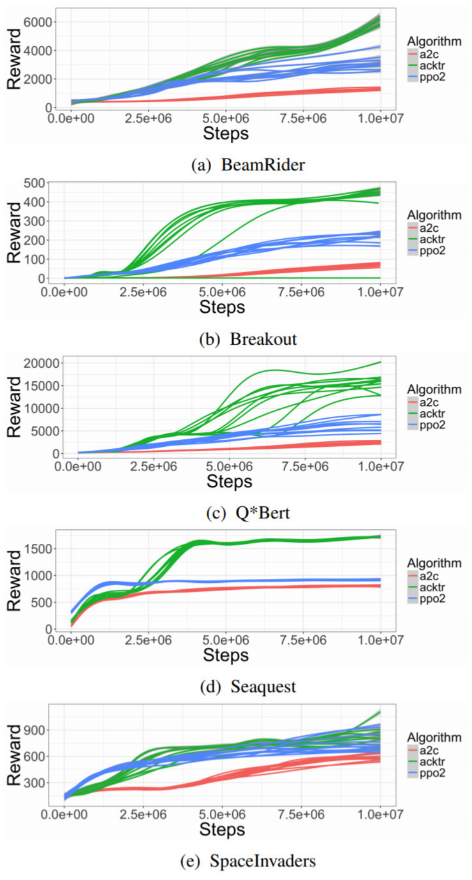


Fig. 4–5: Performance of ACKTR, A2C and PPO2 in five Atari games compared [46]

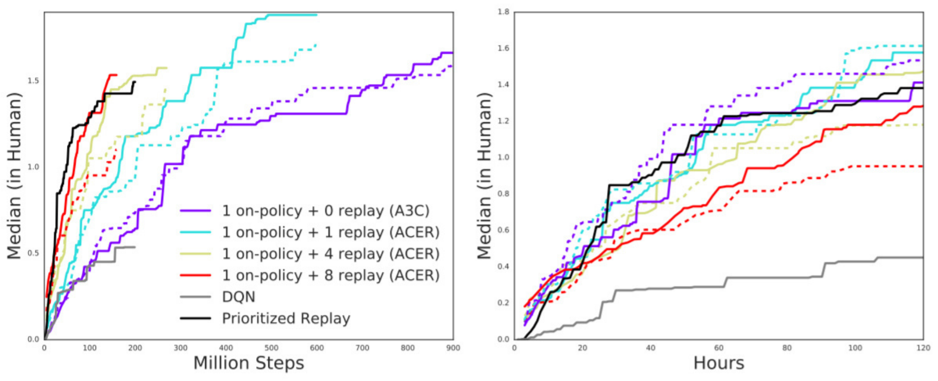


Fig. 4–6: Performance of ACER, A3C and DQN in 57 Atari games [45]

PPO seems to work well in games in which all other algorithms have difficulty, such as *Kangaroo* or *Zaxxon*. ACER shows similar behavior for other and fewer games. ACKTR overall outperforms PPO in more games than vice versa.

Please note that only figure 4–5 was not published by the developers of one of the algorithms and hence is the only fully independent comparison. As reported by *Henderson et al.* in [48], improvements over state-of-the-art algorithms, especially when reported by the developers themselves, do not have to be meaningful since choosing only some games or optimizing the hyperparameters very intensively can lead to an improvement for a very special case that cannot be replicated in other settings. Without neglecting the findings of the developers of the algorithms, this has to be taken into account when using the figures as a comparison.

Tab. 4–1: Comparison of RL algorithms for discrete action space and multiprocessing available

Algorithm	Based on	Performance	Stability
A2C	A3C	Slowest	Least stable
ACER	A2C	Second-slowest	Second most stable
ACKTR	A2C	Fastest	Second least stable
PPO	TRPO and A2C	Second-fastest	Most stable

4.2.8 Decision for an algorithm

Table 4–1 summarizes the findings of the performance and scalability analysis. Since A2C performs worst in both performance and scalability, it is not considered any further. Since all other algorithms are based on A2C at least to some extent, this result was not unexpected. ACER shows good scalability but is outperformed by both PPO and ACKTR and will therefore not be considered further. Even though ACKTR shows remarkable performance, it cannot scale that performance across all games. PPO in comparison shows good performance and great scalability. Since the developers of PPO laid a special focus on scalability, this was also not unexpected. By combining

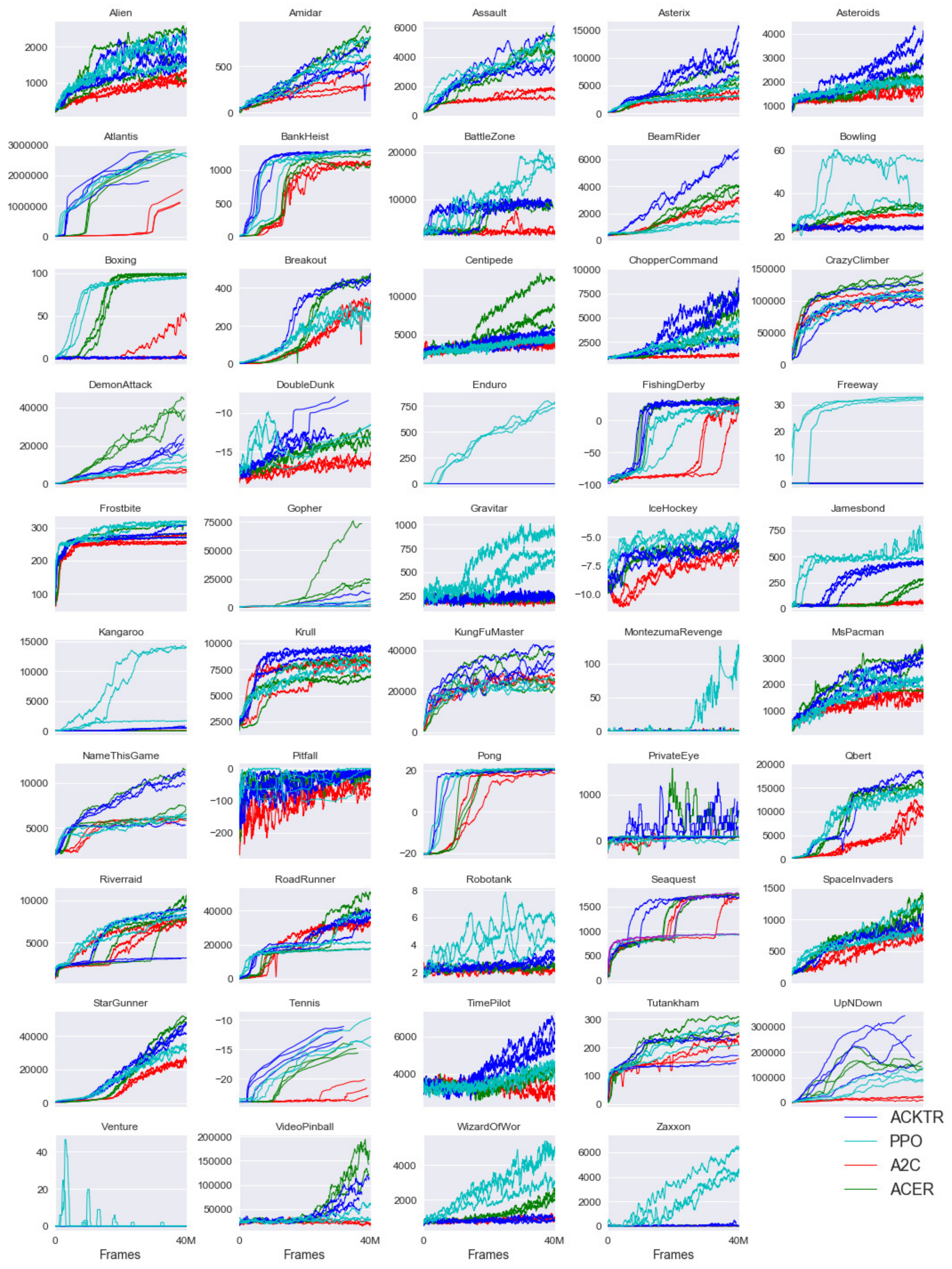


Fig. 4–7: Performance of ACKTR, PPO, A2C and ACER in 49 Atari games [47]

good performance and great scalability, PPO meets the selection criteria better than the other algorithms. Therefore, PPO will be used in this thesis.

4.3 Reward function

Like for the algorithm, before the options for the reward function will be outlined, the main ideas and most important terms of reward functions will be explained.

4.3.1 Fundamentals of reward functions

RL is based on a fairly simple theorem which states that any goal can be described by a reward function that must be cumulatively maximized [28, p. 13]. However, finding a suitable reward function that effectively incentives the agent to reach that goal is arguably the most important design decision for any RL application. Before outlining options for the reward function to train the RL agent in the RACOON-Lab's toolchain on, some fundamentals and basic terms regarding reward functions are introduced.

Typically, the value of an action also includes the rewards for the following actions, which are most often discounted with a discount factor γ:

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s] \text{ [28, p. 27]} \qquad (4–27)$$

One uses high values for γ, i.e., near 1, when the actions are expected to have a strong influence on the following states and rewards. Imagine a game of tic-tac-toe where the agent sees itself in the dilemma that the opponent will win in its next move no matter which action the agent chooses since there are two options for the opponent to get three symbols in a row. When the discount factor γ is low, i.e., near or equal to 0, the negative reward for the opponent's win will not have much influence on the previous action which led to the dilemma. Therefore, the agent will not learn that the previous action was bad and also will not change its behavior.

Rewards can be delayed, i.e., the agent receives the rewards after a certain delay, for example a certain number of timesteps [28, p. 8]. Typically, one expects the agent to figure out that there is a delay and deal with that delay itself.

Reward functions can be sparse or dense. Sparse reward functions give the RL agent only occasional feedback. That means that the agent receives a reward of 0 most of the time. An example of a sparse reward function is an environment of the game tic-tac-toe that only gives the agent a reward once the game is over. Since this reward accounts for all of the actions taken previously, one typically sets $\gamma = 1$ when using a sparse reward function. A dense reward function in contrast provides the agent with continuous feedback about its actions. This means to define certain sub-goals. For the game tic-tac-toe, an environment could for example provide a reward for stopping the opponent from winning or for placing two symbols in a row. To design a dense reward function is often challenging since one has to think of good measurements for the agent's actions. Wrongly designed reward functions will lead to unintended behavior of the agent. However, a well designed dense reward function will accelerate the training of the agent drastically compared to a sparse reward setting, especially for complex environments.

4.3.2 Dense reward: Metric by Pregel Hoderlein

In her Semester's Thesis [17, p. 15ff.], *Pregel Hoderlein* proposes a metric as a loss function for a neural network aiming to enhance depth images in the RACOON-Lab's toolchain. The following will summarize the ideas and calculations of this metric.

The metric is based on the knowledge about the experimental setup at the RACOON-Lab. In the RACOON-Lab, the chaser circles the target with constant distance and constant angular velocity in the XZ-plane. *Pregel Hoderlein* calculates the angle γ_{ref} by which the chaser circled the target in the time $t = t_2 - t_1$ passed between two arbitrary positions with the indices 1 and 2 using the angular velocity ω_{ref}.

$$\gamma_{ref} = w_{ref} \cdot t \text{ [17, p. 15]} \tag{4–28}$$

The correct translational velocity used as a reference $\boldsymbol{v}_{ref}$ is calculated using γ_{ref}:

$$\boldsymbol{v}_{ref} = \begin{bmatrix} v_{x,ref} \\ v_{y,ref} \\ v_{z,ref} \end{bmatrix} = \begin{bmatrix} \frac{R \cdot \sin(\gamma_{ref})}{t} \\ 0 \\ \frac{R \cdot (1-\cos(\gamma_{ref}))}{t} \end{bmatrix} \text{ [17, p. 16]} \tag{4–29}$$

The correct angular velocity used as a reference $\boldsymbol{\omega}_{ref}$ has only one entry since the correct trajectory of the chaser lies in the XZ-plane.

$$\boldsymbol{\omega}_{ref} = \begin{bmatrix} \omega_{x,ref} \\ \omega_{y,ref} \\ \omega_{z,ref} \end{bmatrix} = \begin{bmatrix} 0 \\ \omega_{ref} \\ 0 \end{bmatrix} \text{ [17, p. 16]} \tag{4–30}$$

Pregel Hoderlein calculates translational velocities $\boldsymbol{v}$ for the calculated trajectory as follows:

$$\boldsymbol{v} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} \frac{x_2 - x_1}{t} \\ \frac{y_2 - y_1}{t} \\ \frac{z_2 - z_1}{t} \end{bmatrix} \text{ [17, p. 17, corrected]} \tag{4–31}$$

The DIFODO algorithm outputs the yaw, roll, and pitch angle. The angular velocity $\boldsymbol{\omega}$ for the calculated trajectory is calculated as follows with u being the roll, v the pitch and w the yaw angle:

$$\boldsymbol{\omega} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \frac{w2 - w1}{t} \\ \frac{v2 - v1}{t} \\ \frac{u2 - u1}{t} \end{bmatrix} \text{ [17, p. 17, adpoted]} \tag{4–32}$$

The absolute error E averages the four components $d_{translation,norm}$, $s_{translation,norm}$, $d_{rotation,norm}$ and $s_{rotation,norm}$ which are calculated as follows:

$$d_{translation,norm} = \begin{cases} \frac{\left| \|\boldsymbol{v}_{ref}\| - \|\boldsymbol{v}\| \right|}{\|\boldsymbol{v}_{ref}\|} & \text{if } \frac{\left| \|\boldsymbol{v}_{ref}\| - \|\boldsymbol{v}\| \right|}{\|\boldsymbol{v}_{ref}\|} \leq 1 \\ 1 & \text{if } \frac{\left| \|\boldsymbol{v}_{ref}\| - \|\boldsymbol{v}\| \right|}{\|\boldsymbol{v}_{ref}\|} > 1 \end{cases} \text{ [17, p. 18f.]} \tag{4–33}$$

$$s_{translation,norm} = \frac{1 - \frac{\boldsymbol{v}_{ref}}{\left\|\boldsymbol{v}_{ref}\right\|} \cdot \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}}{2} \text{ [17, p. 18f.]} \tag{4–34}$$

$$d_{rotation,norm} = \begin{cases} \frac{\left|\left\|\boldsymbol{\omega}_{ref}\right\| - \|\boldsymbol{\omega}\|\right|}{\left\|\boldsymbol{\omega}_{ref}\right\|} & \text{if } \frac{\left|\left\|\boldsymbol{\omega}_{ref}\right\| - \|\boldsymbol{\omega}\|\right|}{\left\|\boldsymbol{\omega}_{ref}\right\|} \leq 1 \\ 1 & \text{if } \frac{\left|\left\|\boldsymbol{\omega}_{ref}\right\| - \|\boldsymbol{\omega}\|\right|}{\left\|\boldsymbol{\omega}_{ref}\right\|} > 1 \end{cases} \text{ [17, p. 18f.]} \tag{4–35}$$

$$s_{rotation,norm} = \frac{1 - \frac{\boldsymbol{\omega}_{ref}}{\left\|\boldsymbol{\omega}_{ref}\right\|} \cdot \frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}}{2} \text{ [17, p. 18f.]} \tag{4–36}$$

$$E = \frac{d_{translation,norm} + s_{translation,norm} + d_{rotation,norm} + s_{rotation,norm}}{4} \text{ [17, p. 19]} \tag{4–37}$$

All components are normalized between 0 and 1 and thus the absolute error E is also normalized between 0 and 1. A value of 0 stands for a perfect estimation and a value of 1 for a poor one. $d_{translation,norm}$ measures the similarity of the length of the velocity vector $\boldsymbol{v}$ to the reference velocity vector $\boldsymbol{v}_{ref}$. In other words: it compares the absolute value of the velocities. $s_{translation,norm}$ checks whether the two velocity vectors are pointing in the same direction. $d_{rotation,norm}$ and $s_{rotation,norm}$ do the same for the angular velocity vector $\boldsymbol{\omega}$ and the reference angular velocity vector $\boldsymbol{\omega}_{ref}$.

This metric can be adopted fairly easily as a dense reward function for the RL environment by using the negative absolute error E as the reward function which then would be in $[-1, 0]$. Alternatively, one could scale the reward function to $[-1, 1]$ to give a positive reward for a perfect estimation. Since the RL agent learns to maximize the returns anyway, scaling the reward function will not have much impact besides using the whole range of $[-1, 1]$ which is typically used for reward functions. Thereby, the distance between two rewards is doubled compared to the non-scaled version. When E_t is the absolute error for timestep t, the final reward function $R(t)$ can be calculated as follows:

$$R(t) = 1 - 2E_t \tag{4–38}$$

This thesis assumes that actions are not influencing the performance of the following decisions much. Therefore, the discount factor γ will be set to 0 and the performance of an actions is only determined by the reward for that specific action. However, since it is uncertain how large that influence is, it might be interesting to test, for example, $\gamma = 0.5$ in future studies of the RACOON-Lab evaluating RL.

A dense reward function should split the overall goal into smaller pieces. The overall goal for the agent is to apply filters that make the trajectory calculated by the DIFODO algorithm more accurate. For the experiments conducted at the RACOON-Lab, this means to bring the estimated trajectory closer to the shape of a circle. The proposed dense reward function, adopted from *Pregel Hoderlein* [17], splits the overall goal into appropriate sub-goals, because if all elements of the calculated trajectory were circular, the calculated trajectory as a whole would be a circle.

4.3.3 Sparse Reward: Total distance to best-fit circle

Alternatively, one could use a sparse reward function that sums up the squared distance of each point on the trajectory calculated by the DIFODO algorithm to the best-fit circle. This sum then would be given as a negative reward to the agent. This reward function can only be calculated at the end of an episode, i.e., when all images of an experiment were processed by the toolchain. That is the case because the best-fit circle can only be calculated correctly when the final trajectory has been calculated by the DIFODO algorithm. Therefore, when using the best-fit circle as a metric, the reward needs to be sparse. When using a sparse reward function, one should set $\gamma = 1$ so that all actions get the same reward, since with a sparse reward function one cannot judge which action had the greatest impact on the reward. Also, all rewards but the last one should be set to 0 for a sparse reward. Finally, the result should be scaled to $[-1, 1]$. It should therefore be divided by a constant c and added with 1. For example, c could be the sum of squared distances of the recorded dataset as calculated by the RACOON-Lab's toolchain not using RL. As long as the agent does not double that sum of squared distances, the reward will be greater than -1. When δ_i is the distance of point i from the best-fit circle, this reward function $R(t)$ can be calculated as follows for n timesteps:

$$R(t) = \begin{cases} 0 & \text{if } t < n \\ 1 - \frac{1}{c}\sum_{i=0}^{n-1} \delta_i^2 & \text{if } t = n \end{cases} \tag{4–39}$$

4.3.4 Sparse Reward: Comparison to toolchain without Reinforcement Learning

The final option for a reward function also relies on the best-fit circle as a metric. Thus, the same points regarding sparse rewards as for the previous reward function apply, i.e., $\gamma = 1$ and all rewards except the last one set to 0. This reward function also needs to calculate the sum of the squared distance of each point from the best-fit circle. However, instead of directly using a factored version of that function as a reward function like the previous option, this reward function compares this measure to the current toolchain of the RACOON-Lab which does not use RL. Thereby, it ensures the agent actually improves the quality of the toolchain's results. When δ_{rl} is the summed squared distance of all points from the best-fit circle for the toolchain using RL and δ_{cur} the summed squared distance of all points from the best-fit circle for the current toolchain not using RL, one could calculate the reward function $R(t)$ for n timesteps as follows:

$$R(t) = \begin{cases} 0 & \text{if } t < n \\ \frac{\delta_{cur} - \delta_{rl}}{\delta_{cur}} & \text{if } t = n \end{cases} \tag{4–40}$$

If $\delta_{rl} < \delta_{cur}$, i.e., the agent improves the performance, the reward is positive. Otherwise, if $\delta_{rl} > \delta_{cur}$, i.e., the agent would worsen the performance, the reward function is negative. Dividing the function by δ_{cur} ensures that it stays within reasonable range. As long as $0 \leq \delta_{rl} \leq 2\delta_{cur}$, $R \in [-1, 1]$.

4.3.5 Decision for a reward function

Three options for a reward function were introduced. In the following, they will be evaluated and one of them will be selected.

The greatest benefit of the sparse options is that they are using the sum of squared distances from the best-fit circle as a metric, just like the metric developed by *Rehn* in [16] which is used in the RACOON-Lab's toolchain not using RL. This ensure that they incentive the agent in the right way. The third option which compares the performance of the RL toolchain against the toolchain not using RL additionally ensures that the trained agent makes an improvement to the toolchain. However, both sparse options suffer from the drawbacks of sparse reward functions, most importantly the low training speed compared to dense reward functions. These drawbacks get a greater influence the more challenging the environment is and the more actions are taken before a reward is received by the agent. Since the recordings in the RACOON-Lab can consist of more than thousand depth images, it would take a very long time to train the RL agent to achieve significant performance. If only sparse reward functions would incentive the agent in the right way, one would have to accept this major drawback.

Fortunately, an option for a dense reward function is also available. Since it is dense, it will not suffer under the low training speed of sparse reward functions. However, it cannot use the metric developed by *Rehn* and therefore may incentive the agent in a wrong direction. The probability for this was considered low when designing the reward function since the metric used in the first reward function measures how close the shape of the trajectory element is to a circle element. The remaining low probability was considered to be far outweighed by the higher training speed of the dense reward compared to the sparse ones. The first reward function was assumed to split the overall goal of having a circular trajectory in suitable sub-goals. Hence, it was implemented and used in this thesis. Figure 4–8 shows the resulting toolchain for the RACOON-Lab. However, as discussed in chapter 7.2.2, the metric suffers under fluctuations in the coordinates and orientation angles calculated by the DIFODO algorithm and thus did not perform as expected.

4.4 Actions the agent can choose from

4.4.1 Depth images of mirror-like surfaces

Inaccuracies in the depth images recorded in the RACOON-Lab are mainly caused by reflections of the mirror-like surfaces of the target's solar-panels [6]. According to *Mallick et al.* [49], mirror-like surfaces are not identified as objects, causing large patches of holes in the depth image or the interpretation of false objects behind the particular surface. The latter are caused by the fact that objects visible as a reflection in the mirror-like surface are calculated as being behind the surface at the exact distance they are in front of it. Figure 4–9 shows the effect by comparing (a) a RGB and (b) a depth image of the same scene. In the depth image, the mirror's pixels are either black or white. The black pixels refer to a hole in the depth image since no object was detected. The white pixels stand for an object detected behind the mirror, at almost the

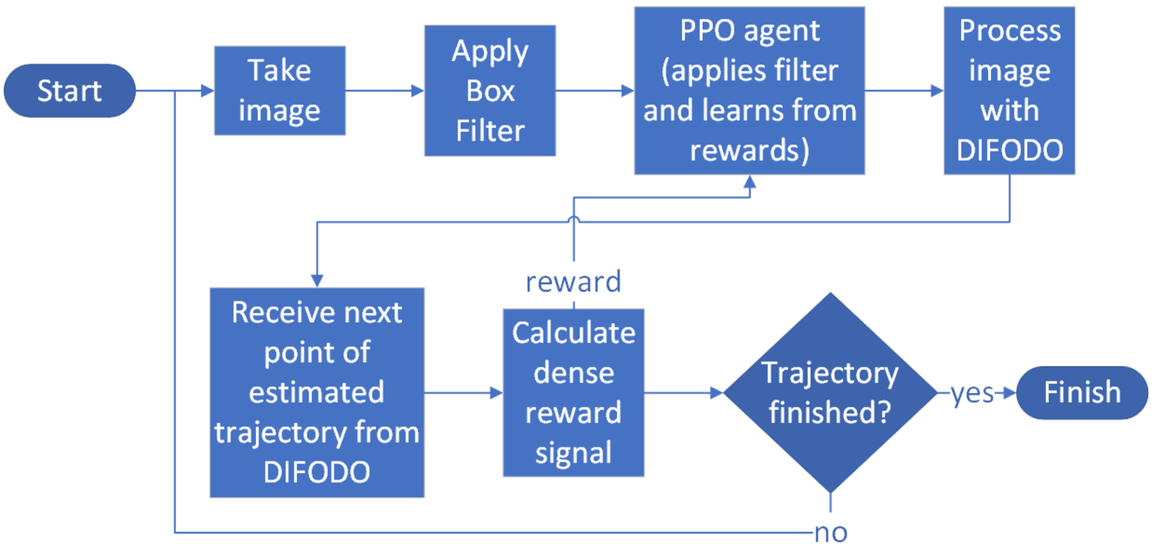

Start
Take image
Apply Box Filter
PPO agent (applies filter and learns from rewards)
Process image with DIFODO
Receive next point of estimated trajectory from DIFODO
reward
Calculate dense reward signal
Trajectory finished?
yes
Finish
no

Fig. 4–8: The resulting toolchain for the RACOON-Lab using PPO and a dense reward

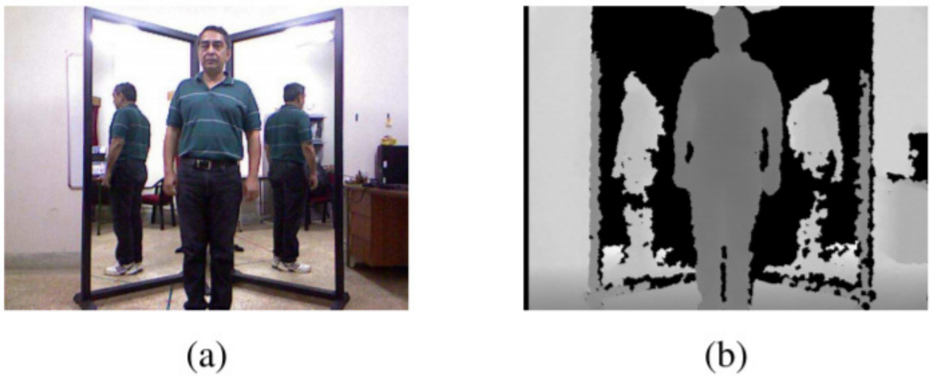
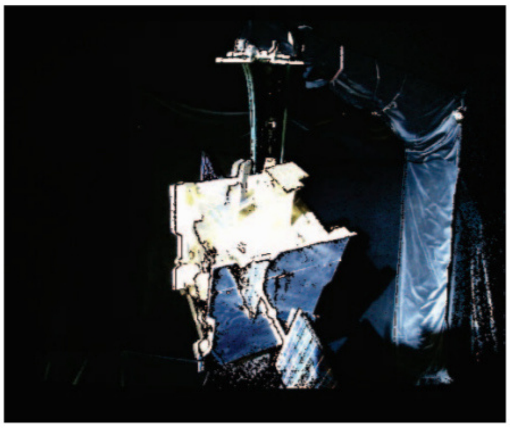


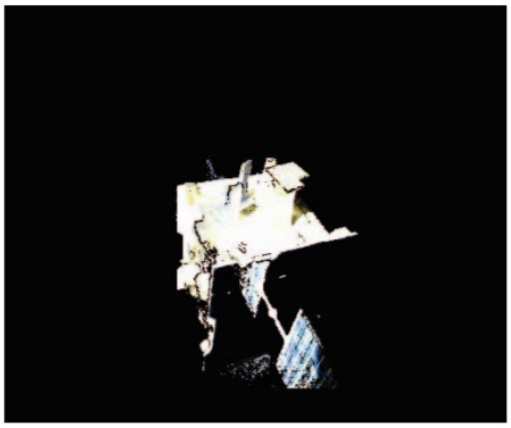
Fig. 4–9: Comparison of (a) RGB and (b) depth image showing that the Kinect camera taking the picture is unable to recognize the surface as an object and instead estimates the rear sides of the human with a lot of hole noise [49]

distance of the wall. Especially the inexistent objects cause problems for algorithms taking the depth image as an input like the DIFODO algorithm in the RACOON-Lab. Holes in comparison are considered to have much lower impact on the algorithm's performance. However, it would be ideal if the correct distance of the reflective surfaces would be calculated. Therefore, filters dealing with the reflections should be available for the RL agent to choose from.

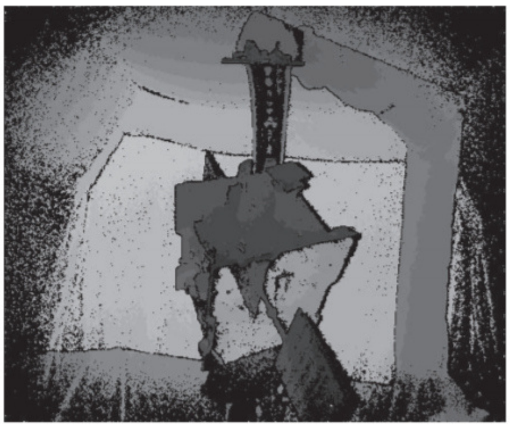
Figure 4–10 shows the effect on an image from the RACOON-Lab. In (d), the estimation for pixels of the solar panel fail because of the reflection on the same. The depth image (d) shows the solar panel behind the side of the target on which it is mounted, although it clearly is in front of it, as visible in the RGB image (b). Also, a reflection of the solar panel leads to miscalculation of points on the side the panel is mounted on. These points are much lighter than the rest of the surface in figure 4–10.



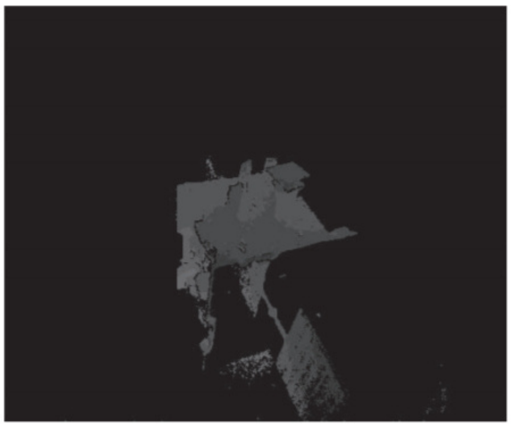
(a) raw RGB input



(b) filtered RGB output



(c) raw depth map input



(d) filtered depth map output

Fig. 4–10: RGB and depth image before and after application of the cuboid box filter used to filter out the Lab’s structures. In (d), the reflections on the solar-panel lead to a depth value estimation that is much further away from the camera than the solar panel actually is. [5]

4.4.2 Definition of selection criteria

Even though the cameras installed in the RACOON-Lab provide RGB and depth images, only the depth images are used for the toolchain since RGB images may be not available in an application in space when the mission is conducted in the Earth's shadow. Thus, guided filters, i.e., filters relying on RGB data, cannot be used. The following will therefore focus on unguided depth image filters.

In contrast to the previous design decision, multiple options will be chosen since the agent needs to have multiple options available to choose from. Therefore, only suitable options will be outlined.

Also, the filters will be chosen based on an expected performance. If a filter is not performing as expected, the RL agent will learn so and choose the filter less often. In the description of the results, the rate to which each filter was chosen over time will be analyzed.

Additionally, only filters which can be easily implemented or for which an implementation is already available will be used since this thesis does not aim to build new depth image filters but to use them as actions for RL agent.

A final constraint for suitable filters is that they should not need to be trained on the images. Otherwise, this would need additional learnings which are time-consuming and, more importantly, increase the probability of overfitting the data, i.e., performing well on the training data but worse on the testing data and in a real-world application.

Please note that mirror detection algorithms such as PSPNet [50], MirrorNet [51] and a method proposed by *Lin et al.* in [52] are working with RGB images as an input. Since RGB images may not be available in an application in space, these filters cannot be used as actions for the RL agent.

4.4.3 Adaptive depth threshold Filter

Since reflective surfaces are often calculated to be further away from the camera then they actually are, an adaptive depth threshold could address the problem. The filter could for example set any depth values which are above 95% of the maximum depth value to 0. In case the reflective surfaces are calculated to be behind the rest of the structure, this filter could overwrite them. However, this filter will not be of any use when the miscalculated points are within the range of the other depth values. In this case, the filter will overwrite correct depth values, which probably leads to worse performance of the DIFODO algorithm. The idea in using this filter is that the RL agent will recognize so and learn for which depth images this filter is useful and for which it is not.

This filter could be built using the *Inverted Threshold to Zero* function of the OpenCV library which sets all pixel values over a defined threshold to 0. A python implementation of the OpenCV library is available on GitHub [53] and also offers a documentation [54].

4.4.4 Percentile Filter

With a similar idea and concept as the *Adaptive depth threshold Filter*, a *Percentile Filter* is another option. Instead of setting values above 95% of the maximum depth value to 0, this filter sets values above the 95^{th} percentile to 0. The main difference is that the effect of this filter will not decrease when the maximum depth value is a far outlier as it might for the *Adaptive depth threshold filter*. However, both may be useful in different situations and will therefore be options for the RL agent to choose from.

To build this filter, the same libraries as for the *Adaptive depth threshold Filter* can be used.

4.4.5 IP-Basic

Ku et al. [55] proposed an unguided algorithm that only relies on basic image-processing operations and does not need to be trained on the images but still out-performed other methods on the *KITTI depth completion benchmark* [56]. The authors also see potential benefits for Simultaneous Localization and Mapping (SLAM) algorithms that perform a comparable task as the DIFODO algorithm. HEnce, it may be interesting to test the algorithm as a filter for the depth images from the RACOON-Lab even though it is not focused on dealing with reflective surfaces. The filter might need to be adjusted depending on its performance in future studies since it was developed for the *KITTI depth completion benchmark* and uses information about this dataset which may not apply to the RACOON-Lab.

Ku et al. have made their code available on GitHub and also give a recommendation which setup of their algorithm to use. They recommend their bilateral filter without extrapolation for practical use but also offer a slower version that additionally removes noise. This version still runs with 30 frames per second (FPS) on a Central Processing Unit (CPU) compared to 87 FPS for the bilateral filter. Since the toolchain to implement is not meant to run in real-time, the slower version called *Multi-Scale, Bilateral, Noise Removal, No Extrapolation* will be used as an action for the RL agent.

4.4.6 Median Filter

The median filter is a very common filter for RGB and depth images. Like the IP-Basic filter, it does not focus on dealing with reflective surfaces. However, it still could be a valid option for images without reflections. The median filter is implemented in the OpenCV library mentioned previously.

4.4.7 Do nothing

Another very valid option for the RL agent should be to not filter the depth image. Most probably there is a wide range of depth images that the DIFODO algorithm can handle without further filtering since the toolchain without RL also performs acceptable in most depth images. For these cases, no further filter is needed and thus the RL agent will be provided with the option to hand over the depth image to the DIFODO algorithm without applying any filter to it.

4.5 Programming Library

4.5.1 Definition of selection criteria

In this final section of the design decision chapter, a library for the PPO algorithm is selected. Only libraries written in Python will be taken into account since Python has become the standard programming language for ML tasks and hence offers the widest range of optimized libraries. At the end of the chapter, the library for the RL environment will also be selected. However, for most libraries implementing RL algorithms, there is a standard library to use for custom environments. Since the library for the algorithm should support the library of the environment, the standard environment library of the algorithm's library will be used if available.

When selecting a library for the PPO algorithm, the following criteria will be checked: Firstly, the library should be well-documented, so that the implementation does not take longer than necessary and more time can be spend on other tasks. Secondly, as already reasoned in the last paragraph, the library should support a well-documented environment library. Next, the library should show good performance. If available, the performance will be compared for different libraries using ALE as a metric, as done when an algorithm was selected. The last criterion for a suitable library for a RL algorithm is that it should be up-to-date. That means that the library relies on recent ML libraries and therefore uses good optimization methods which contributes to the performance and stability of the algorithm.

Now that the selection criteria have been outlined, the following will introduce some options for libraries for the PPO algorithm following a selection guide by *Simonini* [57] which includes five libraries. Since one of those, *KerasRL*, does not include a PPO implementation, it will not be introduced. For Stable Baselines (SB) both *Stable Baselines 2 (SB2)* and *Stable Baselines 3 (SB3)* will be introduced. After the introduction, a suitable library will be chosen and the respective library for a RL environment will be introduced shortly.

4.5.2 OpenAI Baselines

OpenAI Baselines is a standard library for RL which includes a range of algorithms. It is based on *Tensorflow*, a common ML library. The master branch supports versions 1.4 to 1.14 while the *tf2* branch supports *Tensorflow* 2.0. The library thus fulfills the last criterion. Custom environments can be built following the *Gym* interface of OpenAI. It is available on GitHub [37]. According to *Simonini*, it however lacks documentation and comments in the code. Hence, the library does not fully fulfills the first criterion.

4.5.3 TF agents

TF agents is a library offered by the developers of *Tensorflow*. Therefore, it supports *Tensorflow* versions 1.4.x and 2.0. It is available on GitHub [58] and offers some tutorials. Custom environments can be built following the *PyEnvironments* interface. Although *Simonini* points out the high quality of the implementations of TF agents, he

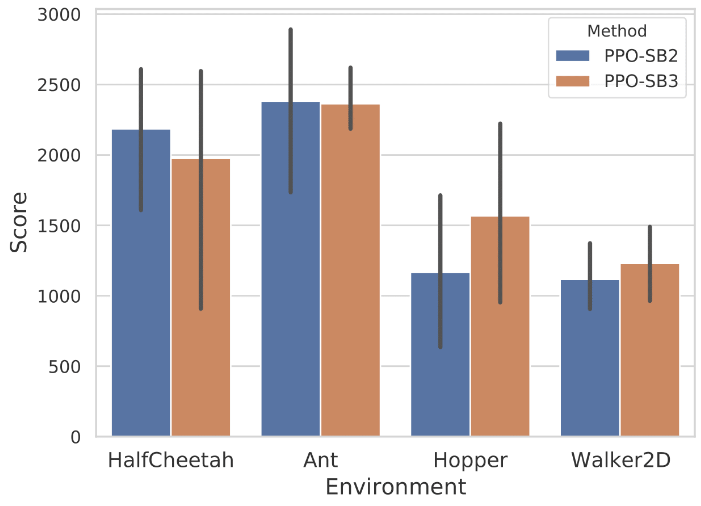


Fig. 4–11: Comparison of the performance of SB2 and SB3 in four Atari games showing similar performance for both libraries [64]

sees a lack of documentation. As the *OpenAI Baselines* library, *TF agents* fulfills the last criterion but does not fully fulfill the first one.

4.5.4 Stable Baselines 2

Stable Baselines 2 (SB2) is a fork of *OpenAI Baselines*, which the developers of *SB2* reported to be unstable, uncommented and written without a common codestyle when they were releasing the initial version of SB [59]. It is available on GitHub [60] and well-documented [38]. Custom environments can be used with the OpenAI *Gym* interface [61]. The library is based on *Tensorflow*. Unfortunately, *SB2* only supports *Tensorflow* versions 1.8.0 to 1.15.0 but not the versions 2.0.0 or above. It thereby does not fully fulfill the last selection criterion. Since the *SB2* library is in maintenance mode as of 30.08.2021, the documentation recommends to use SB3 instead.

4.5.5 Stable Baselines 3

Stable Baselines 3 (SB3) is the most recent version of SB which is based on *Pytorch*, another major ML library, but does not offer all of the algorithms of *SB2*. However, PPO is available in *SB3* and shows similar performance as the version in *SB2* as shown in figure 4–11. *SB3* is also available on GitHub [62] and well-documented [63]. *SB3* also uses OpenAI's *Gym* interface [61] for custom environments. Since *SB3* is based on *Pytorch* versions 1.8.1 or higher, it is up-to-date. According to *Simonini*, *SB3* has many additional features such as vectorized environments.

4.5.6 Tensorforce

Tensorforce is a RL library based on *Tensorflow* that offers an implementation of PPO. It is available on GitHub [65] and is documented [66]. However, according to *Simonini*, the documentation is incomplete. This library thus does not fully fulfill the first criterion.

A benefit of *Tensorforce* is that it supports multiple interfaces for custom environments. Since *Tensorforce* is based on *Tensorflow* 2.5.1, it fulfills the last criterion.

4.5.7 Decision for a programming library

Since *OpenAI Baselines*, *TF agents*, *SB2* and *Tensorforce* do not fully fulfill the defined selection criteria but *SB3* does, *SB3* is selected as the programming library for the implementation. Hence, the RL environment will be implemented using OpenAI's *Gym* interface [61]. It is available on GitHub [61] and a documentation is available [67].

5 Software System Design

This chapter provides a description of the software code developed for this thesis. After an overview, the python code is described in detail.

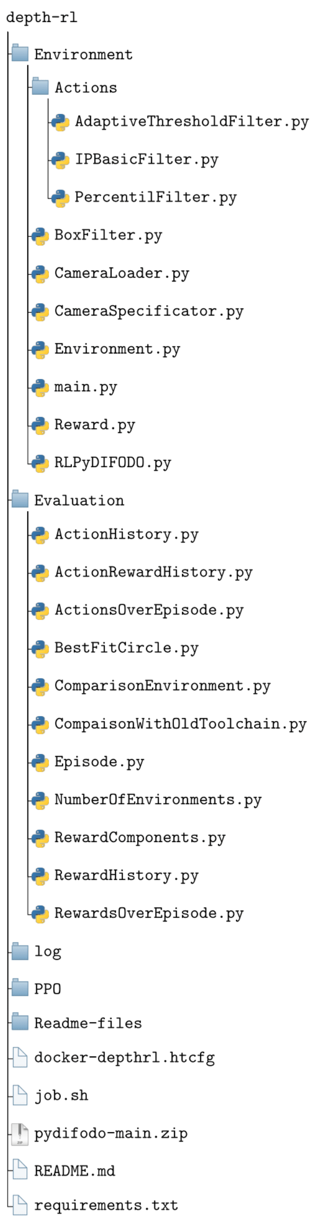
5.1 Overview

The code for the thesis was developed in modules to make it easily reusable for future projects. All files were maintained on the GitLab of the Münchner Wissenschaftsnetz (MWN) and will be made available upon request. A *README.md* is also provided there with additional information about the developed code. Figure 5–1 provides an overview of the code structure.

The python code is stored in the *Environment* and *Evaluation* folders. As the names already suggest, the code for the RL environment and its training is stored in the *Environment* folder while the code for the evaluation of the results is stored in the *Evaluation* folder. A description of the python files within these folders follows below, after the description of the other elements in the code structure.

The *log* folder is empty when passed to the *Condor* GPU cluster and used to store the log, error, and out files. The *PPO* folder is typically empty when passed to the *Condor* GPU cluster and used to store any output of the training, such as the actions chosen, the rewards received and the trajectory as well as the saved agent. In case a model is to be loaded, the model and the saved histories need to be saved in the *PPO* folder and passed to the *Condor* GPU cluster.

The *docker-depthrl.htcfg* file is a description file for the *Condor* GPU cluster which specifies the job to execute. The Bourne shell script *job.sh* is the code that is initially run by the *Condor* GPU cluster and used to install all necessary packages via *pip*. Afterwards it ex-



```
depth-rl
├─ Environment
│  ├─ Actions
│  │  ├─ AdaptiveThresholdFilter.py
│  │  ├─ IPBasicFilter.py
│  │  └─ PercentilFilter.py
│  ├─ BoxFilter.py
│  ├─ CameraLoader.py
│  ├─ CameraSpecificator.py
│  ├─ Environment.py
│  ├─ main.py
│  ├─ Reward.py
│  └─ RLPyDIFODO.py
├─ Evaluation
│  ├─ ActionHistory.py
│  ├─ ActionRewardHistory.py
│  ├─ ActionsOverEpisode.py
│  ├─ BestFitCircle.py
│  ├─ ComparisonEnvironment.py
│  ├─ CompaisonWithOldToolchain.py
│  ├─ Episode.py
│  ├─ NumberOfEnvironments.py
│  ├─ RewardComponents.py
│  ├─ RewardHistory.py
│  └─ RewardsOverEpisode.py
├─ log
├─ PPO
├─ Readme-files
├─ docker-depthrl.htcfg
├─ job.sh
├─ pydifodo-main.zip
├─ README.md
└─ requirements.txt
```

Fig. 5–1: Overview of the code structure

ecutes the python code. The zip-package *pydifodo-main* is used to install the python version of the DIFODO algorithm. The *README.md* contains a description of the project to be displayed on the GitLab site on which the project is stored. The *Readme-files* folder contains the figures used in *Readme.md*. Finally, the *requirements.txt* text file specifies the python packages to install when executing *job.sh*.

See 5–2 for an overview of how the training routine is executed. Please be aware that all files called by the *Environment* class are not included to keep the overview character.

5.2 Description of the python code

The following will explain the purpose of each python file in the code structure, following the order displayed in figure 5–1.

5.2.1 Environment folder

Figure 5–3 shows a class diagram of the environment class and associated classes. It focuses on the most important aspects of the classes, i.e., most attributes and some less important functions are not displayed. To increase understandability, the function returns use a descriptive name for which specifies the parameters passed rather than the datatype.

Within the *Environment* folder, there is a folder called *Action* which contains the actions the RL agent can choose from (as described in chapter 4.4), that needed their own python file.

The *BoxFilter.py* file contains a python class that is initialized with the values of the box filter and has a function called *filter* that takes an image and returns the box-filtered image.

CameraLoader.py contains a python class that is initialized with the path to the training files, the file number and the camera to load images from and that is used to read images and other information out of the *NetCDF4* files that were recorded during earlier studies in the RACOON-Lab by *Franceschini* [6]. Its function *next* returns the next image of the specified file. The function *new* is automatically called when the function is initialized and loads all images of the specified dataset in the memory to speed up the process of loading each single image afterwards. The *reset* function restarts the loading at the first file. The functions *len, getDepthConversionFactor, getOmegaRef, getFPS, getRadius*, and *getDimensions* read the respective information from the dataset.

The *CameraSpecificator.py* is a python class that provides the information used to initialize the *CameraLoader.py*, i.e., the file number, while being also initialized with the path to the training files and the camera. When its function *next* is called, the file number of the next dataset which contains images from the camera specified in the initialization is returned. If the end of the directory with the training files is reached, the *CameraSpecificator* is reset with its function *reset*, i.e., the search continues with the

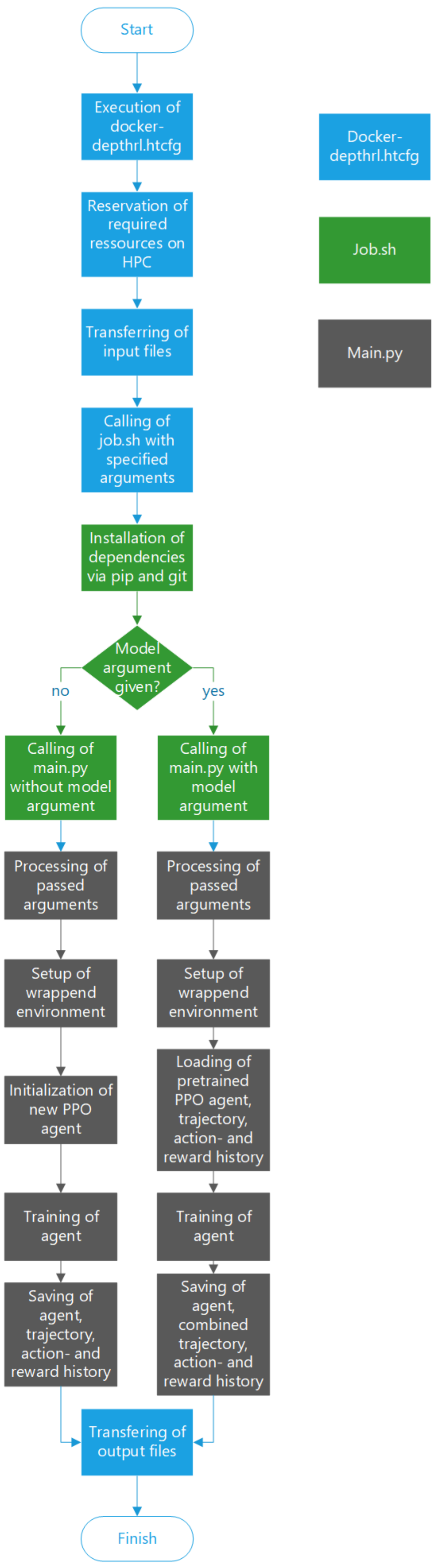

Start
Execution of docker-depthrl.htcfg
Reservation of required ressources on HPC
Transferring of input files
Calling of job.sh with specified arguments
Installation of dependencies via pip and git
Model argument given?
no
yes
Calling of main.py without model argument
Calling of main.py with model argument
Processing of passed arguments
Processing of passed arguments
Setup of wrappend environment
Setup of wrappend environment
Initialization of new PPO agent
Loading of pretrained PPO agent, trajectory, action- and reward history
Training of agent
Training of agent
Saving of agent, trajectory, action- and reward history
Saving of agent, combined trajectory, action- and reward history
Transfering of output files
Finish
Docker-depthrl.htcfg
Job.sh
Main.py

Fig. 5–2: Code flowchart of training routine

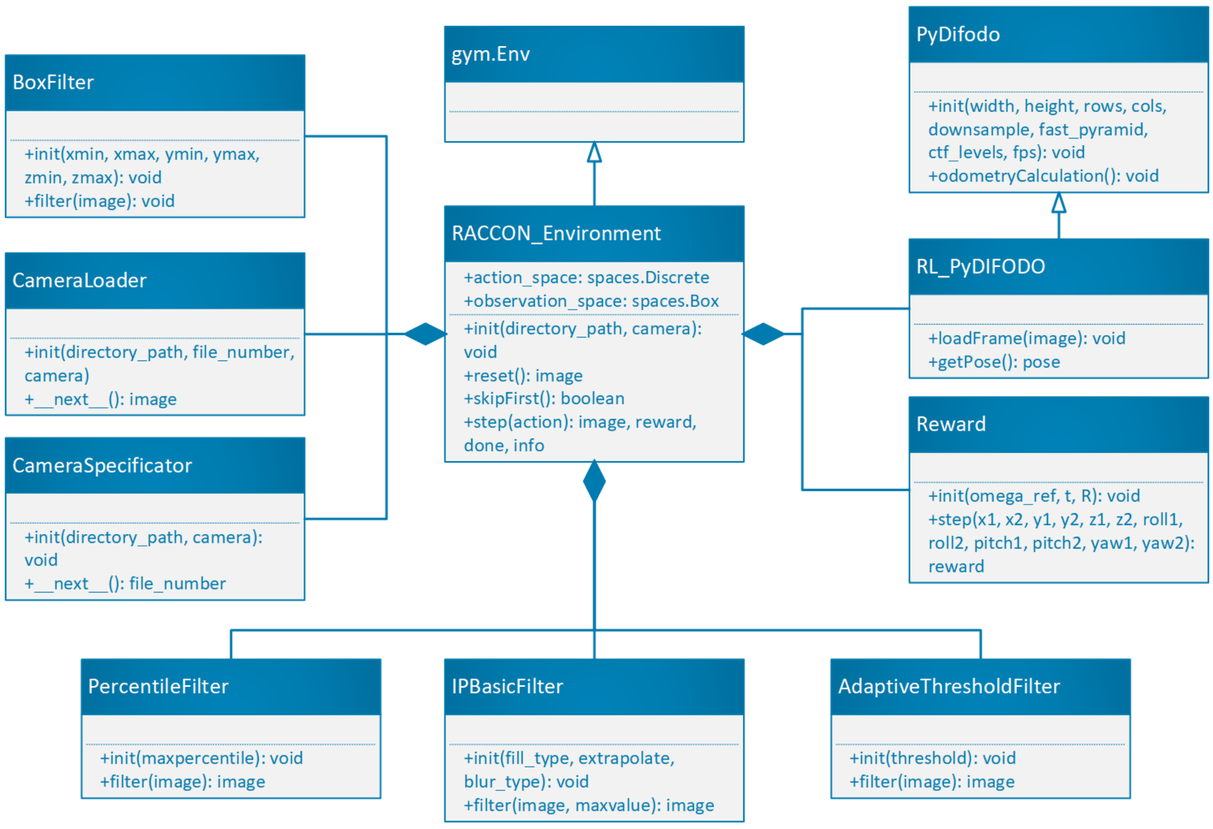

BoxFilter
+init(xmin, xmax, ymin, ymax, zmin, zmax): void
+filter(image): void
CameraLoader
+init(directory_path, file_number, camera)
+__next__(): image
CameraSpecificator
+init(directory_path, camera): void
+__next__(): file_number
gym.Env
RACCON_Environment
+action_space: spaces.Discrete
+observation_space: spaces.Box
+init(directory_path, camera): void
+reset(): image
+skipFirst(): boolean
+step(action): image, reward, done, info
PyDifodo
+init(width, height, rows, cols, downsample, fast_pyramid, ctf_levels, fps): void
+odometryCalculation(): void
RL_PyDIFODO
+loadFrame(image): void
+getPose(): pose
Reward
+init(omega_ref, t, R): void
+step(x1, x2, y1, y2, z1, z2, roll1, roll2, pitch1, pitch2, yaw1, yaw2): reward
PercentileFilter
+init(maxpercentile): void
+filter(image): image
IPBasicFilter
+init(fill_type, extrapolate, blur_type): void
+filter(image, maxvalue): image
AdaptiveThresholdFilter
+init(threshold): void
+filter(image): image

Fig. 5–3: Class diagram of the environment and associated classes

first file in the directory.

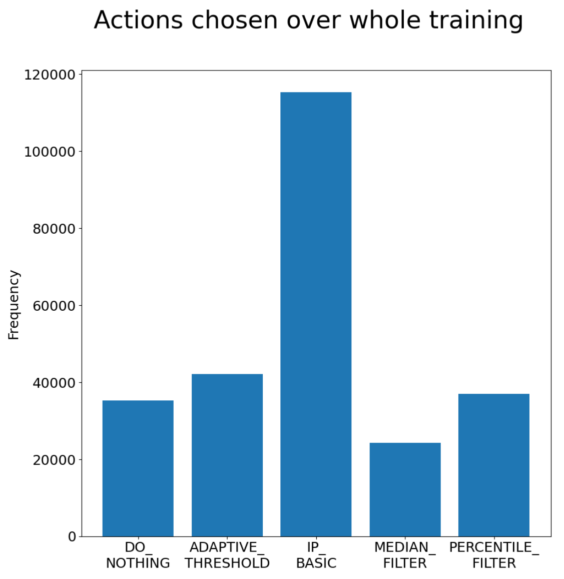
Environment.py contains a python class called *RACOON_Environment* that follows the *Gym* interface for RL environments from which it also inherits. In this class, all components of the environment, i.e., all python files within the *Environment* directory besides the *main.py*, are used. When being initialized, the class defines an action and an observation space. The action space is a discrete action space with the number of actions as length. Other than a continuous action space, a discrete one only allows discrete values, i.e., integers, each of which stands for one action to take. The observation space is an array that can store a RGB image. Since the three cameras in the RACOON-Lab have different resolution and the shape of observation space cannot be updated during the training, the *RACOON_Environment* class gets passed the camera for initialization and once initialized only works with that camera. That means that any agent trained with the *RACOON_Environment* only works with one camera. Despite being inevitable, this restriction also increases the chances of the agent to succeed since the variation of the input images is much lower when only images from one camera are used. During the initialization, the class also defines its *CameraSpecificator, CameraLoader, BoxFilter, AdaptiveThresholdFilter, IPBasicFilter*, and *PercentileFilter*. For evaluation purposes, an *action_history, reward_history, Dtrans_history, Drot_history, Strans_history, Srot_history*, and *trajectory* are also initialized. These are saved after the training is finished by the *main.py*.

In order to follow the *Gym* interface, the *RACOON_Environment* also needs to define a *reset, step, render* and *close* function, while the *render* and *close* function can be empty. The *reset* function needs to reset the environment once it completed an episode. Think of an environment that implements the game tic-tac-toe. Once a game ends, it should be able to reset, i.e., start a new game. In the case of the *RACOON_Environment*, resetting means to get to start a new series of images from a new experiment. The *action_history, reward_history* and *trajectory* get markers when the environment is reset. Also, the *PyDIFODO* and *Reward* are initialized. This needs to happen every time the environment is reset since the *PyDIFODO* cannot be reset and the *Reward* may have new reference parameters to work with. The *reset* function is also automatically called before the training starts. Every time a new image set is started, the first image that is passed to the *PyDIFODO* raises an error which is why the function *SkipFirst*, which excepts that error, is called at the end of the *reset* function. The function returns the next image which is then processed by the RL agent.

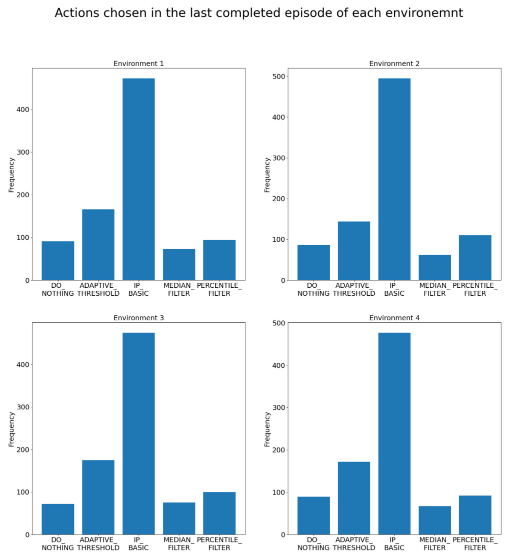
The *step* function is what actually makes the *RACOON_Environment* since in this function the behavior of the environment for each step is defined. The function gets passed an *action* that was chosen by the agent which is also saved in the *action_history*. Next, the action is applied to the image which is then handed to the *PyDIFODO*, which calculates the odometry. Afterwards it is asked for the current pose which is saved in the *trajectory* and used to calculate the reward which is saved in the *reward_history*. In case the reward is NaN, it is set to 0 to prevent errors. The components of the rewards are also saved in individual histories for evaluation. At the end of the *step* function, the next image is loaded and it is checked whether the end of the image set is reached. That determines the value of the variable *done*. The *step* function returns the next image, the reward, the *done* variable as well as a variable *info* which is empty in the case of the *RACOON_Environment*.

main.py is the file that runs the training of the RL agent. It gets passed six required and one optional arguments via the command line: The required arguments are *timesteps, trainingpath, save, camera, envs* and *learningrate*. A pre-saved *model* can optionally be passed to continue the training of the passed model. This setup makes it possible to change the parameters of the training without changing the *main.py* file. Once the parameters are read in, as many *RACOON_Environemnts* as specified by the parameter *envs* are wrapped in a *SubprovVecEnv* which parallelizes the training of the agent. Then a PPO agent is initialized using the passed *learningrate*. Alternatively, the agent from *model* is loaded. The *timesteps* argument is passed to the *learn* function of the agent which is called right after the agents initialization. Once the training is finished, the *action_history, reward_history*, the reward components histories and *trajectory* are saved in the *PPO* folder for later evaluation.

Reward.py contains a python class that calculates the reward as designed in chapter 4.3. For initialization, it takes the parameters *omega_ref, t* and *R* and computes the reference value *gamma_ref* and the reference velocities *v_ref* and *omega_ref*. The function *step* is used to calculate the reward for one step. It takes *x1, x2, y1, y2, z1, z2, roll1, roll2, pitch1, pitch2, yaw1*, and *yaw2* as input parameters and calculates the reward which is returned by the method according to the formulas introduced in



(a) **Actions chosen over all episodes**



(b) **Actions chosen in last episode**

Fig. 5–4: Example of figures created by *ActionHistory.py*

chapter 4.3.

The file *RLPyDIFODO.py* contains a python class that implements the PyDIFODO for the *RACOON_Environment*. It inherits from the *PyDifodo* class and implements the method *loadFrame* which gets passed an image and saves it under *_depth_wf*. Additionally, it has a method *getPose* which returns the *cam_pose*. This method is called by the *RACOON_Environment* and then passed to the reward function.

5.2.2 Evaluation folder

The files in the *Evaluation* folder can be used to evaluate the agent's performance and the toolchain. Some figures distinguish between the environments. Since the trainings in this thesis were run with four parallel environments, the functions are build for four parallel environments. Nevertheless, they can be easily adopted for another number of parallel environments.

ActionHistory.py defines a function with the same name that creates two bar charts using the *action_history*. The first one shows all actions chosen over the whole training in all four environments. The second one displays only the actions in last episode of each environment. The function takes three parameters (*date, camera* and *timesteps*) to specify which file to read in as well as the parameter *save* to specify whether to save the figures in a corresponding folder. See figure 5–4 for an example output.

The *ActionRewardHistory* function defined in *ActionRewardHistory.py* takes the same inputs as the *ActionHistory* function. It creates one boxplot of the reward for each action. See an output example in figure 5–5.

The same inputs are used for the *ActionsOverEpisode* function which is defined in

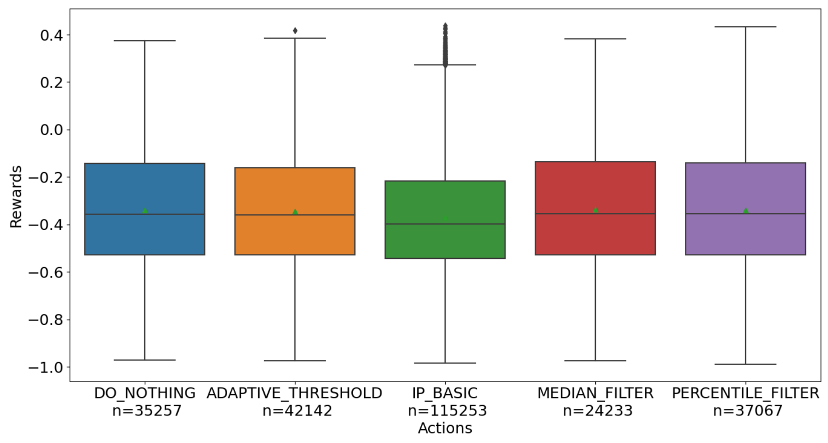


Fig. 5–5: Example of figure created by *ActionRewardHistory.py*

ActionsOverEpisode.py. It creates a figure that displays the actions taken over the last episode of each of the four environments in a stackplot. To do so, a lookback is defined that specifies how many previous timesteps are taken into account when calculating the distribution of actions for the current timestep. This is necessary since without a lookback the distribution of actions would only consist of the action of the current timestep what would not be very informative. See figure 5–6 for an example output.

The function *computeBestFitCircle* which is defined in *BestFitCircle.py* and computes the best-fit plane and best-fit circle for a trajectory which is passed to it. To do so, it follows the procedure described by *Rehn* in [16].

The *ComparisonEnvironment* class is defined in the *ComparisonEnvironment.py* file and inherits from the *RACOON_Environment*. It runs the typical environment but adds a version of the old toolchain without RL from which it saves the trajectory. This environment is used to compare the new and the old toolchain of the RACOON-Lab. It was not tested in this thesis but may still be helpful for future projects.

The *ComparisonWithOldToolchain.py* file compares the new and the old toolchain of the RACOON-Lab. For each completed episode it computes the best-fit circle for both toolchains. It also creates a boxplot of the errors of both toolchains. The code was not tested in this thesis but may still be helpful for future projects.

Episode.py defines a function which creates figures for one specific episode. It takes the number of the episode to analyze as well as three parameters (*date, camera* and *timesteps*) to specifying which file to read as arguments. For the episode, it then creates an action histogram like figure 5–4b, an actions over episode figure like figure 5–6 and a reward histogram like figure 5–9.

NumberOfEnvironments.py offers the option to create a figure showing the performance measured in timesteps per second over the number of environments. The data needs to be put in manually. Figure 5–7 shows a figure created using *NumberOfEnvironments.py*.

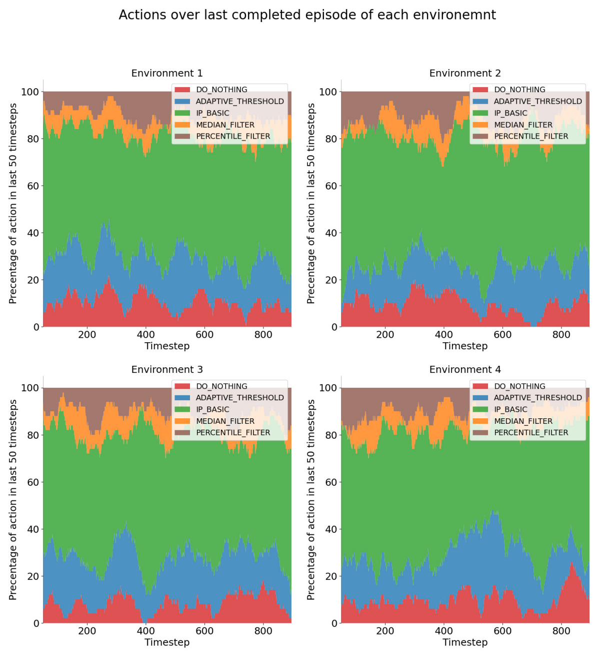


Fig. 5–6: Example of figure created by *ActionsOverEpisode.py*

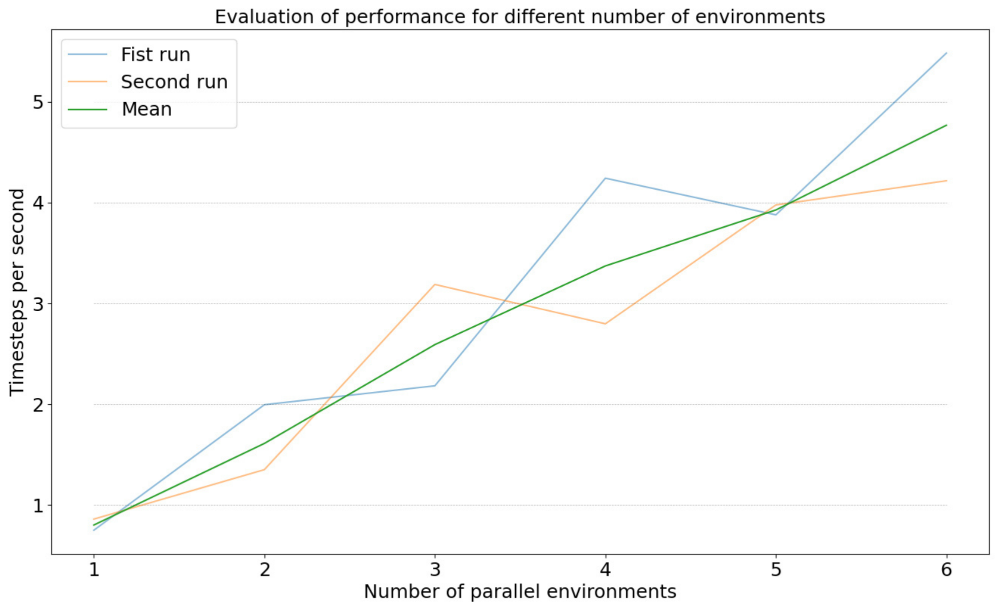
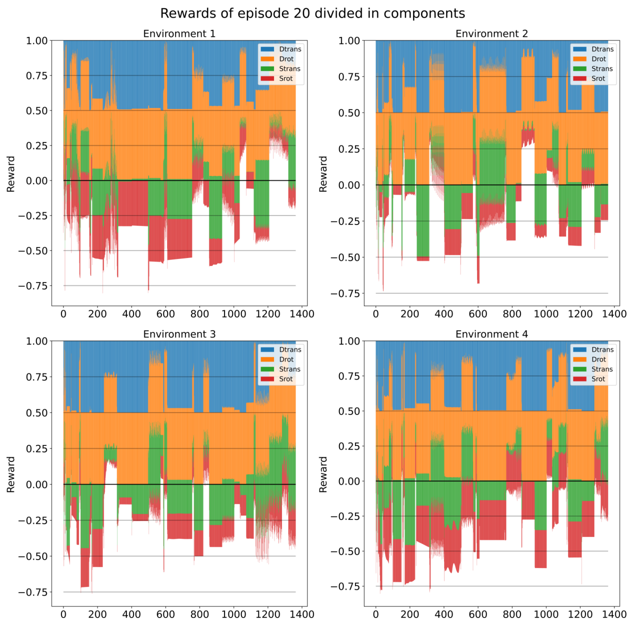


Fig. 5–7: Example output of *NumberOfEnvironments.py*

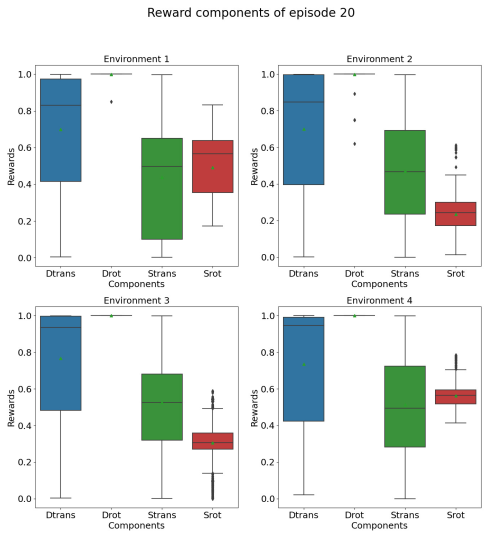
The reward components can be analyzed using *RewardCompoents.py*. The function takes an episode-number as well as three parameters (*date, camera* and *timesteps*) to specifying which file to read in. It creates three figures: The first shows the reward components of each timestep of the episode stacked, so that the total reward can also be observed, for the four environments. The second is a boxplot of the reward components observed during the specified episode. The third figure is also a boxplot of the reward components but for the whole training. Figure 5–8 shows examples of the figures created by *RewardComponents.py*

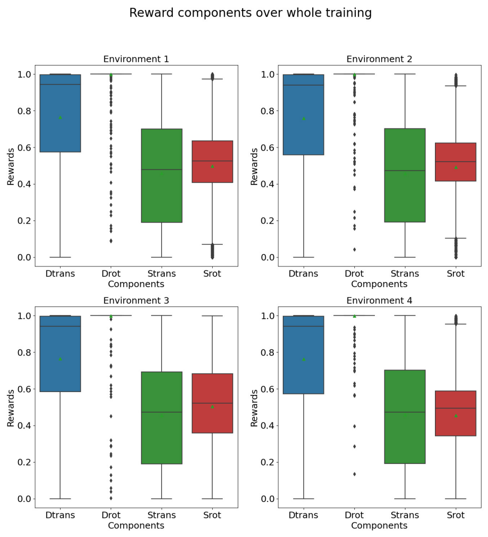
Within the file *RewardHistory.py* an eponymous function is defined which again takes the same inputs as the three functions *ActionHistory, ActionRewardHistory* and *ActionsOverEpisode* described above. This function creates four figures. The first is a histogram of the rewards for the whole training and all environments. The second one displays a histogram of the rewards received in the first episode of each environment. The third figure does the same for the last episode. The last figure is a area chart that displays the mean reward for each episode for all four environments. See an example output in figure 5–9.

The file *RewardsOverEpisode.py* contains a python function which takes an episode-number as well as three parameters (*date, camera* and *timesteps*) to specifying which file to read in. It creates a figure showing the rewards over the specified episode of each environment. Figure 5–10 shows an example of the figure created by *RewardsOverEpisode.py*.



(a) Reward components over episode

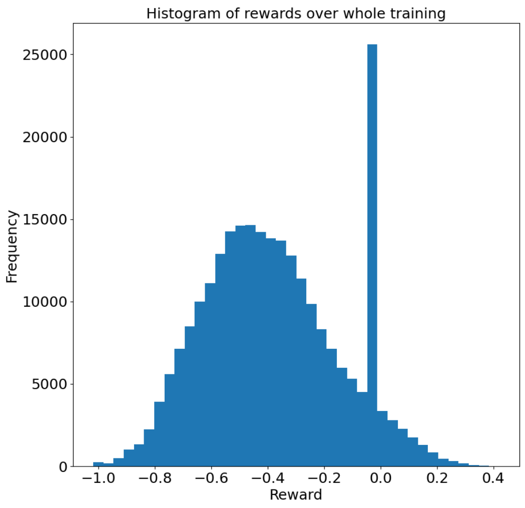




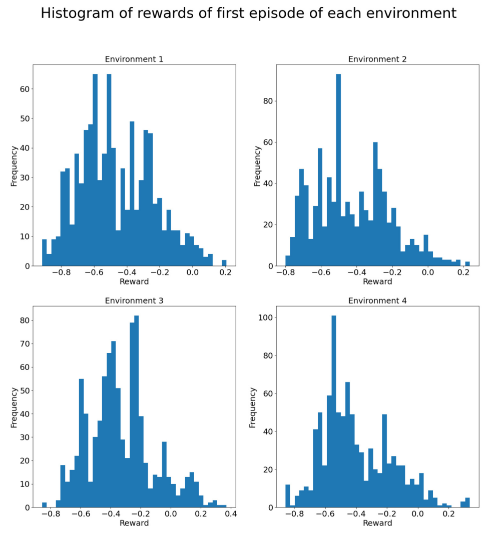
(b) Boxplot of reward components of single episode

(c) Boxplot of reward components of whole training

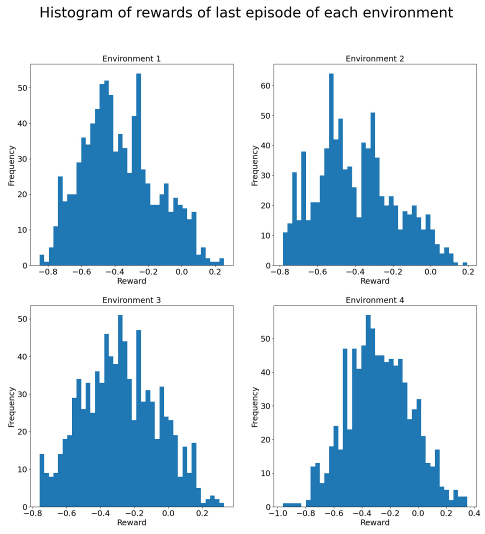
Fig. 5–8: Example of figures created by *RewardComponents.py*



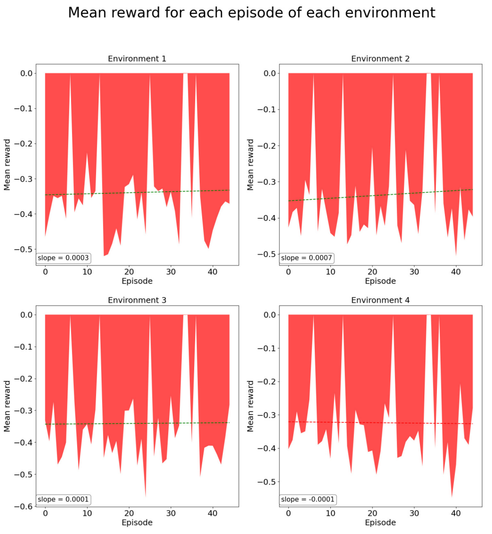
(a) Rewards for whole training



(b) Rewards for first episode of each environment



(c) Rewards for last episode of each environment



(d) Mean reward for each episode of all environments

Fig. 5–9: Example of figures created by *RewardHistory.py*

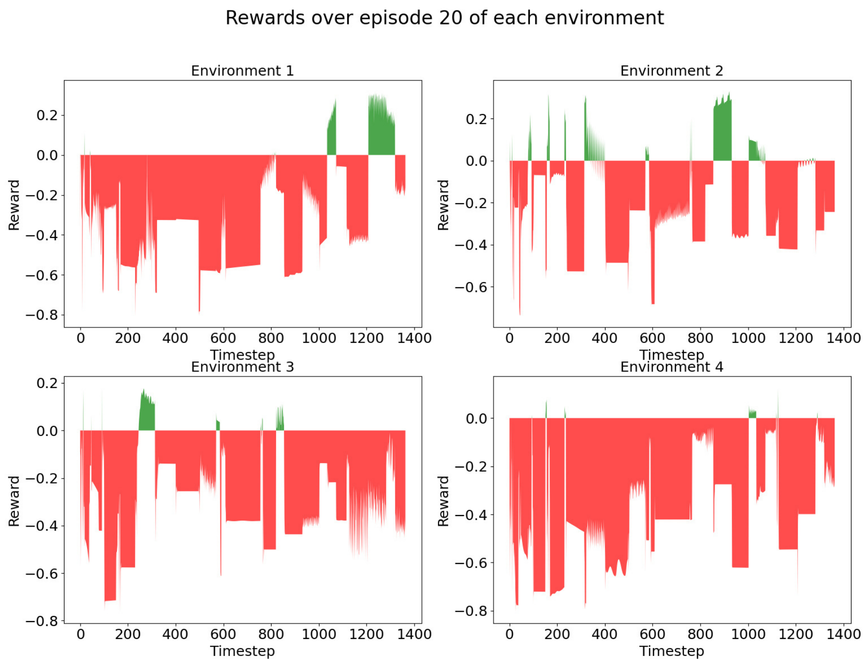


Fig. 5–10: Example of figure created by *RewardsOverEpisode.py*

6 Results

This chapter will describe the results of the training. The trainings were performed with 43 recordings made by *Franceschini* in a previous study of the RACOON-Lab [6]. Since they included a wide range of parameters, a detailed description of the parameters is omitted. Two datasets will be analyzed. As described in the objective in chapter 3, the purpose of this analysis is to demonstrate the capabilities of the figures to evaluate the RL agent's performance and the toolchain. Additionally, two parameters (the number of parallel environments and the learning rate) will be iterated to demonstrate how the training can be optimized.

The data presented in the following was recorded with an error in the calculation of ω which influenced the rotational components of the reward $d_{rotation,norm}$ and $s_{rotation,norm}$. Instead of taking the difference in the yaw, pitch, and roll angles between two timesteps to calculate ω as shown in equation 4–32, the absolute values of the current timestep were taken. Although the rotational components of the reward were not calculated correctly, the demonstration of the capabilities of the figures is still possible with the data. After the description of the datasets and before the iteration of parameters, a corrected version of the component boxplots and the rewards over episode figures will be given for one of the datasets. Chapter 7.2.1 discusses the impact of the error on the validity of the results.

6.1 Dataset 21-11-11 04-41-55

The first dataset which will be analyzed was the result of a training started with the parameters listed in the upper part of table 6–1. The bottom part of the table shows parameters observed during the training.

Tab. 6–1: Parameters of dataset 21-11-11 04-41-55

parameter	value
camera	Intel Realsense D435
timesteps	250000
number of parallel environments	4
fps	3.47
iterations	31
time elapsed	73199 s
total timesteps	253952
learning rate	0.0003
loss	0.0421
n_updates	300

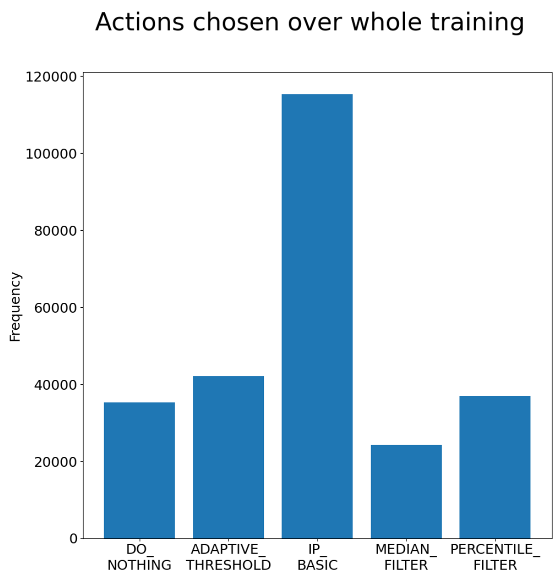


Fig. 6–1: Actions chosen over whole training of dataset 21-11-11 04-41-55 showing the *IP_Basic filter* to be most dominant

In order to prevent errors stopping the training, NaN rewards were overwritten with the value 0 in episodes 5, 6, 13, 25, 33, and 34 in all four environments. In each case where a reward was set to 0, the components had the following values: $D_{trans} = 1.0, D_{rot} = 1.0, S_{trans} = NaN, S_{rot} = NaN$.

6.1.1 Overview of results

The following will give an overview of the results of the dataset. Interesting episodes will be identified and analyzed in more detail afterwards.

Figure 6–1 shows the number of times the agent chose each action over the whole training. The *IP_Basic filter* is by far the most dominant one here being chose more than twice as often as the second-most chosen *Adaptive Threshold filter*. The actions *Do nothing* and *Percentile filter* were chosen almost equally often and a bit less often than the *Adaptive Threshold filter*. The *Median filter* was chosen the fewest, only about half as often as the actions *Do nothing* and *Percentile filter*.

The same observations but more pronounced can be made for figure 6–2 showing the actions chosen in the last episode of each environment. In all environments, the *IP_Basic filter* is chosen about three times as often as the second-most chosen action which again is the *Adaptive Threshold filer* for all four environments. Also quite interesting is that all four environments have only few differences. The most noticeable difference is that in environments 1 and 4 the *Do nothing* action was chosen slightly more often than the *Percentile filter*, while in environments 2 and 3 it is vice versa.

A detailed look on the actions chosen over the last episode of each environment as available in figure 6–3 also underlines the clear dominance of the *IP_Basic filter*. The relative share of all actions is fluctuating heavily. In Environments 1 to 3, there is also a

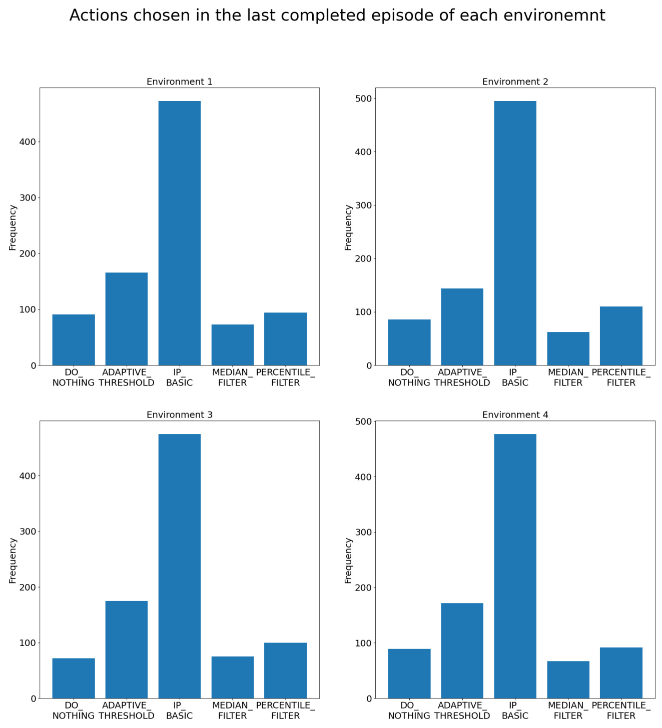


Fig. 6–2: Actions chosen in the last episode of each environment of dataset 21-11-11 04-41-55 showing the *IP_Basic filter* to be by far the most dominant

decline in its relative share around timestep 300 for about 50 timesteps. In environment 4, the decline is even greater but around timestep 550 for about 100 timesteps.

Figure 6–4 shows a boxplot of the rewards categorized by actions. Remember that the reward function designed in chapter 4.3 allows rewards in the range from -1 (worst) to $+1$ (best). The median reward for all five actions lies a little above -0.4 with the median of the rewards received after application of the *IP_Basic filter* is a little lower than for the other actions as is the first quartile and the third quartile. The same is true for the mean. The interquartile range is also smaller for the *IP_Basic filter* compared to the other actions which all have approximately the same interquartile range. The maximum boxplot value of the *IP_Basic filter* of about 0.275 is about 0.1 lower than the other actions'. For the *IP_Basic filter* there are many outliers above the maximum boxplot value while there is only one outlier for the other actions which is above the maximum boxplot value of the *Adaptive Threshold filer*. The minimum boxplot value for all actions is at -1.0.

A more detailed look on the rewards received over the whole training is provided by figure 6–5. It looks approximately like a normal distribution around the value -0.4 where it reaches about 15000 occurrences. The right side of the distribution however declines a little slower than the left side rose. At the value of 0 the occurrences jump to 25000 just to continue the declining trend right afterwards. The maximum reward received is approximately 0.4 but has occurred only a few times. The minimum reward

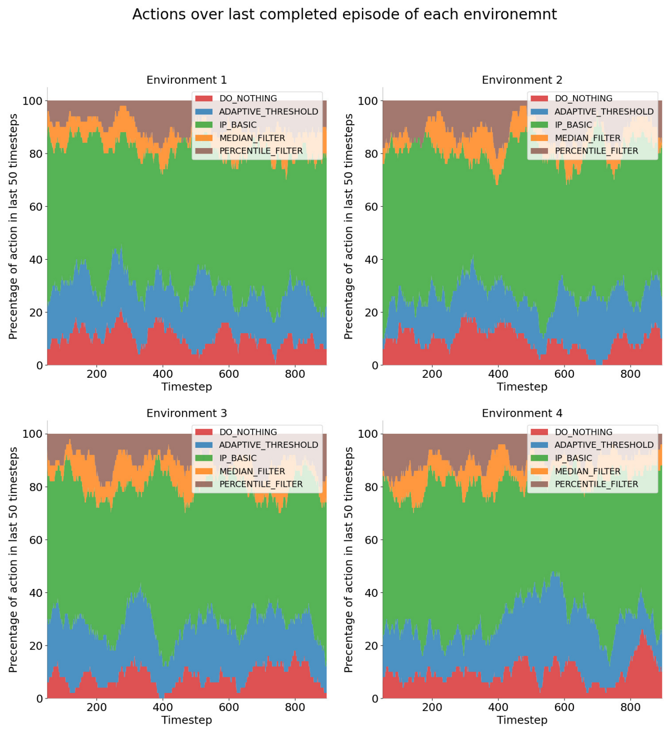


Fig. 6–3: Detailed look on actions chosen over the last episode of each environment of dataset 21-11-11 04-41-55

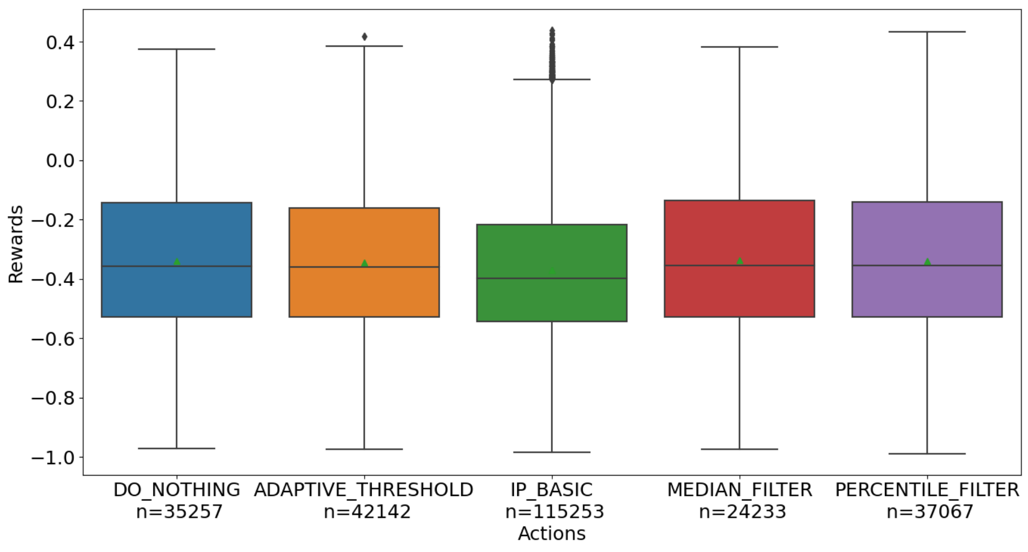


Fig. 6–4: Rewards over actions for dataset 21-11-11 04-41-55 showing many outliers for the *IP_Basic filter*

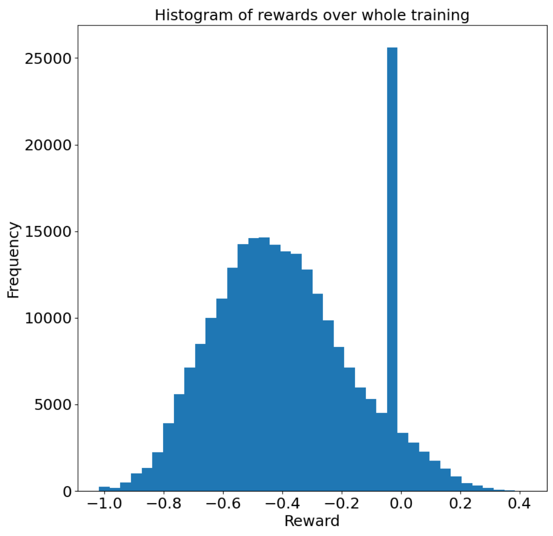
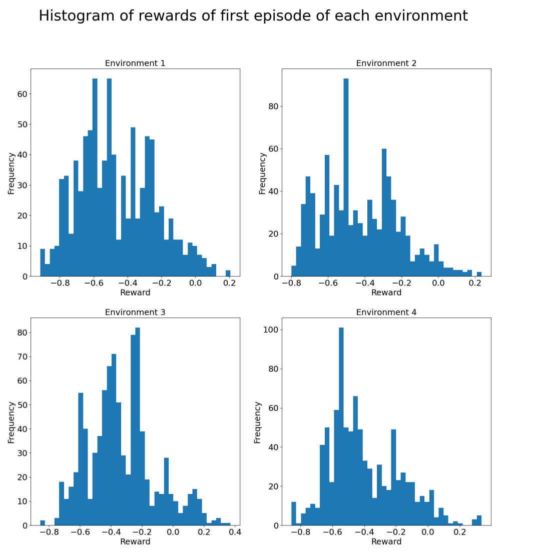


Fig. 6–5: Histogram of reward received over whole training of dataset 21-11-11 04-41-55 which looks approximately like a normal distribution around -0.4 besides an outlier at 0

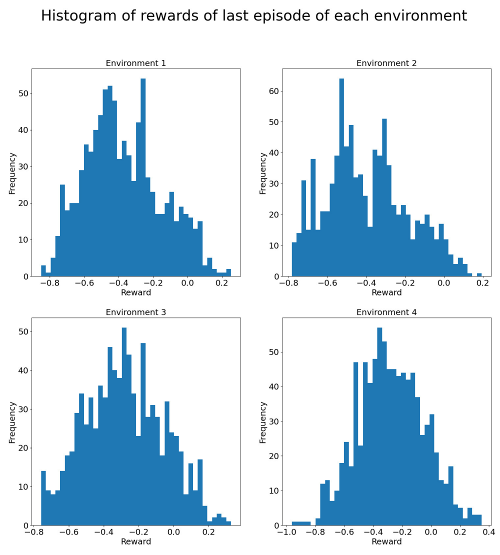
received of -1.0 also occurred rarely but much more often than the maximum one.

Looking on the rewards received in the first and last episode of each environment in figure 6–6, the normal distribution observed in figure 6–5 can only be partially identified. Especially for the rewards received in the first episode of each environment as shown in figure 6–6a a normal distribution cannot be identified. The maximum value is reached at -0.5 in environments 1, 2, and 4 and at -0.2 in environment 2 and therefore still in the middle part of the histograms. Figure 6–6b showing the rewards received in the last episode of each environment looks much more like a normal distribution, however not as clearly as figure 6–5. These distributions reach their maximum around -0.3, so a bit higher than the -0.4 for the whole training. The rewards for the environments displayed in figure 6–6 range between -0.9 and 0.3, a slightly smaller range than the rewards for the entire training. Only the rewards received in the last episode of environment 4 exceed these values and both of them at the same time. The jump at the value of 0 as observed in figure 6–5 cannot be identified in any of the episodes displayed in figure 6–6.

In figure 6–7, the reward components of each environment are shown in a boxplot. In most environments, the values for the four components take the full range of 0 to 1. Only the D_{rot} component does not take the value 0 in environment 1, 2, and 4 but only in environment 3. This component also has a median and mean value of approximately 1 in all four environments, the highest values reached by all components. Hence, all values below 1 are categorized as outliers. The number of outliers for the D_{rot} component varies between the four environments but in all environments there are several. The component with both the second-highest mean and median is the D_{trans} component. Its median ranges around 0.95 while the mean takes values of



(a) First episode of each environment



(b) Last episode of each environment

Fig. 6–6: Rewards received in first and last episode of each environment of dataset 21-11-11 04-41-55

approximately 0.75 in all four environments. Since the minimum value of the boxplot is 0, there are no outliers in any of the environments. The component with the third-highest mean and median is the S_{rot} component. Its median is approximately 0.5 and the mean slightly lower – approximately 0.45 – in all four environments. In environments 1, 2, and 4, the maximum of the boxplot of the S_{rot} component is lower than 1 which makes it the only component with that characteristic. In environments 1 and 2, the minimum of the boxplot of the S_{rot} component is higher than 0 which makes it the only component besides the D_{rot} component with that characteristic. In all cases where the minimum or maximum of the boxplot is not 0 or 1, there are outliers that exceed the minimum or maximum which also reach values of 0 or 1. The interquartile range of the S_{rot} component is the second-lowest after the D_{rot} component's, where the interquartile range is approximately 0. The S_{trans} component has not only the lowest mean and median value in all four environments with both ranging slightly below 0.4, but also the highest interquartile range. Its minimum and maximum values of the boxplots take the values 0 and 1 in all four environments.

Figure 6–8 displays the mean reward over the episodes of each environment which is negative in all cases. In all environments, the mean reward is fluctuating. An overall trend cannot be observed although the linear regression straight is ascending in environments 1 to 3 since the slope of the regression grade is near 0 for all four environments. The plots look quite similar for all four environments. All environments have 6 episodes (episode number 6, 13, 25, 33, 34, 36) with a mean reward of about 0 which clearly stand out from the other episodes where the highest reward is approximately -0.2 in episode 20 of environment 2. These episodes will be analyzed in more detail below.

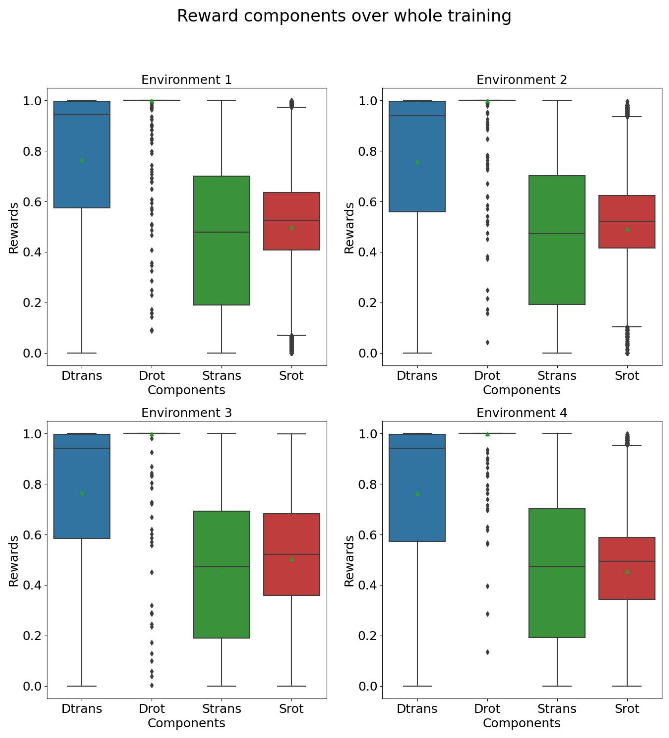


Fig. 6–7: Boxplot of reward components of each environment of dataset 21-11-11 04-41-55

6.1.2 Detailed analysis of interesting episodes

6.1.2.1 Episodes with outstanding high mean rewards of about 0

Figure 6–9 is a histogram of the rewards received in episode 6 of each environment. It clearly shows that all rewards received were about 0. In fact, the chart looks exactly the same for all other episodes with outstanding high mean rewards. Thus, they will not be analyzed further.

6.1.2.2 Episode 20 of environment 2

Figure 6–10 shows the rewards received in episode 20 of each environment. In contrast to the rewards received for the episodes with outstanding high mean rewards of about 0 shown in figure 6–9, the rewards take more than one value. As for figure 6–6, no normal distribution can be identified for any of the episodes in figure 6–10. Also, the mean reward of environment 2 is clearly higher than the one of the other environments since the highest bar is around the value -0.2 here compared to -0.6 for environment 1 and -0.4 for environment 3 and 4. Also, there is a high concentration of rewards around the value 0 in environment 2 which is not present in the other environments. Rewards above 0.2 were only received in environments 1 and 2 but more so in environment 2. Rewards smaller than -0.6 were received the least often in environment 2. The maximum reward received is about 0.3 in environments 1 and 2 while being

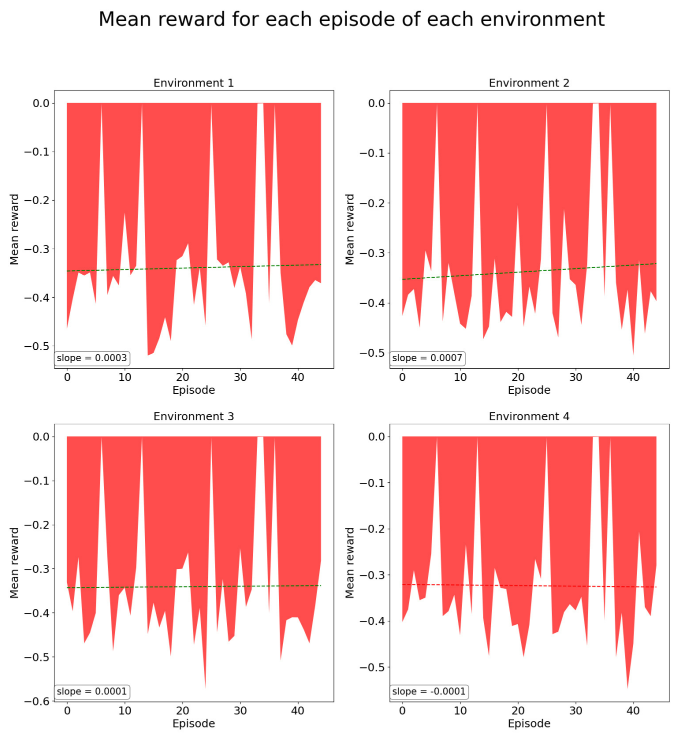


Fig. 6–8: Mean reward for all episodes of each environment of dataset 21-11-11 04-41-55

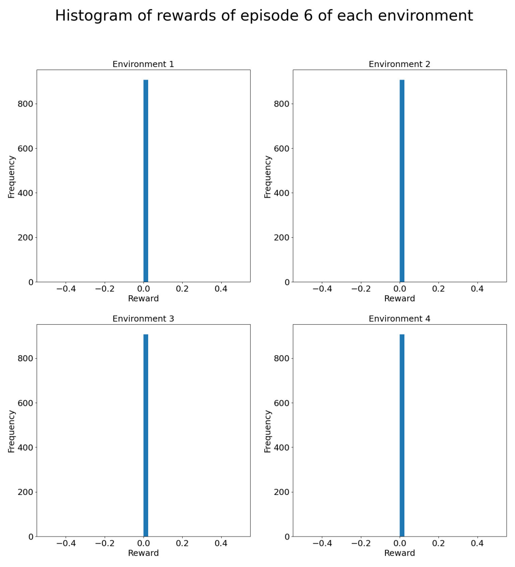


Fig. 6–9: Rewards received in episode 6 of dataset 21-11-11 04-41-55

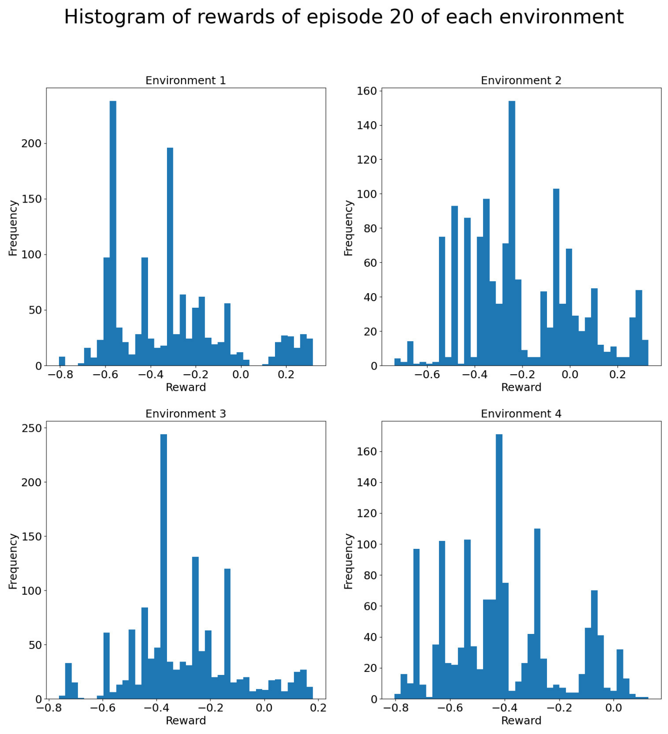


Fig. 6–10: Rewards received in episode 20 of dataset 21-11-11 04-41-55

only slightly lower than 0.2 in environment 3 and only about 0.1 in environment 4. The maximum negative reward is the lowest in environment 2 with a value of about -0.7 compared to about -0.8 for the other environments.

A boxplot of the reward components of episode 20 is shown in figure 6–11. The greatest difference of environment 2 compared to the other environments and the boxplot of the whole training as shown in figure 6–7 is the low mean and median values of the S_{rot} component which are slightly above 0.2. The maximum of the boxplot for the S_{rot} component is only approximately 0.4 and the maximum outlier only 0.6. Therefore, in environment 2 and also in environment 3, the D_{rot} component has the lowest mean and median value of all components. As for the whole training, it has the second-lowest interquartile range following the D_{rot} component in all four environments. The D_{trans} component in environments 1 and 2 have lower mean and median values than the one of the whole training with the median being slightly above 0.8 and the mean approximately 0.7. The D_{rot} component has a mean and median of 1 in all four environments as already observed for the whole training.

The actions chosen in episode 20 for each environment are displayed in figure 6–12. The distribution looks almost the same for environments 1 to 3 with the *IP_Basic filter* being the most-chosen one with about 400 occurrences and the actions *Do nothing* and *Percentile filter* being the second-most chosen ones with about 275 occurrences each. The *Median filter* was the fewest chosen one with about 200 occurrences in all three environments. In environment 4, the distribution is slightly different with the *Adaptive*

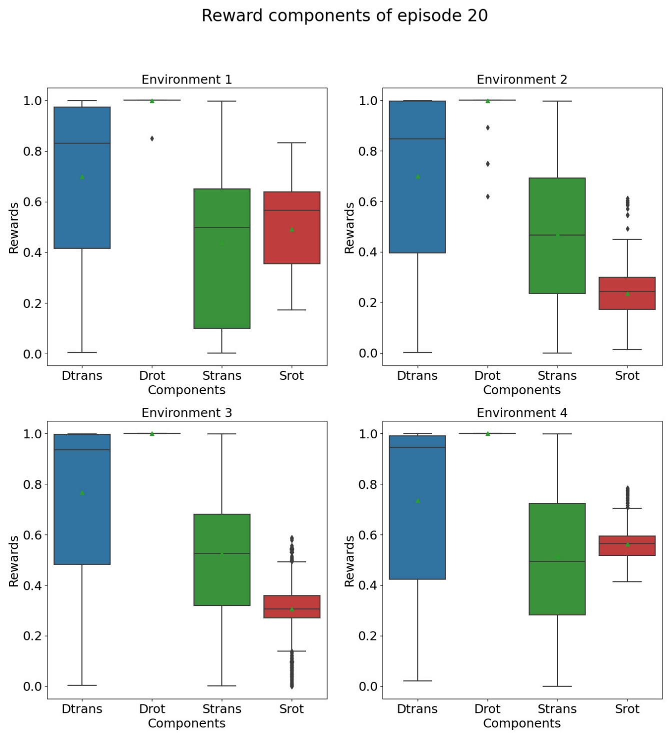


Fig. 6–11: Boxplot of reward components of episode 20 of each environment of dataset 21-11-11 04-41-55

Threshold filter sharing the second place with the *Percentile filer* with about 275 occurrences each while the *Do nothing* action is in fourth place. Also in environment 4, the *Median filter* is the fewest chosen one, this time with about 150 occurrences.

With figure 6–13, the timestep of episode 20 in which the actions were chosen can be analyzed. As in figure 6–3, the relative share of all actions is fluctuating heavily. Although all environments look slightly different, observations only true for environment 2 can hardly be found. The characteristic increase of the *Do nothing* action in the last timesteps for example can also be found in environment 3 although not as strong as in environment 2. The strong decrease of the *Adaptive threshold filter* at timestep 1100 also exists in environment 1. The reasons for that decrease, however, are different. In environment 1, the decrease is mainly caused by a strong increase of the *Do nothing* action's relative share whereas in environment 2, it is mainly caused by an increase of the share of the *Percentile filer*.

Figure 6–14 shows the rewards of each environment over the timesteps of episode 20. The higher mean reward of environment 2 and the lower mean reward for environment 4 compared to the other environments as observed in figure 6–8 can be further analyzed with this figure. In contrast to figure 6–13, clear differences between the environments can be found in this figure.

In all environments, the negative rewards dominate and also reach higher absolute values than the positive ones. Environment 2 has the highest proportion of positive

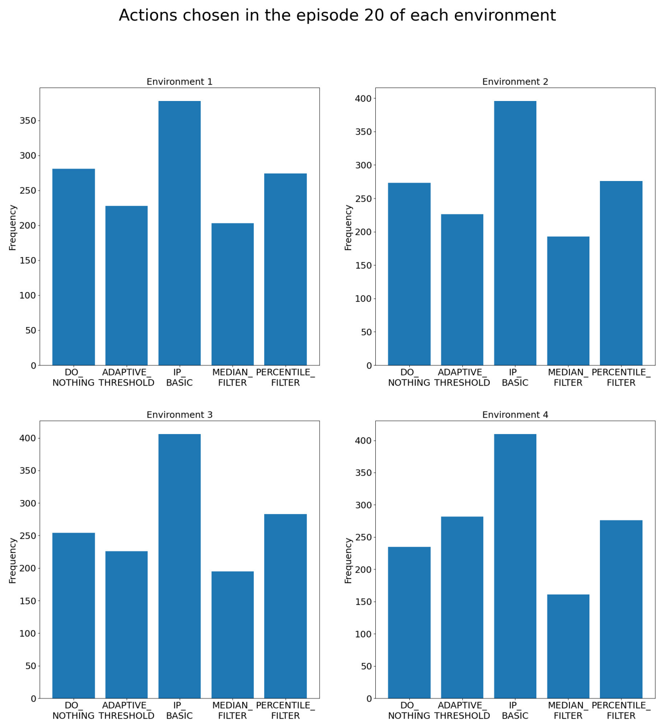


Fig. 6–12: Actions chosen in episode 20 of each environment of dataset 21-11-11 04-41-55 showing similar distribution for environments 1 to 3

rewards, followed by environment 1, shortly followed by environment 3. Environment 4 has only few positive rewards and hence the lowest proportion of positive rewards of all environments. The maximum positive reward in environment 4 is also low compared to other environment, with a value of about 0.1, as also observed in figure 6–10. Environment 2 does not only have the highest proportion of positive rewards, it also has the most segments of positive rewards, i.e., the most series of ongoing positive rewards. The first of these segments in environment 2 are relatively short compared to the long ones around timestep 900 and 1100 but about as long as most of the segments in other environments. The longest segment, however, is around timestep 1300 in environment 1 and lasts for about 100 timesteps. The positive segments of the different environments barely correlate. Interesting is a segment around timestep 350 of environment 2 where positive and negative rewards alternate with high frequency. In this segment, the absolute value of the positive rewards is greater than that of negative rewards. The overall trend of this alternating segment is decreasing, i.e., the last positive and negative rewards are smaller than the respective first ones. A correlation between the positive segments and the actions chosen during these segments in the respective environment cannot be identified.

The negative rewards in environment 2 less often take high absolute values compared to the other environments. Also, the maximum negative reward has a higher absolute value than the maximum negative rewards of the other environments as already observed in figure 6–10. In all environments, the positive rewards have changing val-

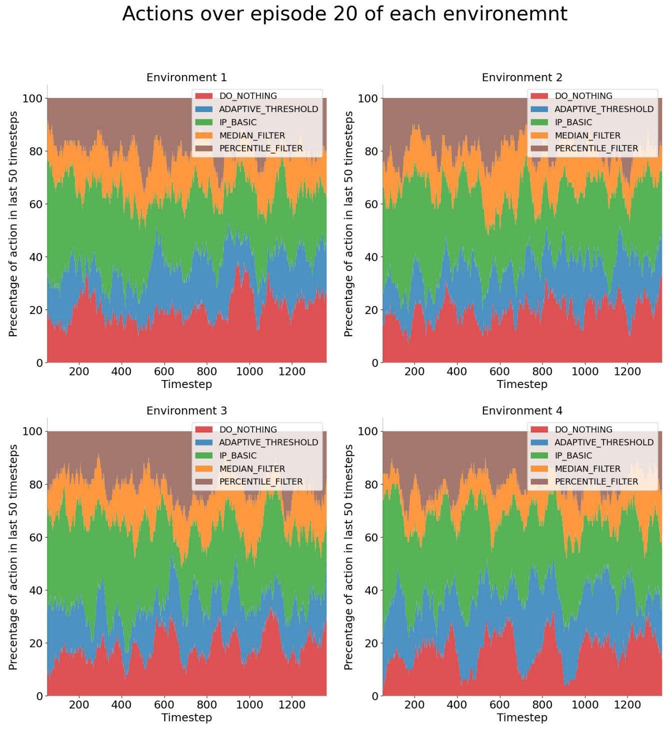


Fig. 6–13: Detailed look on actions chosen over episode 20 in each environment of dataset 21-11-11 04-41-55

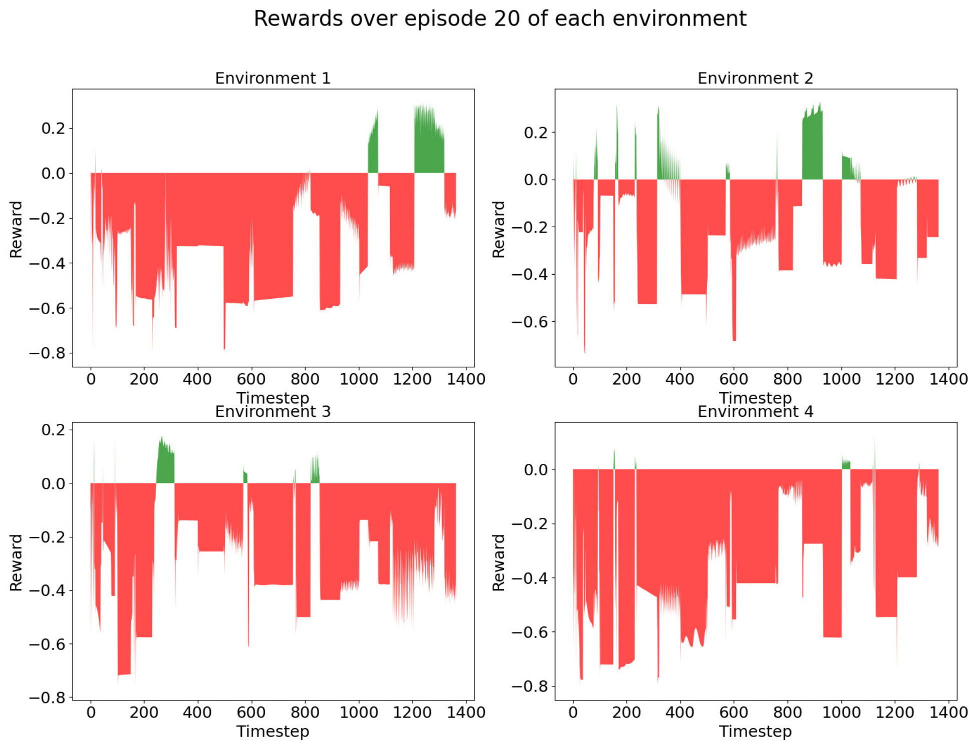


Fig. 6–14: Rewards over episode 20 of each environment of dataset 21-11-11 04-41-55

ues. For the negative rewards there are segments in which the reward is approximately constant for 50 to 150 timesteps, for example around timestep 700 in environment 4. Negative reward segments with changing values are rare, the segment around timestep 700 in environment 2 is an example. Often the first negative rewards after a segment of positive rewards take a greater absolute value than the following negative segment. See the small peak at timestep 300 in environment 3 at the start of the negative segment as an example. However, there are also segments with different behavior, for example the negative segment starting at timestep 300 in environment 4 which has an increasing absolute value.

Using figure 6–15, the rewards received over episode 20 of each environment as shown in figure 6–14 can be analyzed in more detail. The figure shows the rewards divided in its four components which are subtracted from a starting value of 1. Please be aware that the component values displayed here are only half of the values shown in figure 6–11 to make the total reward range between -1 and 1. As already observed in figure 6–11, the D_{rot} component is constantly high over all timesteps leading to a subtraction of 0.5. In contrast to that, the other components take a range of values as also observed in figure 6–11. For timesteps with positive rewards, these three components typically all have relatively low values. The D_{trans} component often takes values near 0 for the timesteps where the combined reward is positive. Compared to its typical subtraction of 0.4 or greater this has a huge influence on the combined reward. However, there are also timesteps where the D_{trans} component is approximately 0 and the combined reward is still negative such as timestep 600 in environment 1. In many timesteps, a positive reward is already impossible after the subtraction of only the D_{trans} and D_{rot} components as for example from timestep 400 to 550 in environment 2. The S_{trans} component is even more often approximately 0. However, since its typical value is lower, its influence on the positive rewards is slightly lower. The S_{rot} component causes the lowest subtraction as already observed in figure 6–11. However, it still ranges between approximately 0 and 0.3. The constant negative segments observed in figure 6–11 are caused by all four components being constant in these segments. Fluctuating rewards in positive seem to be mainly caused by the D_{trans} component as for example visible around timestep 700 of environment 2.

6.2 Dataset 21-11-12 14-18-56

The next dataset which will be analyzed was started with the parameters listed in the upper part of table 6–2 and returned the parameters observed during training as listed in the bottom part of the table. Please note that the learning rate was set to 0.00005 for this training.

In this dataset, no NaN reward occurred, so none was overwritten.

6.2.1 Overview of results

The following will give an overview of the results of this dataset. Interesting episodes will be identified and analyzed in more detail afterwards.

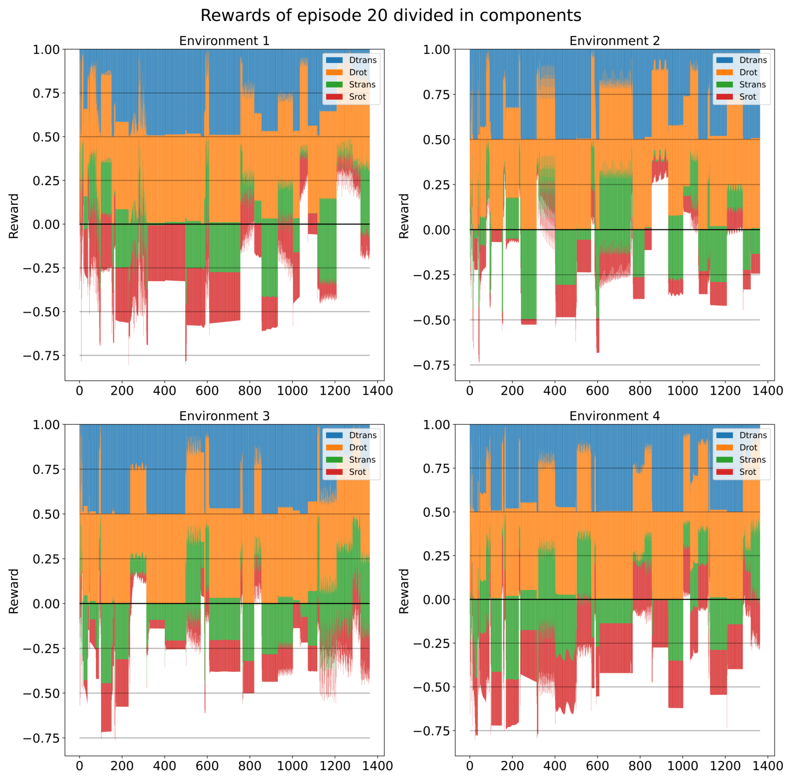


Fig. 6–15: The rewards over episode 20 of each environment of dataset 21-11-11 04-41-55 divided in components

Figure 6–16 shows the actions chosen over the whole training. As in figure 6–1, the *IP_Basic filter* clearly stands out as the most-chosen action. It was chosen more than twice as often as the second-most chosen action, the *Median filter*. The actions *Do nothing* and *Adaptive Threshold* follow with approximately 20% less occurrences compared to the *Median filter*. The *Percentile filter* was chosen the least often.

The actions chosen in the last episode of each environment are shown in figure 6–17. In contrast to figure 6–2, the dominance of the *IP_Basic filter* is less extreme in the last episode compared to the whole training. It is still the most chosen action in every environment. However, it was chosen only 1.5 times as often as the second-most chosen action which is the *Adaptive threshold filter* for environments 1 and 2 and the *Median filter* for environment 3 and 4. For environments 1 and 2, the *Median filter* follows as the third-most chosen action. For environments 3 and 4, the *Do nothing* action was the third-most chosen one. The least chosen action was the *Percentile filter* for environments 1, 3, and 4 and the *Do nothing* action for environment 2. These actions were still chosen more than half as often as the most chosen *IP_Basic filter* – another large contrast to figure 6–2 where the least chosen actions were chosen about a tenth as often as the most chosen one.

As already observed for other figures showing the actions over an episode for each environment, the relative shares in figure 6–18 showing the actions over the last completed episode of each environment are fluctuating a lot. The strong increase of the

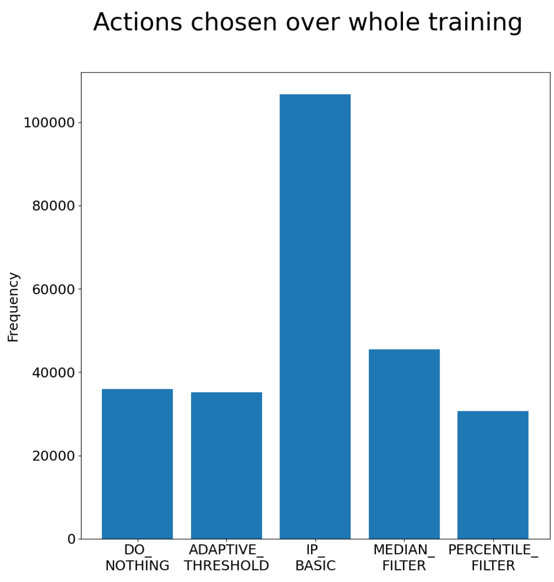


Fig. 6–16: Actions chosen over whole training of dataset 21-11-12 14-18-56 showing the *IP_Basic filter* to be the most dominant

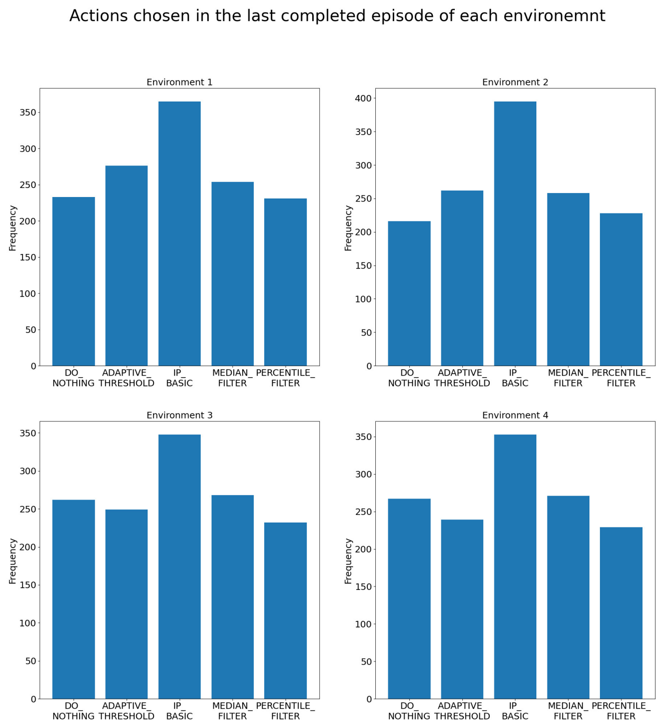


Fig. 6–17: Actions chosen in the last episode of each environment of dataset 21-11-12 14-18-56 showing the *IP_Basic filter* to be chosen most often

Tab. 6–2: Parameters of dataset 21-11-12 14-18-56

parameter	value
camera	Intel Realsense D435
timesteps	250000
number of parallel environments	4
learning rate	0.00005
fps	3.10
iterations	31
time elapsed	81969 s
total timesteps	253952
learning rate	0.00005
loss	0.00298
n_updates	300

relative share of the *Percentile filter* in environment 3 starting at timestep 400 lasting for about 50 timesteps just to decrease to its starting value within another 50 timesteps is interesting since the relative share hits about 4 times the starting value at its peak. The same phenomenon, but on a much smaller scale, can be observed in the other environments. Another interesting point is timestep 900 in environment 1 where the relative share of the *Adaptive Threshold* declines to almost zero, just to increase again so much that the relative share of the *IP_Basic filter* decreases to almost zero around timestep 1000. This phenomenon cannot be observed in the other environments.

Figure 6–19 shows a boxplot of the rewards received categorized by the action that was chosen previously. The action with both the highest mean and median reward was the *IP_Basic filter*. All other actions follow with a small gap and have approximately the same mean and median reward. The interquartile range is approximately the same for all actions. Compared to figure 6–4, the maximum values of the boxplot is smaller for all actions. Especially all actions but the *IP_Basic filter* have a much smaller maximum boxplot value. All actions have many outliers above the maximum boxplot value while only the *IP_Basic filter* has one outlier below its minimum boxplot value.

A histogram of rewards received over the whole training is provided in figure 6–20. The first noticeable difference compared to figure 6–5 is that there is no outlier at the value of 0. Thus, the shape of the histogram looks even more like the one of a normal distribution with a peak value of approximately -0.4. As in figure 6–5, the maximum value is approximately 0.4 and the minimum one -1.0. The slightly slower decline on the right side of the distribution compared to the left side can also be found in both reward histograms.

Figure 6–21 shows the rewards received in the fist and last episode of each environment. As in other figures showing the reward of single episodes, no overall shape can

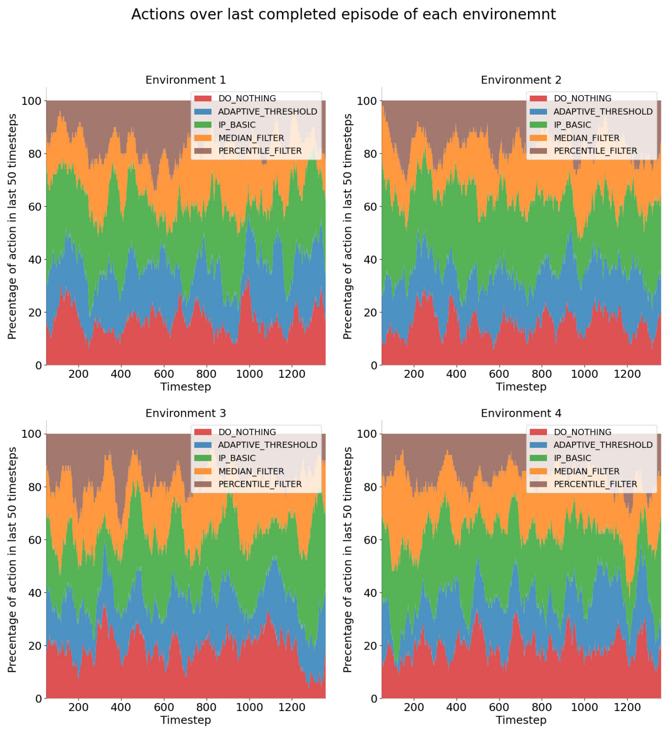


Fig. 6–18: Detailed look on actions chosen over the last episode of each environment of dataset 21-11-12 14-18-56

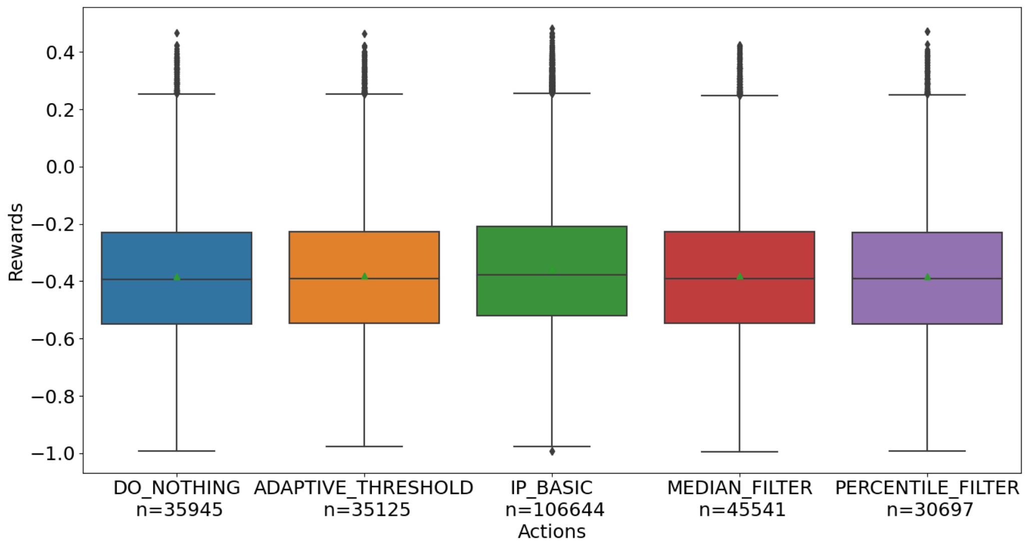


Fig. 6–19: Rewards over actions for dataset 21-11-12 14-18-56 showing the *IP_Basic filter* to have the highest mean and median reward

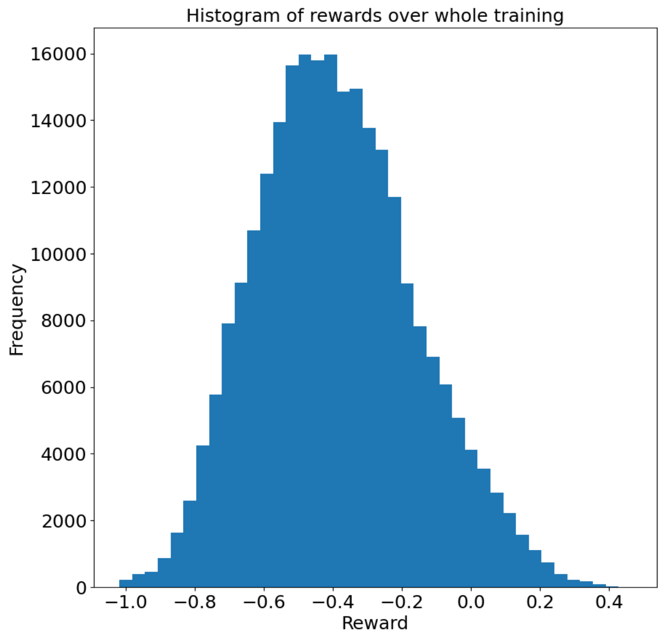
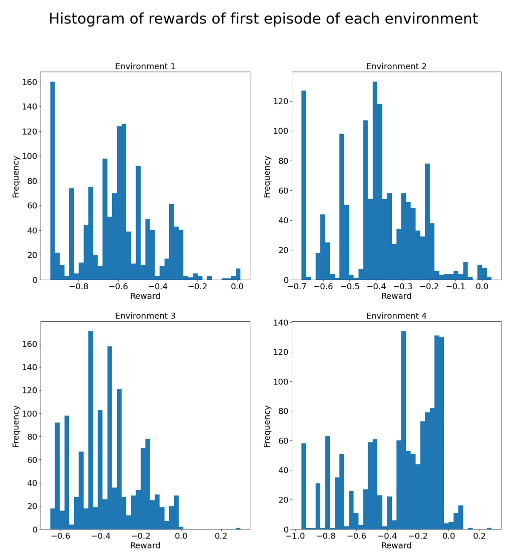


Fig. 6–20: Histogram of rewards received over whole training of dataset 21-11-12 14-18-56 which looks approximately like a normal distribution around -0.4

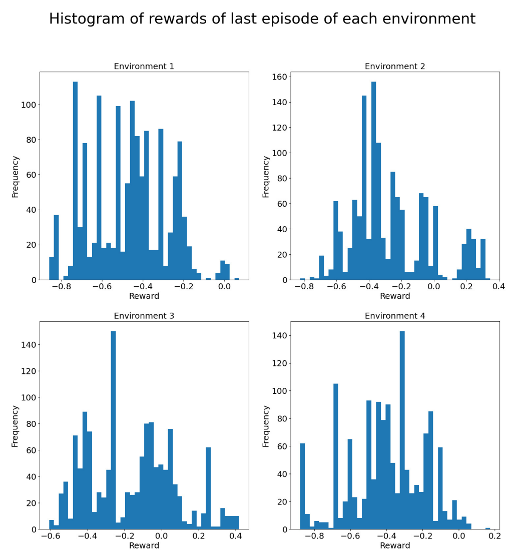
be clearly identified. For environments 1 to 3, the rewards of the last episode are overall higher than in the first episode, i.e., the distribution shifts towards higher rewards. The effect is very strong for environment 3 where almost all rewards are below 0 in the first episode but some rewards reach values of 0.4 in the last episode. For environment 4 however, the effect is the other way around, i.e., the rewards in the last episode are overall lower than the ones in the first episode.

In figure 6–22, a boxplot of the reward components of each environment is shown. It looks pretty similar to figure 6–7, only the S_{rot} component shows major differences. Therefore, only these differences between the two figures will be outlined in the following. Overall, both the mean and median value of the S_{rot} component are lower than in figure 6–7 and now range around the values of the S_{trans} component of approximately 0.5. In environment 3, both are even slightly lower than the ones of the S_{trans} component. The interquartile range of the S_{rot} component in environment 3 lowered. In figure 6–7, three minimum or maximum boxplot values of the S_{rot} component were 0 or 1. In figure 6–22, only the minimum boxplot value of environment 3 is 0 while all other minimum and maximum boxplot values have the value of 0 or 1. However, in all these cases, there are outliers taking the value 0 or 1. While in environment 1, 3, and 4, all minimum and maximum boxplot values of the S_{rot} component are farer away from 0 or 1 than in figure 6–7, both values of environment 2 are nearer to 0 or 1.

The mean reward for each episode of each environment is shown in figure 6–23. As in figure 6–8, the values are fluctuating. In contrast to it, however, the regression straight is rising in all four environments. Also, the minimum slope of the straights is higher than



(a) First episode of each environment



(b) Last episode of each environment

Fig. 6–21: Rewards received in first and last episode of each environment of dataset 21-11-12 14-18-56

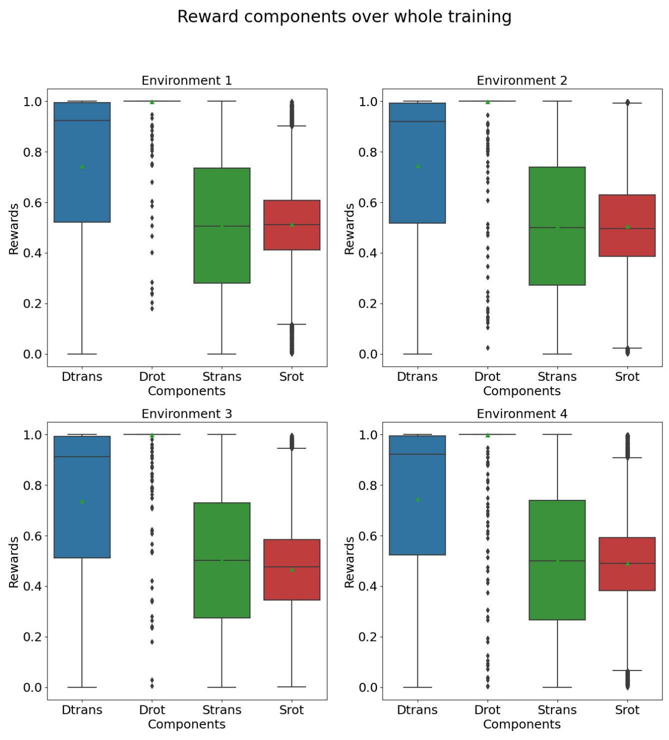


Fig. 6–22: Boxplot of reward components of each environment of dataset 21-11-12 14-18-56

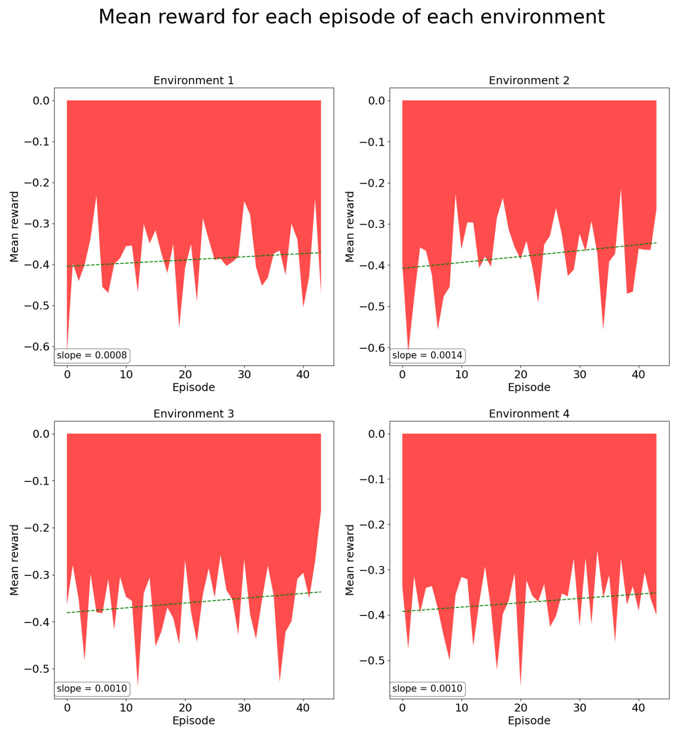


Fig. 6–23: Mean reward for all episodes of each environment of dataset 21-11-12 14-18-56

the maximum slope in figure 6–8 and the maximum slope is twice as high as in figure 6–8. The slope values are still near to 0 but an overall rising trend can be observed for all environments. The highest mean reward of all environments was observed in the last episode of environment 3. Since it was partially discussed in detail in the previous paragraphs, the detailed analysis of this episode will only include figures which were not discussed yet.

6.2.2 Detailed analysis of last episode of environment 3

The components of the rewards in the last episode of each environment are shown in a boxplot in figure 6–24. While the D_{trans} in environment 4 has an extremely high median value of almost 1 and a relatively small interquartile range of only slightly above 0.2, in environment 3, the D_{trans} component shows relatively low mean of 0.75 and median of 0.8. In all environments, the D_{trans} component has the second-highest mean and median value. The D_{rot} component again takes almost only values of 1 in all four environments and hence has both the highest mean and median value of the components in all environments. The S_{trans} component shows a very high interquartile range from 0.15 to 0.8 in environment 3 while having a low one from 0.2 to 0.5 in environment 4. In environments 1 and 4, it has the lowest mean and median values and in environment 2 the lowest mean value of the components. In environment 2, it has the second-lowest median value and in environment 3 both the second-lowest mean and median values of all components. The S_{rot} component shows very different behavior in the four en-

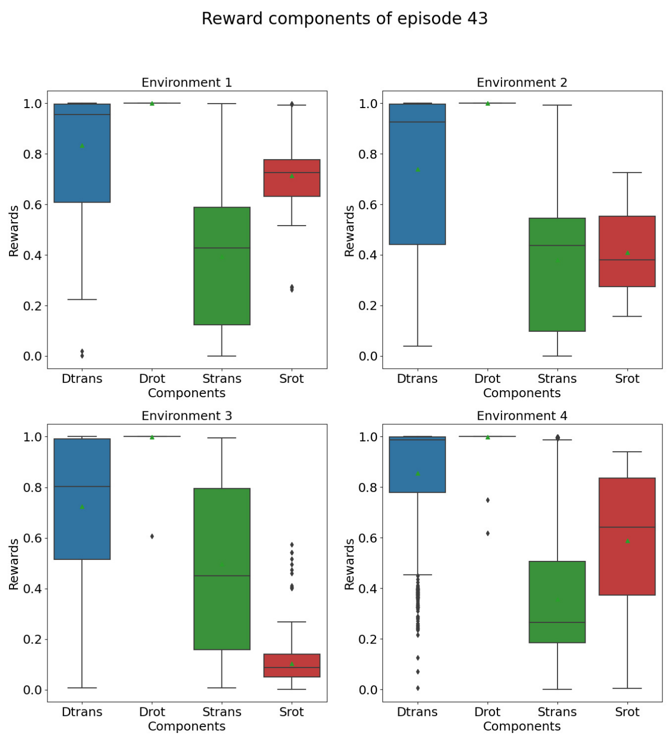


Fig. 6–24: Boxplot or reward components of last episode of each environment of dataset 21-11-12 14-18-56

vironments. In environment 1, it has a relatively high mean and median, both being approximately 0.7. In environment 4, the mean and median values of approximately 0.6 are also relatively high. However, more outstanding in environment 4 is the interquartile range of the S_{rot} component which is the greatest of all components and almost as high as the ones of the other three components combined. In environment 2, the mean and median value the S_{rot} component of approximately 0.4 are a little lower than the 0.5 observed for all environments over the whole training as shown in figure 6–22. The interquartile range is about as large as the ones observed for the whole training in all environments. The S_{rot} component in environment 3 is highly interesting due to its low mean and median values of approximately 0.1 and its low interquartile range of also approximately 0.1. The maximum value of the boxplot of the S_{rot} component in environment 3 is approximately 0.25, a very low value compared to the ones from the whole training where 0.9 was the smallest value. Also, the maximum outlier value was observed below 0.6 while in each environment there were outliers observed at 1 for the S_{rot} component.

Figure 6–25 shows the rewards over the last episode of each environment. Environment 3 shows the most positive rewards whereas environment 1 barely shows any. No similarities can be identified between the environments. Also, no correlations are observed between positive rewards and the actions chosen as visualized in figure 6–18. The observation made for figure 6–14, that the rewards often stay constant for segments of negative rewards but not for positive ones, can be supported. In figure 6–14,

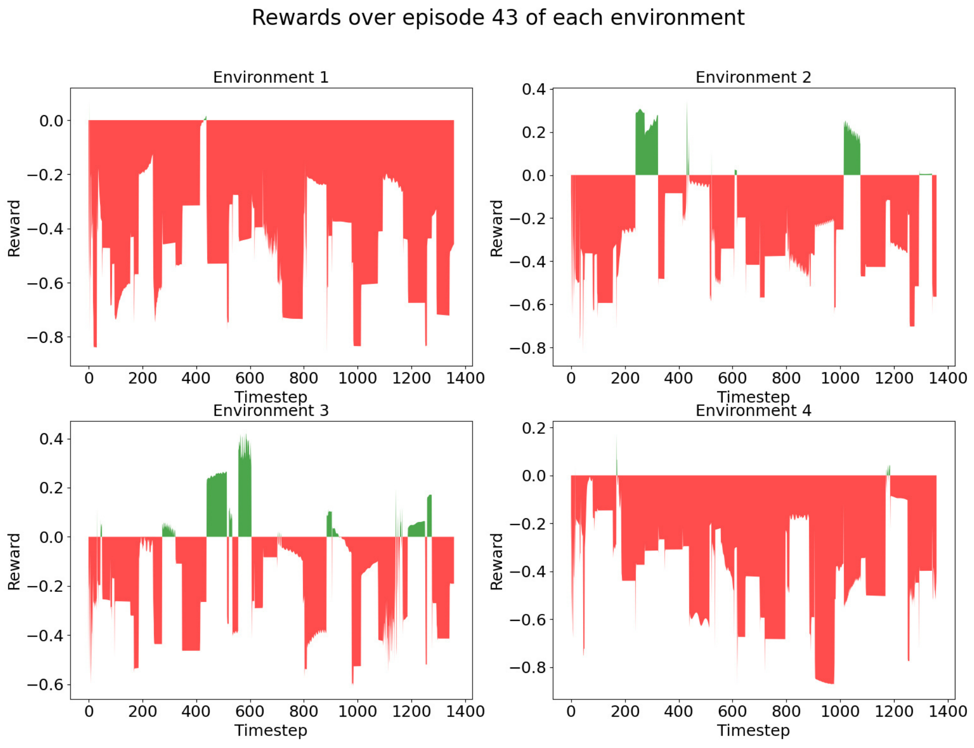


Fig. 6–25: Rewards over last episode of each environment of dataset 21-11-12 14-18-56

the first negative reward after a positive segment often had a higher absolute value than the following negative rewards. This cannot be observed in figure 6–25. Although this phenomenon can also be observed as for example at the negative reward segment starting at timestep 300 of environment 2, the opposite, i.e., the first negative reward after a positive segment being of a smaller absolute value than the following ones, is more common. The negative reward segment starting at timestep 300 of environment 3 is an example for that.

The rewards over the last episode of each environment divided into components are shown in figure 6–25. As in figure 6–11, the positive rewards are caused by D_{trans}, S_{trans}, and S_{rot} all taking relatively low values. Very high rewards as received around timestep 600 in environment 3 are even caused by the S_{trans} and S_{rot} components being almost 0. The observation made for figure 6–15 that negative reward segments with constant combined reward are caused by constant component values can be supported. The very low mean and median values as well as the low interquartile range of the S_{rot} component in environment 3 as observed in figure 6–24 lead to very small subtractions caused by the S_{rot} component in environment 3.

6.3 Corrected reward components for dataset 21-11-12 14-18-56

As described at the beginning of this chapter, in the data shown above an error was made in the calculation of the reward components. Using the trajectory recorded, a corrected version of the rotational reward components is calculated for dataset 21-11-

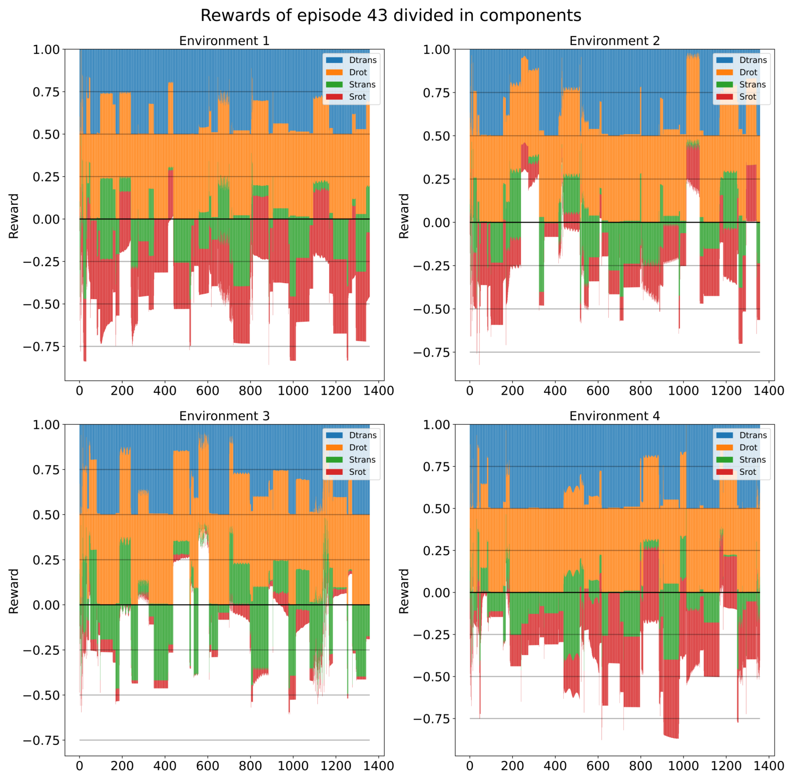


Fig. 6–26: The rewards over the last episode of each environment of dataset 21-11-12 14-18-56 divided in components

12 14-18-56. Although this corrected version could not be used to train the agent, corrected versions of the figures showing the reward components were created. These will be compared to the uncorrected versions in the following.

Figure 6–27 shows the corrected reward components boxplot. Compared to figure 6–22, as expected, the rotational components show different behavior and the translational ones do not. Especially the D_{rot} component is different since it shows values different than 1 in the corrected boxplot. Even though the median is still 1, the interquartile range has increased from 0 to approximately 0.1. Also, the minimum of the boxplot for the D_{rot} component has changed from previously 1 to 0.7 in the corrected version. The mean of the D_{rot} component has lowered from 1 to 0.9. In figure 6–7, below the minimum of the boxplot for the D_{rot} component, there were only few outliers. In contrast to that, in figure 6–27 there are many outliers all the way from the minimum boxplot value to 0. The S_{rot} component does not show as many differences between figure 6–27 and figure 6–7 as the D_{rot} component. Its median and mean stay at approximately 0.5 in all environments. The interquartile range, however, has slightly increased from approximately 0.2 to approximately 0.25. As in figure 6–22, the environments still do not show major differences to each other.

The differences to the uncorrected version are even larger for the boxplot of the reward components of the last episode as shown in figure 6–28. Compared to the uncorrected version in figure 6–24, the interquartile range of the D_{rot} component has increased in

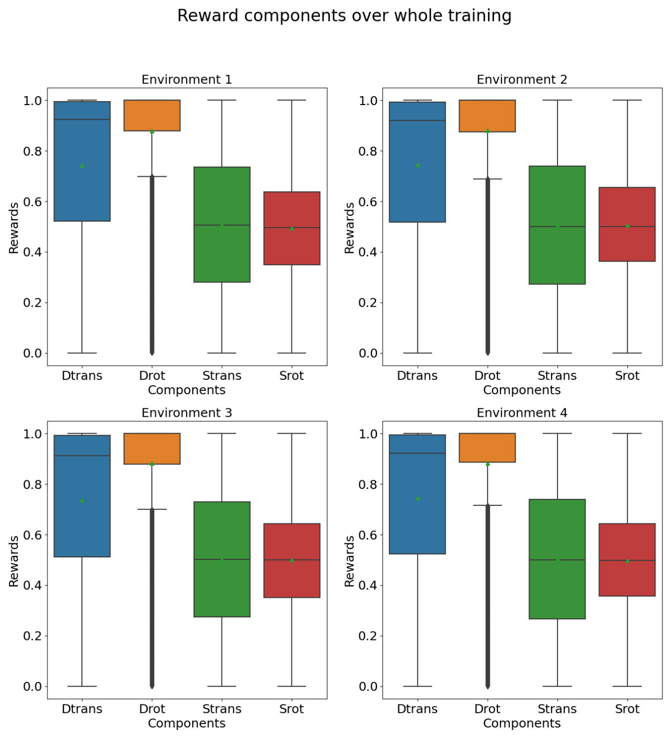


Fig. 6–27: Corrected boxplot of reward components of each environment of dataset 21-11-12 14-18-56

all four environments. Since it was 0 for all environments in the uncorrected version, no environment shows a correlation for the D_{rot} component between the uncorrected and the corrected version. In environments 1 and 4, even the median of the D_{rot} component is lower than 1 while in environments 2 and 3 it still is 1 as in the figure for the uncorrected version and the whole training as shown in figure 6–27. While in environments 2 to 4, the D_{rot} component's mean is still at 0.9 as in the figure for the whole training, in environment 1, it is lower than 0.8. In environments 2 to 4, the minimum boxplot value of the D_{rot} component for the last episode is higher than the one for the overall training. Environment 1, however, shows a very low minimum boxplot value smaller than 0.2 for the D_{rot} component. In all environments, there are outliers below the minimum boxplot value of the D_{rot} component. However, they are not as dense as in the figure for the whole training. The S_{rot} component's boxplot shows a much larger interquartile range compared to the whole training and also compared to the uncorrected version. The S_{rot} component's mean and median were approximately 0.5 in all environments in the figure for the whole training. For the last episode, this can only be observed for environments 3 and 4. While the S_{rot} component's mean is still 0.5 in environment 2, the median is 0.4 as are the mean and median in environment 1. The interquartile range of the S_{rot} component is much larger in all four environments than for the whole training. Environment 2 shows the maximum interquartile range for the S_{rot} component of the four environments with a value of 0.6. Compared to the uncorrected version, in no environment a correlation can be identified for the S_{rot}

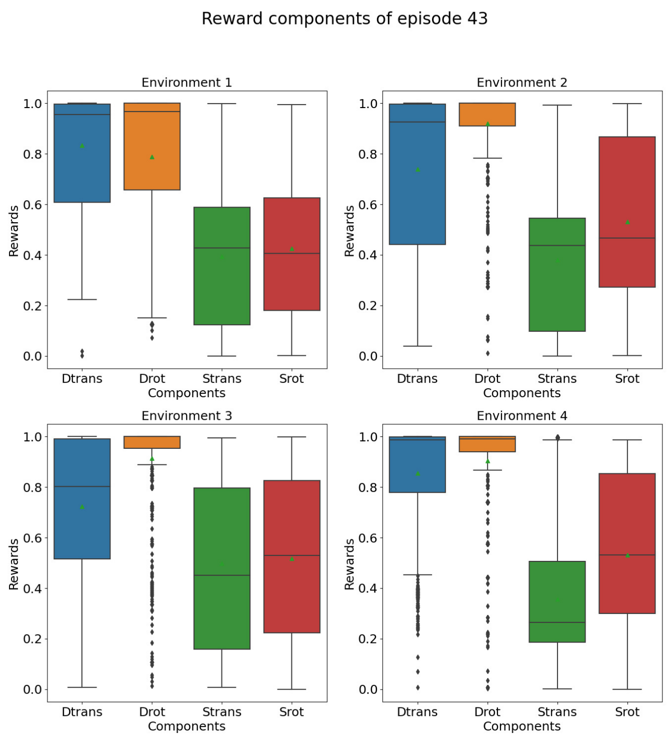


Fig. 6–28: Corrected boxplot or reward components of last episode of each environment of dataset 21-11-12 14-18-56

component, neither for mean or median nor for the interquartile range.

Figure 6–29 shows the corrected version of the rewards over the last episode. In the uncorrected version as shown in figure 6–25, an overall fluctuating behavior without any correlations between the environments was identified. This can be confirmed by the corrected version. Some positive reward segments of the uncorrected version can also be identified in the corrected version. An example is the positive reward segment around timestep 300 in environment 2. Others from the uncorrected version, for example the one at timestep 1000 in environment 2, cannot be observed in the corrected version. Meanwhile, new positive reward segments can be identified in the corrected version, for example at time step 300 in environment 1. The uncorrected and the corrected version therefore show some correlations but also some differences.

6.4 Reward function for good trajectory

In the following, an almost perfect trajectory from previous studies of *Franceschini* [6] will be evaluated with the reward function. For that purpose, the recorded trajectory in file *08d56b02-9451-4bbd-9cf7-f030228ecd6f.nc* was used. This trajectory was computed with a tested version of the DIFODO algorithm. The trajectory projected in a best-fit plane and the best-fit circle for the same are shown in figure 6–30a. The trajectory seems to align perfectly with the best-fit-circle, of which it covers about two thirds. The points of the trajectory are dense. In figure 6–30b, the start of the trajectory shown

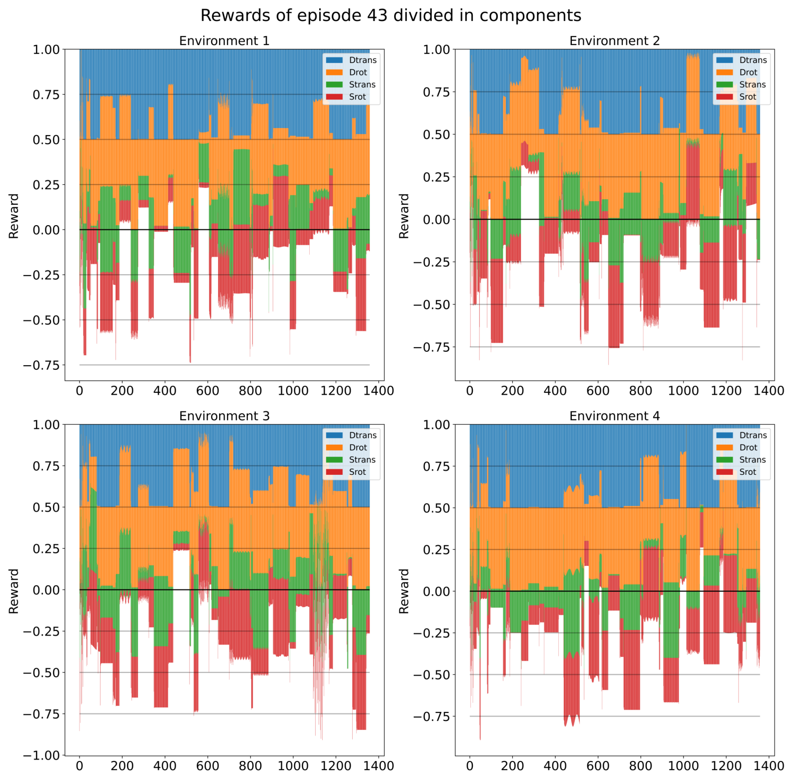
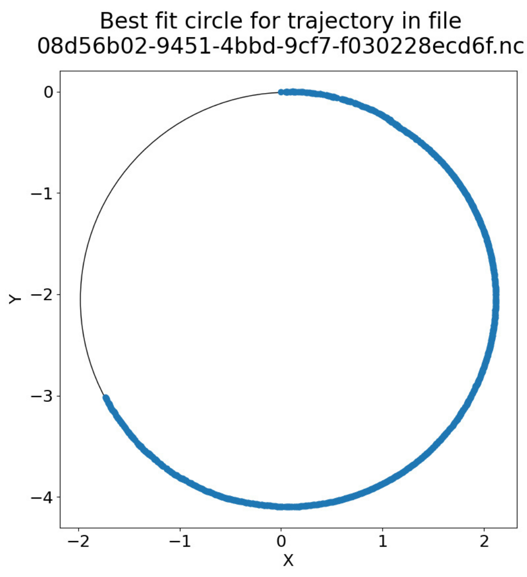
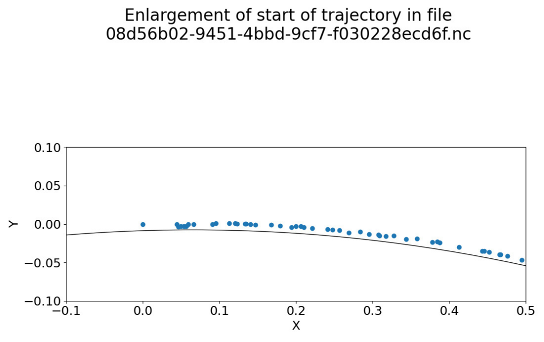


Fig. 6–29: The corrected rewards over the last episode of each environment of dataset 21-11-12 14-18-56 divided in components

in figure 6–30a is enlarged. In this version, the points of the trajectory do not lay perfectly on the best-fit-circle but are still all very near to the same. The distance between the points is unregular, i.e., some points lie in groups with a small distance between them while others are separated from other points.

Figure 6–31a shows the yaw, pitch, and roll angles of the trajectory shown in figure 6–30a.For a perfect reward in the rotational components, the yaw and roll angles would need to be constant while the pitch angle would need to rise linearly following the red line displayed in the figure. At the beginning of the episode, the roll angle's line jumps between -180 and $+180$ multiple times. One single jump from -180 to $+180$ can be observed for the yaw angle around timestep 700. At the beginning of the episode, both the yaw and the roll angle are rising respectively falling with a small slope, besides the mentioned jumps for the roll angle. The yaw angle has a value of approximately 0 and the roll angle of approximately -180 during these timesteps. In the second quarter of the episode, however, both angles switch their values. During this switch, both lines look almost like one half of a sinus-curve. After the switch, both values stay approximately constant for the rest of the episode, besides the mentioned jump in the yaw angle's curve. The pitch angle follows the red line during the first third of the episode, i.e., it shows the expected trend during the first third of the episode. From the second third of the episode on, however, it starts decreasing with about the slope it was increasing with before. This change is observed at the same timesteps, the yaw and roll angle switch their values. From that point on, the pitch angle is decreasing linearly





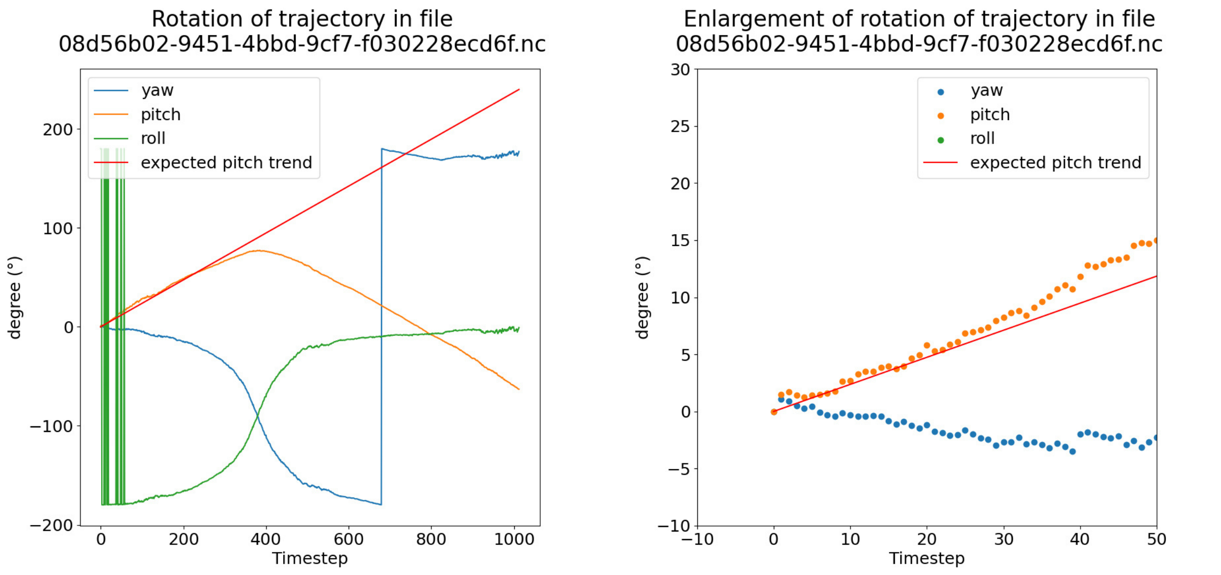
(a) Good trajectory and best-fit circle for the same

(b) Enlargement of start of good trajectory on the left

Fig. 6–30: Good trajectory, best-fit-circle, and enlargement of the start of the trajectory

until the end of the episode. Figure 6–31b shows an enlargement of figure 6–31a. In contrast to figure 6–30b, no groups can be observed. Instead, the points fluctuate a little, i.e., some points behave contrary to the overall trend. Hence, the distance on the degree axis is also fluctuating. The points do not have a constant distance here. Because of the fluctuations, the direction to get from one point to the next one, i.e., the one of the next timestep, is alternating.

After the description of the trajectory itself, it will now be evaluated with the reward function. For that purpose, figure 6–32 shows the rewards over the episode divided in components in figure 6–32a and a boxplot of the reward components in figure 6–32b. The total reward is fluctuating a lot, but approximately -0.5 for most timesteps of the episode. However, there are also few positive rewards, some of which are even higher than 0.5. Even though the D_{trans} and D_{rot} components are 1 for almost all timesteps, their mean is about 0.9 for the D_{trans} and 0.95 for the D_{rot} component. Both components show outliers over the whole range of the reward spectrum, i.e., from 0 to 1. The S_{trans} and S_{rot} components in contrast show values different than 1 much more frequently. The interquartile range of the S_{trans} component reaches from 0.2 to 0.6. The mean and median of this component are at approximately 0.4. The S_{rot} component shows higher values. Its interquartile range is between 0.4 and 0.8 while the mean and median are at approximately 0.6. Both the boxplot for the S_{trans} and the one for the S_{rot} component have their minimum at 0 and their maximum at 1. The D_{trans} and D_{rot} components show no change over the episode. In contrast to that, the S_{trans} and S_{rot} component show a trend looking somewhat like a sinus-curve. The S_{trans} component starts with a subtraction of 0.5, i.e., a reward component value of 1, at the beginning of the episode of which it decreases to almost 0 around timestep 800, from where on it increases again. The S_{rot} component shows the inverse behavior. It starts with almost 0 at the



(a) Rotation of good trajectory with expected behavior for perfect reward **(b) Enlargement of rotation of good trajectory**

Fig. 6–31: Rotation of good trajectory

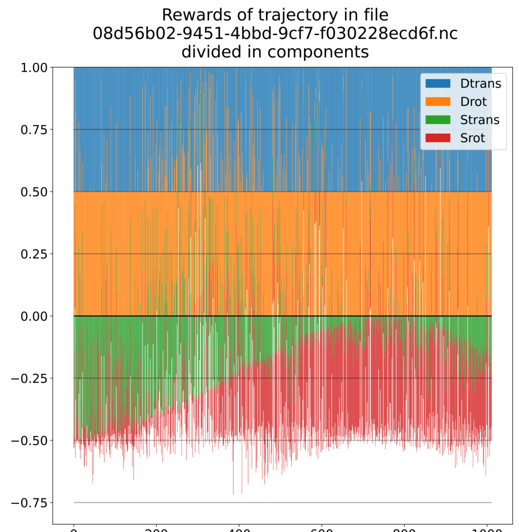
beginning of the episode, increases to a subtraction approximately of 0.5, i.e., a reward component value of 1, around timestep 800, from where on it decreases again.

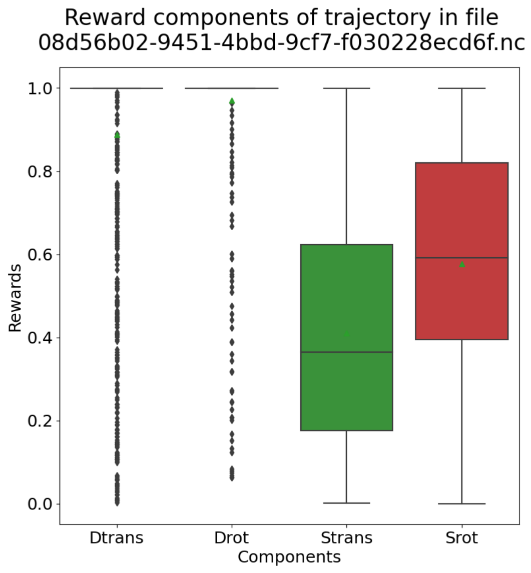
6.5 Iterating the number of parallel environments

Figure 6–33 shows the performance for different number of parallel environments measured in timesteps per second. It consists of two training runs and the mean of those. The maximum number of parallel environments is 6 since the largest memory available at the *Condor* GPU cluster was exceeded by 7 or more parallel environments. While the two runs show large differences in places and there are even declines in the curves such as between environment 3 and 4 of the second run, the mean value increases roughly linearly. The mean performance for 6 environments is about 4.8 timesteps per second. The mean performance observed in figure 6–33 also matches the performance observed for 4 parallel environments by the datasets analyzed in more detail where performances of 3.47 and 3.10 timesteps per second were observed.

6.6 Iterating the learning rate

In figure 6–34, the mean reward of each episode of each environment for different values for the leaning rate are shown. The values range from 0.1 to $1\mathrm{e}{-06}$ and are divided by 10 between each step. All runs were conducted for 125000 timesteps and used images taken by the *Intel Realsense D435*. For the learning rate 0.001, NaN rewards were received and overwritten with 0 in episodes $7, 17$, and 18. The reward components in the corresponding timesteps had the following values: $D_{trans} = 1.0, D_{rot} = 1.0, S_{trans} = NaN, S_{rot} = NaN$. In the training with learning rate $1\mathrm{e}{-05}$, NaN rewards were received in episode 6. The components had the following values:





(a) Rewards of the good trajectory divided in components

(b) Boxplot of reward components of good trajectory

Fig. 6–32: Reward components of good trajectory

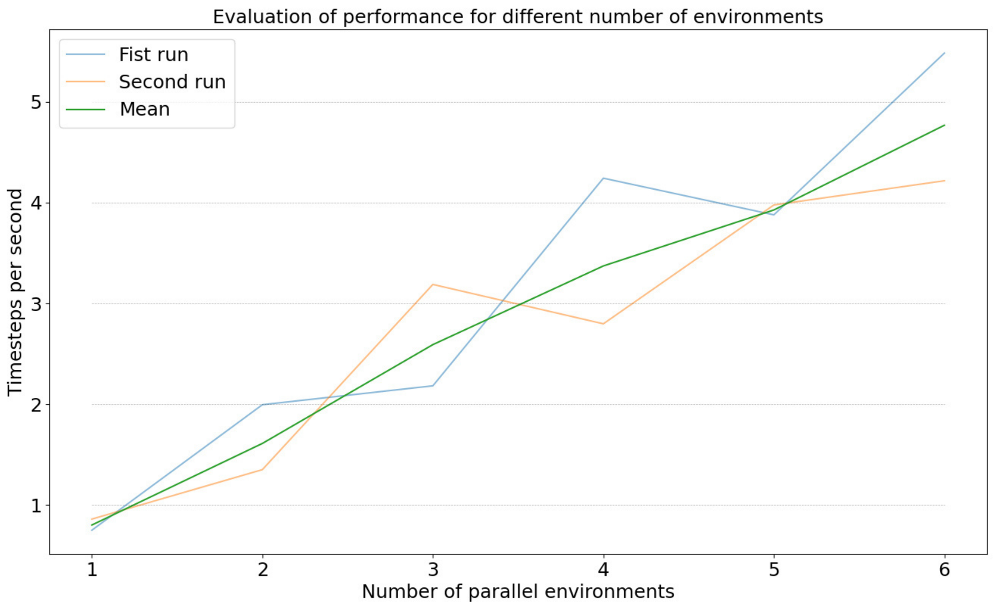


Fig. 6–33: Performance for different number of parallel environment showing linear performance increase with number of episodes

$D_{trans} = 1.0, D_{rot} = 1, S_{trans} = NaN, S_{rot} = 0.32$. The value for S_{rot} was changing over the timesteps in which NaN rewards were received but all other component had constant values. For the learning rate $1\mathrm{e}{-06}$, NaN rewards were received and overwritten in episodes $5, 7$, and 13. The rewards component values for all observed timesteps with NaN rewards were: $D_{trans} = 1.0, D_{rot} = 1.0, S_{trans} = NaN, S_{rot} = NaN$. Due to the low number of episodes the training ran for, the influence of 0 mean rewards on the regression line is relatively high. Thus, in case mean rewards of approximately 0 were observed, two regression lines – one including the respective rewards and one which does not – were plotted.

The mean reward for a learning rate of 0.1 as shown in figure 6–34a clearly shows a declining trend in all environments. The slope of the regression lines varies between -0.0038 in environment 2 and -0.0055 in environment 1.

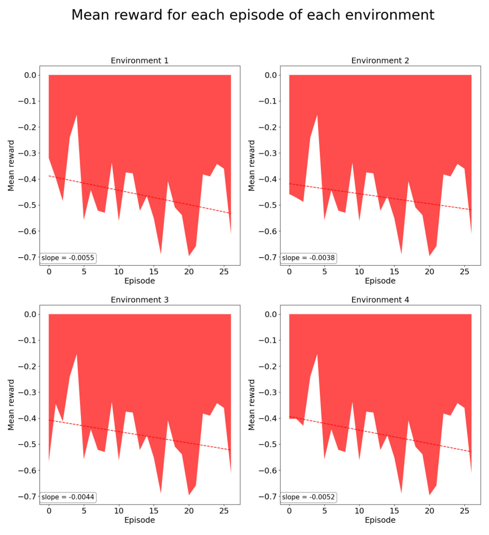
For a learning rate of 0.01 as shown in figure 6–34b, however, an increasing trend can be identified in all environments. In environments 2 and 3, the slope of the regression line is relatively low with values of 0.0005 in environment 2 and 0.0015 in environment 3. In environments 1 and 4 in contrast, the slope is relatively high with values of 0.0036 in environment 1 and 0.0067 in environment 4. This is the highest slope value observed in any of the environments shown in figure 6–34 together with the one of environment 4 of figure 6–34f showing the mean reward for a learning rate of $1\mathrm{e}{-06}$. The latter one, however, includes the 0 mean rewards observed during the training while the slope value not taking the 0 mean rewards into account is 0.0063.

Figure 6–34c shows the mean rewards for a learning rate of 0.001. When considering the 0 mean rewards, the trend is increasing in all four environments with a slope value between 0.0028 in environment 2 and 0.0057 in environment 3. If the 0 mean rewards are not considered, however, the trend is not that clear. In environments 1 and 4, the trend is almost constant with slope values of 0.0003 in environment 1 and 0.0001 in environment 4. While the regression line in environment 3 is ascending with a slope value of 0.0021, it is declining in environment 2 with a slope value of -0.0013.

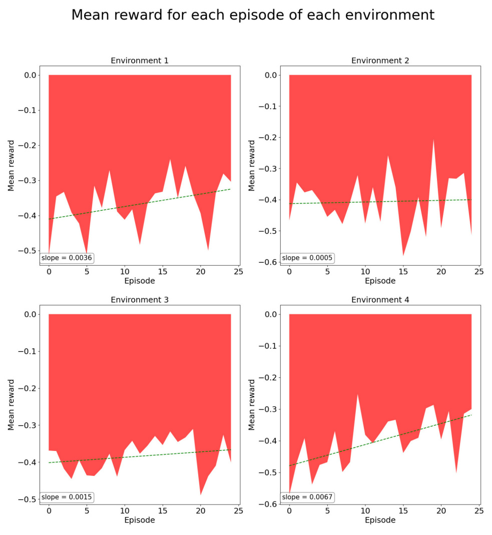
The trend for a learning rate of 0.0001 as shown in figure 6–34d cannot be clearly identified since the regression line is falling in environment 1 while rising in environment 2, 3, and 4. The absolute value of the slope in environment 1 however is lower than the ones in the other environments with a value of only -0.0014 while the slope values in the other environments range between 0.0030 and 0.0063. Episode 6 does not show outstanding high mean rewards although NaN rewards were received in this episode. Instead, episode 6 in environment 2 does even show a much lower mean reward than episodes 5 and 7. In the other environments, episode 6 does not show outstanding mean reward values.

In figure 6–34e, the mean rewards for a learning rate of $1\mathrm{e}{-05}$ are shown. An overall trend cannot be identified since the regression line is rising in environments 1, 3, and 4 and falling in environment 2. The highest slope is observed in environment 1 with 0.0034 and the second highest in environment 4 with 0.0012. In environments 2 and 3, the absolute slope value is below 0.001.

The mean reward for a learning rate of $1e{-}06$ is shown in figure 6–34f. For both the regression line with and without consideration of the 0 mean rewards, the trend cannot be clearly identified. In environment 1, both lines are approximately constant with slope values of 0.0002 for the regression line considering the 0 mean rewards and -0.0004 for the one which does not. Although both regression lines are slowly rising in environment 2, the slope values of 0.0012 and 0.0006 are relatively low. In environment 3, both regression lines are declining, the one considering the 0 mean rewards with a slope of -0.0023 and the one which does not with a slope of -0.0029. The absolute slope values in environment 4 are the highest one observed in figure 6–34f and the highest ones observed in all environments of figure 6–34 in their respective categories. The regression line which does not consider the 0 mean rewards has a slope value of 0.0063 while the one which takes the 0 mean rewards into account shows a slope value of 0.0067.

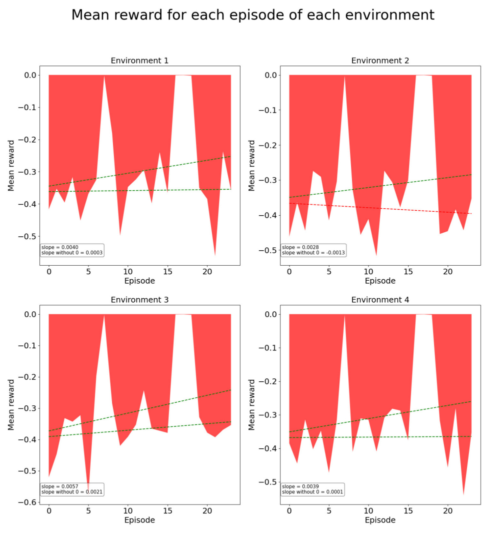


(a) Learning rate 0.1

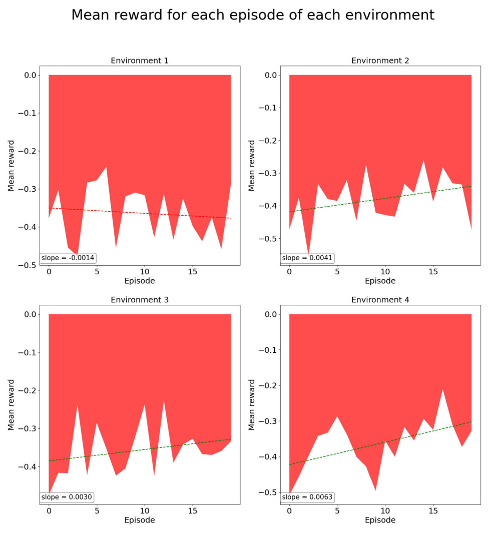


(b) Learning rate 0.01

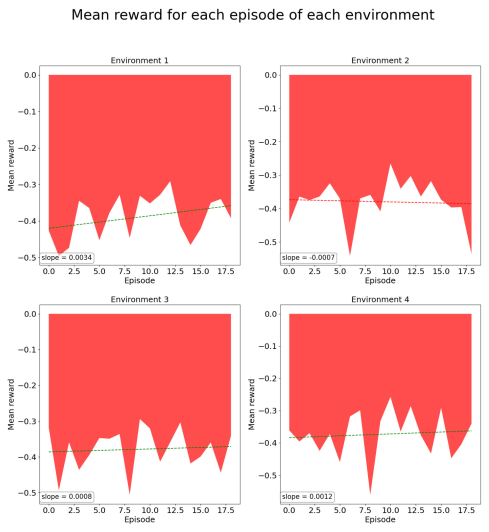
Fig. 6–34: Mean reward of each episode of each environment for different learning rates



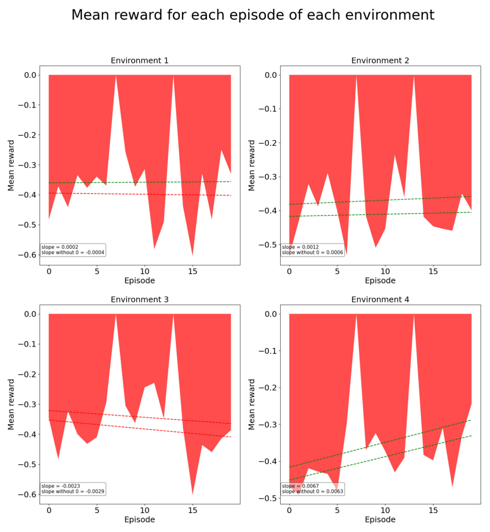
(c) Learning rate 0.001



(d) Learning rate 0.0001



(e) Learning rate $1\mathrm{e}{-05}$



(f) Learning rate $1\mathrm{e}{-06}$

Fig. 6–34: Mean reward of each episode of each environment for different learning rates

7 Discussion

The following will discuss whether the developed toolchain with RL is suitable to evaluate the usage of RL in the RACOON-Lab's toolchain once the DIFODO algorithm or an alternative is successfully tested. It will discuss adoptions to make, especially for the parameters tested in detail. Special focus will be laid on the reward function and the evaluation functions.

7.1 Expressiveness of figure types

The first part of the discussion focuses on the expressiveness of the figure types. The figure types' expressiveness is ranked in three categories: as an individual figure, compared to figures of the same type, and compared to or combined with figures of other types. The criteria for the categories are shown in figure 3–1.

7.1.1 Action histograms

There are two types of action histograms: The first one – the *action histogram of whole training* figure type – takes all actions of the whole training into account. The second one – the *action histogram of individual episode* – takes only actions of one episode into account and also distinguishes between the environments.

The first one gives an overview of the actions chosen over the whole training. This can be used to determine the most- and least-chosen actions. In case the figure looks somewhat like figure 6–1, where one action is clearly the most dominant, after a long training, this might be an indication to use this action as a general option in the RACOON-Lab's toolchain even without the usage of RL. It however needs more indications to actually consider this option, for example that the action has a high reward and is applied constantly over the whole training. The expressiveness of the individual figure will be ranked with $-$ since it only gives an overview, which is considered a less relevant information, and any further information needs to be verified with other figures. Since this is a figure that can be created only once per training, it is a *single-time* figure type. When comparing two figures of this type, one might observe clear overall differences between the trainings but may lack to accurately show small differences. These overall differences are considered a less relevant information. The first were demonstrated in the description of figure 6–16. The expressiveness of the figure when being compared to figures of the same type is hence ranked with $-$. For the purpose of comparing the figure with episodes evaluated in more detail, the diagram is highly useful, since deviations of individual episodes from the overall behavior can be easily identified. These episodes could be for example especially well- or poorly-performing episodes identified using the *mean reward* figure type. This was done in the description of figure 6–2. This can be done multiple times per training since it is possible for each episode. Differences between especially well- or poorly-performing episode and the overall behavior are considered a relevant information. Also, figures of this type can be used in combination with figures of the *reward histogram of whole training* type to get a hint about the differences between trainings, which is considered a less relevant single-time information. The expressiveness when being compared to or combined with figures of other types is thus ranked with o.

The second type of action histograms displays the actions chosen in all environments of an individual episode. As an individual figure, this histogram can be used to determine differences between the environments. This can be very useful in case one environment performed significantly better or worse than the others. The figure can therefore be used to check whether the performance differences are the result of a difference in the actions chosen over the episode, which is considered a relevant information that is also clearly observable. The expressiveness of the individual figure is hence rated with o. This figure type cannot only be compared to the same figure type from other trainings but also to ones from the same training for a different episode, thus it is a *multi-time* figure type. This, for example, could be a comparison of two well-performing episodes in which similarities would be searched. The same can be done for poorly-performing episodes. Both are considered relevant information. The expressiveness when being compared to figures of the same type is therefore ranked with $+$. As already mentioned above, the action histogram of an individual episode can be used to identify deviations from the overall behavior when being compared to the *action histogram of the whole training*. This is considered to be a relevant information. Figures of the *action histogram of individual episode* type can be used in combination with figures of the *Reward histogram of episode* and *Reward components of episode* types. This might give an overview of which actions are responsible for high or low rewards or reward components. Both are considered a less relevant information as they are only overviews. The expressiveness when being compared to or combined with figures of other types is hence ranked with $+$.

7.1.2 Actions over episode

The *actions over episode* figure type shows the actions of each environment of an individual episode over the timesteps of the episode. As an individual figure, it can be used to determine whether differences in the rewards for the environments of an episode are a result from the timestep where the actions were applied. That is considered to be a relevant information that is clearly observable. Partially, it can also be used for similar purposes as the *action histogram of individual episode* figure type, i.e., to determine differences in the number of applications of the actions in the environments. In comparison to this figure type, however, the *actions over episode* figure type will only be useful for a broad overview, which also is not that easy to read out of the figure. This is considered to be a hint to less relevant information. The two figure types hence should be used in parallel, as both can be used to understand differences in the rewards for the environments of two figure types, regardless of which episode or training they are from. The expressiveness of the *actions over episode* figure type is rated with $+$. Before evaluating the expressiveness of the figure type in the two remaining categories, it should be noted that any comparison between two figures of this type is meaningful only when the lookback is the same for both. Like the *action histogram of individual episode* figure type, this figure type can be compared to figures of the same type from both other episodes of the same training and episodes from another training. It thus is a *multi-time* figure type. Similar to the *action histogram of individual episode*, interesting episodes, for example very well- or poorly-performing ones, can be compared to each other to search for explanations of the good or bad performance. However, this is quite difficult relative to comparing two figures of the *action histogram of individual episode* figure type, since the timestep at which a certain behavior occurs could depend on certain conditions for the input image, such as the presence of reflections in these images. Both explanation of well- and poorly-performing episodes are nevertheless considered to be a relevant information. As for the individual episode, a broad comparison of the actions chosen in the episodes and environments can also be made, which is considered to be a less relevant information since it is only an overview. The expressiveness of this figure type compared to figures of the same type is rated with $+$. This figure type can be very well compared to other figure types showing information over episodes, i.e., the *rewards over episode* and *rewards over episode divided in components* figure types. Using this combination, the timesteps with high or low rewards could be explained with the actions chosen near to that timestep. However, if the lookback of the *actions over episode* figure type is too high, this might be difficult. Since any lookback over 1 makes a perfect understanding of what caused the reward impossible but the figure type gets almost useless if the lookback is set to 1, a compromise needs to be made. The *Rewards over actions* figure type already addresses the correlation of rewards and actions. Therefore, the decision is made for a higher lookback. Although not a perfect one, an understanding of which actions caused a reward is still possible, especially for segments of rewards, with the combination of the *actions over episode* and *rewards over episode* or *rewards over episode divided in components* figure types. The correlation between both actions chosen and rewards received and action chosen and reward components received are considered to be relevant information. The expressiveness of the *actions over episode* figure type is rated with $+$.

7.1.3 Rewards over actions

Figures of the *Rewards over actions* type show a boxplot of the rewards received for each action over the whole training. As an individual figure, this can be useful to determine which action led to high rewards and which to low ones. This is considered to be a relevant information. Additionally, the spread of the rewards and hence the constancy of the same can be extracted from figures of this type, which is considered a less relevant information. The number of applications of each action is also given in this figure type, which was considered to be a relevant information above. The individual expressiveness of this figure type is ranked with $++$. This figure type can only be created once per training and is thus a *single-time* figure type. When being compared to figures of the same type of another training, similarities and differences in the rewards for the episode as well as the spreads for the same can be identified. The first is considered relevant information since it can be used to determine which actions are useful and which are not. With that knowledge, one might be able to optimize the actions, for example by tuning the parameters of the *Percentile filter*. The comparison of the spreads of the boxplots are considered to be less relevant information. The expressiveness of this figure type when being compared to figures of the same type is therefore rated with $-$. This figure can barely be combined with figures of other types. The expressiveness of this figure type in the respective category is hence rated with o.

7.1.4 Reward histograms

There are two types of reward histograms. Similar to the action histograms, the first – the *reward histogram of whole training* type – takes all rewards of the whole training into account while the second one – the *reward histogram of individual episode* type – takes only the ones of one episode into account and also distinguishes between the environments.

The first type as an individual figure gives an overview of the rewards received during the training. The shape of the distribution gives an overview of the overall reward structure. This information is clearly observable but considered less relevant. The expressiveness of this figure type individually is hence ranked with $-$. Figures of this type can only be created once per training which is why the figure type is a *single-time* one. When comparing a figure of this type to one of the same, one should make sure to compare only figures for trainings with a similar number of timesteps. Otherwise, in case the rewards improved over time, one would expect the two figures to look differently and a the results of a comparison would therefore be compromised. The comparison of two figures of the *reward histogram of whole training* type can point out differences in the reward structure of the two trainings. This is considered a less relevant information. Comparing multiple figures of this type can also show relevant information regarding the reward function. If the peak for all figures is the same, one might want to find out why that is. The expressiveness of the *rewards histogram of whole training* figure type is rated with $-$. A comparison of figures of this type with figures of the second reward histogram type can point out interesting information. For example, one can identify how well- or poorly-performing episodes, as found with the *mean reward* figure type, differ from the overall reward structure. this again might be useful to evaluate the reward function or to explain the well- or poor performance. It is thus considered a relevant information and, as the second reward histogram is a *multi-time* figure type, can be done multiple times per training. The expressiveness of this figure in the last category is rated with o.

The second type of reward histograms displays the rewards received in all environments of an individual episode. When used individually, it can show differences between the environments for that episode, which is especially useful for episodes in which one environment performed much better or worse than the others. An analysis of the reward structure for the episode might help to explain these performance differences. It is therefore considered a relevant information, which is also clearly observable. The figure type is hence rated with o in the first category. Since this figure type is available for each episode of a training, it is a *multi-time* figure type. Compared to figures of the same type, similarities in the reward structure for both well- and poorly-performing episodes can be identified. Both are considered relevant information, both for evaluating the reward function and the training. The expressiveness of the *reward histogram of individual episode* figure type compared to figures of the same type is rated with $+$. As mentioned in the previous paragraph, figures of this type can be used to show differences in the reward structure of an individual episode to the overall training. That is considered a relevant information. As mentioned for the *action histogram of individual episode* figure type, both figure types for the same episodes can combined to get an overview which actions are responsible for high or low rewards. This is considered to be a less relevant information. The expressiveness of the *reward histogram of individual episode* figure type is thus rated with o.

7.1.5 Reward components boxplots

There are two types of reward components boxplots which both distinguish between the environments: the first – the *reward components boxplot of whole training* figure type – shows a boxplot for each reward component over the whole training. The second – the *reward components boxplot of individual episode* figure type – does the same for an individual episode.

Figures of the first type can be used to get a deeper understanding of the rewards. Not only the mean and median value but also the interquartile range as well as the maximum and minimum values can be identified for each reward component. All of these are considered to be highly relevant since they can be used to evaluate the reward function. Via the boxplot they are also clearly observable and comparable. In the two figures of this type that were analyzed in the results chapter, no clear differences between the environments could be identified. For trainings with more timesteps, the differences would probably get even smaller. Hence, the combination of the data of all environments in one boxplot is proposed for future applications. Nevertheless, the expressiveness of the *reward components boxplot of whole training* figure type is rated with $++$. This figure type can only be created once per training and is therefore a *single-time* figure type. A comparison with a figure of the same type from another episode is useful to evaluate the reward function's components in detail since changes to the reward function should never be made based on only one training. This thus is considered to be relevant information which is why the figure type is rated with $-$ in the second category. Figures of this type can be compared to figures of the *reward components boxplot of individual episode* very well. This is especially useful for well- or -performing episodes which can be identified using the *mean reward* figure type. These comparisons can be used for the evaluation of the reward function and are hence both considered to be relevant information. Since the *reward components boxplot of individual episode* figure type can be created for each episode of a training, this comparison is a *multi-time* one. The expressiveness of the *reward components boxplot of whole training* figure type is rated with $+$.

The second type of reward components boxplots individually is useful for comparing well- or poorly-performing environments to others of the same episode. This will be useful to find out how high and low rewards are composed, which again can be used to evaluate the reward function. Both are therefore considered relevant information, which are clearly observable. The figure type is thus rated with $++$ in the first category. As already mentioned in the paragraph above, this figure type is a *multi-time* one since it can be created for each episode of a training. Comparing two figures of the *reward* *components boxplot of individual episode* type is especially interesting for well- and poorly-performing episodes because the differences will point out which components are most relevant for high or low rewards. Both are considered to be relevant information. Hence, the figure type is rated $+$ in the expressiveness when being compared to figures of the same type category. The same episodes can also be compared to the boxplot of the overall training which will also assist in finding the most influential reward components. This relevant information leads to a rating of $+$ in the third category.

7.1.6 Mean reward

The *mean reward* figure type shows the mean reward of each episode of each environment and a regression line and its slope. As an individual figure it can be used for three main reasons: First, the regression line shows the training trend, i.e., if the RL agent improves over the training. This information is necessary to determine whether the usage of RL in the RACOON-Lab's toolchain makes sense. Second and third, the best and worst performing episodes can be identified using this figure. These information are crucial to further analyze these episodes as described above. The expressiveness of the *mean reward* figure type individually is rated with $++$. The figure is created once per training what makes the figure type *single-time*. Comparisons with other figures of this type are useful to compare the performance of two trainings, which for example could be interesting to see which agent to train further. Also, such comparisons can be used to optimize parameters such as the learning rate as done in figure 6–34. Both information are considered to be relevant. The expressiveness of the figure type when being compared with figures of the same type is rated with o. As described above the *mean reward* figure type is used to identify interesting episodes which can then be further analyzed with other figure types. Although that is a combination of two figure types, it was not treated as category 3 to avoid that some figure types such as *reward histogram of individual episode* can only be rated in that category. Thus, even though the *mean reward* figure type is commonly used in combination with other figure types, this is not represented in the last category, which is why the figure type is rated with $--$ in the same.

7.1.7 Reward over episode

The reward over a single episode of all environments are shown in figures of the *reward over episode* type. Individually, this figure can mainly be used to compare the environments. If all environments have a low reward at some point of the episode, this could be caused by reflections in or bad quality of the input images during these timesteps. This information is clearly observable and considered relevant which is why the expressiveness as an individual figure is rated with o. Since figures of this type can be created for each episode of a training, it is a *multi-time* type. Comparing multiple figures of this type, especially those for well- and poorly-performing episodes as observed with figures of the *mean reward* type, could show similarities. If all well-performing episodes for example would still show badly performing segments, they are probably caused by reflections in the input images at the respective timesteps. This could be the reason to offer another action to the RL agent since the existing ones were not able to handle the reflections. The information is therefore considered relevant which leads to a rating of o in the second category. Together with the figures of the *action over episode* type from the same episode, a figure of the *reward over episode* type can be used to search correlations between rewards and actions chosen to receive them. As above, this information is considered relevant which is why the expressiveness of the *reward over episode* figure type when being compared to or combined with figures of other types is rated with o.

7.1.8 Reward over episode divided in components

Like the *reward over episode* type, the *reward over episode divided in components* type shows the rewards of each timestep of an episode of each environment. Additionally, the rewards are divided in components. Since the total reward of each timestep is still clearly observable in this figure type, all the information that can be identified using the *reward over episode* type can also be identified with this figure type. Thus, only additional information will be listed in this rating. However, the ratings for this figure type will include the information that can be identified with both figure types and will therefore be at least as high as the ones for the *reward over episode* type. In the first category, figures of the *reward over episode divided in components* type additionally shows the reward components. This information can be used to identify the differences between the reward components of high and low rewards. This information is considered relevant since it can be used both to evaluate the toolchain and the reward function. The expressiveness of the *reward over episode divided in components* individually is hence rated with $++$. This figure type can be created for all episodes of a training. It thus is a *multi-time* figure type. Comparing multiple figures of this type can show whether the components of high or low rewards are the same for different episodes. Using this information, the most influential reward component can be identified for example. The first is considered to be relevant and the second less relevant. The figure type is therefore rated with $+$ in the second category. In the third category, additional to the information observable with the *reward over episode* type, correlations between reward components and the actions chosen can be identified when comparing the figures to the respective *actions over episode* figure. This information is also considered relevant since it can be used to evaluate both the toolchain and the reward function. The expressiveness of the *reward over episode divided in components* figure type when being compared to or combined with figures of other types is hence rated with $+$. Since all the information of the *reward over episode* figure type is also clearly observable with this figure type, the further usage of the *reward over episode* figure type is not reasonable.

7.1.9 Summary

Table 7–1 summarizes the ratings of all figure types in the three categories. Even though table 3–1 lists the criteria upon which the ratings were made, it is not fully objective. Especially the decision which information to consider relevant and which less relevant is still dependent on the author. Interestingly, there are figure types which have a high rating in all categories but also figures which have a high rating in one category and low ones in the other two. An example for the first is the *reward over* *episode divided in components* type. For the second, the *rewards over actions* type stands as an example. Although figure types showing high ratings in all categories seem even more useful, figures rating good in only one category are still necessary since the relevant information they show cannot be observed otherwise. Besides the *reward over episode* type, the information of which can be clearly observed in the *reward over episode divided in components* type, all figure types are considered to be useful and proposed to use in future projects of the RACOON-Lab.

7.2 Reward function

7.2.1 Impact of the correction of the reward function

As mentioned at the beginning of chapter 6, the results presented in chapters 6.1, 6.2, 6.5, and 6.6 were recorded with an error in the reward function which influenced the rotational components, i.e., D_{rot} and S_{rot}. In chapter 6.3, the corrected versions of the figures showing reward components for one of the datasets were described and compared to the uncorrected versions. For the whole training clear differences could only be observed for the D_{rot} component. These, however, were not larger than the differences between the figures for the two datasets 6–7 and 6–22. The following will evaluate the influence of the error on the validity of the results presented in chapter 6.

Chapters 6.1 and 6.2 do not rely on the accuracy of the data since they are only meant to demonstrate the capabilities of the different figure types. So, although most figures show incorrect data due to the error in the reward function, the intended purpose is still fulfilled for these chapters. The discussion of the figures' expressiveness in chapter 7.1 thus stays valid.

The rewards for the good trajectory in chapter 6.4 were calculated with a corrected version of the reward function. Therefore, the error had no impact on the data. The description of the results and their discussion in chapter 7.2.2 therefore stay valid without any limitations.

In chapter 6.5, the number of parallel environments was iterated. The reward was not addressed in this chapter. The error in the reward function is not expected to have an impact on the performance of the toolchain. Thus, the results presented in chapter 6.5 and the discussion of the same in chapter 7.3.1 stay valid without any limitations.

Chapter 6.6 shows the results of an iteration of the learning rate. These depend on the total reward and hence affected by the error in the reward function. The impact of the error is expected to be small since the differences of the corrected version of boxplot of the reward components compared to the uncorrected version were not larger than the ones between the boxplots of the two datasets. Nevertheless, any conclusion made in 7.3.2 that is based on the data described in chapter 6.6 is only valid to a limited extent.

Tab. 7–1: Summary of figures' expressiveness

Figure type	Expressiveness of the individual figure	Expressiveness of the figure when being compared to figures of the same type	Expressiveness when being compared to or combined with figures of other types
Action histogram of whole training	$-$	$-$	o
Action histogram of individual episode	o	$+$	$+$
Actions over episode	$+$	$+$	$+$
Rewards over actions	$++$	$-$	$--$
Reward histogram of whole training	$-$	$-$	o
Reward histogram of individual episode	o	$+$	o
Reward components boxplot of whole training	$++$	$-$	$++$
Reward component boxplot of individual episode	$++$	$+$	$+$
Mean reward	$++$	o	$--$
Reward over episode	o	o	o
Reward over episode divided in components	$++$	$+$	$+$

7.2.2 Evaluation of the reward function

In the following, the reward function and its component will be discussed. Problems in the current reward function are observed and a proposal is made how to adopt the reward function to address these problems.

Most important for the evaluation of the reward function are the results presented in chapter 6.4. Here, according to the best-fit-circle-metric developed by *Rehn* in [16], a good trajectory was analyzed with the reward function. The trajectory is considered good according to the metric developed by *Rehn* because the distance of the trajectory's points from the best-fit-circle is very low. The orientation of the points on that trajectory is not considered by this metric. The reward function used in this thesis as introduced in chapter 4.3 compares the observed velocity and angular velocity to reference ones calculated with parameters set during the recording in the RACOON-Lab. It considers not only the coordinates, but also the orientation of the points of the trajectory. In contrast to the metric developed by *Rehn*, it can be evaluated before the trajectory was fully observed, i.e., it can be used as a dense reward.

With figure 6–32 it gets clear that a trajectory which was considered to be good by the metric developed by *Rehn* does not necessarily receive high rewards by the reward function used in this thesis. In fact, the trajectory analyzed in chapter 6.4 receives a lot of negative rewards. Compared to other episodes analyzed in detail such as the last episode of dataset *21-11-12 14-18-56* as shown in figure 6–25, it actually performed worse. The reward signal used in this thesis thus does not seem to be a good metric for evaluating the correctness of a trajectory. The following will discuss why that is the case. For that purpose, the intent behind the four components of the reward and difficulties in practice will be analyzed. Afterwards, a conclusion is drawn and an option to handle the difficulties is proposed.

The D_{trans} component checks that the absolute value of the velocity of the chaser is similar to the one of the reference velocity known from the recording, which is constant. In practice, this means that all points of the trajectory need to have a set constant distance. The trajectory analyzed in chapter 6.4 does not fulfill this criterion as proofed by the high values of the boxplot for the D_{trans} component in figure 6–32b. Even if the distance was not set and it would be only evaluated whether the points of the trajectory have a constant distance, the trajectory analyzed in chapter 6.4 would not have performed well. The reason for that is that the points of the trajectory do not have a constant distance as observed in figure 6–30b. If the trajectory calculated with DIFODO is circular, the points of this trajectory do not necessarily seem to be evenly spaced. Fluctuations in the distance of the points of a trajectory are expected when using a algorithm like the DIFODO. According to the metric developed by *Rehn*, a good trajectory does not need to fulfill the criterion of the D_{trans} component. In an application of the RACOON-Lab's toolchain, a constant distance of the points of the trajectory also is not a criterion. Hence, the D_{trans} component would only be useful for perfect trajectories with constant distance between the points but does not seem very useful to evaluate the quality of a trajectory. Therefore, using it to evaluate the action chosen by the RL agent does not make sense.

The D_{rot} component serves the same purpose as the D_{trans} component but for the angular velocity. It compares the absolute value of the angular velocity calculated from the rotations computed by the DIFODO algorithm to the one of the reference angular velocity, which is constant. As observed in figure 6–31b, the rotations computed by the DIFODO do not have constant distance on the degree axis, which the D_{rot} component uses to compute the angular velocity. As for the D_{trans} component, these fluctuations are not unexpected when using an algorithm like the DIFODO. The metric developed by *Rehn* does not take the rotations into account. For the reward signal, they were intended to ensure the circularity of the trajectory. With fluctuating angles, however, this intent cannot be fulfilled by the D_{rot} component. Therefore, it is neither useful to evaluate the quality of a trajectory nor to measure the performance of an action chosen by the RL agent.

The reward's S_{trans} component compares the direction of the computed velocity vector to one of the reference velocity vector. Since the reference velocity vector is constant, this component in its current version does not measure the circularity of the trajectory. The changing behavior observed in figure 6–32a is caused by this constancy since the calculated velocity has to be rotating in order to make a circular trajectory. An easy adoption seems to be to rotate the reference velocity vector with a rotation matrix by γ_{ref}, i.e., by the angle the chaser is expected to turn between two timesteps. However, one major question remains open: with which value for the reference velocity to start. If the velocity rotates by exactly γ_{ref} every timestep the S_{trans} component would be constant over the episode. If the reference velocity was chosen incorrectly at the beginning of the episode, this constant value would not be 0. In the extreme case that the reference velocity vector points exactly in the opposite direction of the calculated velocity vector, the S_{trans} component would be 1 for the whole episode. In figure 6–32a, for example, the ideal reference velocity to start with would have been turned -800 times, i.e., 800 times in the opposite direction in order to minimize the S_{trans} component since the minimum S_{trans} component value for the non-rotated reference velocity vector was observed at timestep 800. It is impossible to know the ideal number of rotations for the velocity vector before the total trajectory was observed. Even an estimate is impossible to make a priori. Hence, the S_{trans} component cannot be adopted in a way that it actually measures the circularity of the trajectory. Additionally, the fluctuations of the points of the trajectory observed in figure 6–30b might also influence the direction of the calculated velocity vector and thus also the S_{trans} component of the reward. Therefore, it does not make sense to use the S_{trans} component of the reward to evaluate the actions chosen by the RL agent. To evaluate the quality of a complete trajectory, however, it could be adopted since the a priori knowledge is available here as long as the fluctuations are manageable.

The direction of the reference angular velocity is compared to the one of the calculated angular velocity by the S_{rot} reward component. The latter one is calculated using the changes of the three rotational angles. This component aims to measure the circularity of the trajectory, just like the S_{trans} component but with different data. For a perfect S_{rot} component, the pitch angle would need to rise linearly and the other two angles would need to be constant. For the first timesteps of figure 6–31a, this was approximately the case which is why the S_{rot} component in figure 6–32a is low for these timesteps. In the second half of the episode, the yaw and roll angles are almost constant but the pitch angle is decreasing linearly. Therefore, the S_{rot} component shows high values in figure 6–32a for the second half of the episode. This component, in contrast to the S_{trans} component does not need knowledge which is unavailable a priori. However, the rotation angles computed by the DIFODO algorithm and used to calculate this component are not as accurate as necessary. The change in the pitch angle as well as the switch of the yaw and roll angle as observed in figure 6–31a underline that. In the metric developed by *Rehn*, the rotation of the chaser is not included. If knowledge of the rotation of the chaser is necessary for the intended use-case of the RAOON-Lab's toolchain, an adopted, less fluctuating version of the S_{rot} component might be suitable to be used as a metric for the rotation. In that case, the adopted version would also be suitable to be one component of a reward function.

To summarize: the current components do not align with the metric developed by *Rehn*. Hence, a trajectory which is considered good by the metric developed by *Rehn* can get negative rewards by the reward function of this thesis as observed in chapter 6.4. The D_{trans}, D_{rot}, and S_{rot} components mainly suffer under the fluctuations and unequal distances of the points and rotations of the trajectory and cannot be used as part of a reward function as long as this problem exists. Since the DIFODO algorithm or an alternative are not expected to output trajectories without any fluctuations, the problem needs to be addressed by a the reward function itself. The S_{trans} component is not useful as a component of the reward function since it needs to be rotated and the ideal starting value for the reference velocity is crucial but unknown a priori. It can still be adopted as part of a metric to evaluate a complete trajectory as long as the fluctuations are manageable since the ideal starting value for the reference velocity can be calculated in that case. One option to handle the fluctuations could be to not take the difference between two sequential points of the trajectory but over a a larger span of for example 10 points. The idea is that the fluctuations would arguably be lower and thus might not influence the reward components that much. The training of the RL agent could not start immediately but only after enough points of the trajectory were observed. Since these 9 additional timesteps without training the RL agent for the example of a 10 points span would not make much of a difference in episodes of more than 1000 timesteps, the author considers this a minor problem. If this or another handling of the fluctuations works, the reward function could still be dense but more helpful to actually train the RL agent, once the implementation of the DIFODO algorithm was tested.

7.3 Parameters

In chapters 6.5 and 6.6, parameters were iterated. The following will discuss these results and propose how to set them in a future usage of the toolchain.

7.3.1 Number of parallel environments

The number of parallel environments was iterated between 1 and 6 for two training runs. The resulting performance measured in timesteps per second are shown in figure 6–33 together with the mean values. While the values of the trainings are fluctuating, the mean value is rising linearly from 1 to 6 parallel environments. The fluctuations seem to be typical performance fluctuations. A rising performance for a higher number of parallel environments was expected. However, this trend is limited by the GPU's capacity which typically is expected to be the limiting factor for ML applications. Figure 6–33 shows that this limit is not reached by 6 parallel environments. This is probably caused by a slow performance of the DIFODO implementation used in this thesis which suffers under low parallelization of the code. Therefore, the processing of one image takes almost $1s$ which is very slow compared to the 30 frames per second reached by the DIFODO implementation used in previous studies in the RACOON-Lab.

As long as that is the case, the number of parallel environments should be set as high as possible with the memory limitation of the cluster. With the current job sizes available, the memory limit for the largest job size was reached by 6 parallel environments. When either the DIFODO implementation or the job sizes of the cluster are updated, a new iteration of this parameter should be executed to find the optimal value for the number of parallel environments. The iteration of the number of parallel environments thus is a suitable option to optimize this parameter and is introduced as part of the developed toolchain.

7.3.2 Learning rate

The learning rate was iterated in figure 6–34 with a step factor of 10. One would expect a smaller learning rate to ensure a training improvement, probably with smaller performance. The results in figure 6–34 cannot confirm this expectation. This is arguably caused by the reward function which does not seem to work well with fluctuating coordinates and rotation angles as observed in the discussion of the reward function in chapter 7.2.2. Additionally, the implementation of the DIFODO algorithm used in this thesis was not tested and therefore may not perform as the original implementation. This could be an additional factor for the unexpected behavior of the learning curves in figure 6–34.

An optimal value for the learning rate cannot be observed in figure 6–34. Hence, as soon as the reward function was improved and the DIFODO implementation was tested, a new iteration should be made to find the optimal value for the learning rate. However, the *mean reward* figure type seem to be suitable for that purpose and thus serves as a part of the toolchain to optimize the learning rate.

8 Conclusion

In this thesis, the first steps towards the usage of RL for depth image enhancement in the RACOON-Lab's toolchain have been taken. The objective of this thesis as described in chapter 3 was to build a tool capable of evaluating the usage of RL for depth image enhancement in the RACOON-Lab's toolchain as well as to test and verify the same as far as possible with the untested implementation of the DIFODO algorithm.

A toolchain to train the RL agent was proposed. The SB3 implementation of the PPO algorithm was chosen. A dense reward function built of four components was selected. The components measure the difference of the absolute value and direction between the observed and expected velocity and angular velocity. A RL environment was implemented following the *Gym* interface of *OpenAI* which can be used independently of the RL algorithm and implementation used in this thesis. Five actions have been proposed for the agent: *Do nothing*, a *Adaptive threshold filter*, a *IP_Basic filter*, a *Median filter* and a *Percentile filter*. The toolchain was implemented for an execution on the chair's *Condor* GPU cluster. This implementation was tested and verified as far as possible with the untested implementation of the DIFODO algorithm, i.e., a training improvement could not be observed.

Additionally, figures to evaluate both the training of the RL agent and the toolchain itself have been proposed. Functions were implemented to automatically create the same using the data recorded during the training and demonstrated with two datasets. The figures' expressiveness has been discussed and ranked in three categories: . Especially figure types showing the reward components, i.e., the *reward components boxplots* and the *reward components divided in components* figure type, have proven to be very useful to evaluate both the agent's performance and the toolchain.

Furthermore, the reward function was tested with a trajectory calculated by the DIFODO implementation used in previous studies of the RACOON-Lab. The test showed large differences between the metric used in previous studies of the RACOON-Lab and the reward function used in this thesis. In the discussion, fluctuations in the points and rotation angles calculated by the DIFODO algorithm were identified as the main reason. These make the reward function in the proposed state unsuitable to determine the quality of a trajectory. Therefore, it was proposed to adapt the reward function to compute the difference over multiple timesteps instead of only one in future studies of the RACOON-Lab.

8.1 Outlook

The next step for the evaluation of the usage of RL in the RACOON-Lab's toolchain would be to adopt the reward function. Any adoption of the reward function should from now on be verified with existing trajectories before being used for the training. This is meant to validate the desired functionality under the data quality of the trajectories calculated by the DIFODO algorithm.

As soon as an adopted reward function was verified and the DIFODO implementation was tested and verified to work as intended, the toolchain can be used to train a RL agent for depth image enhancement. Before starting a long training, it is recommended to tune the learning rate parameter using the proposed learning rate iteration.

The next step would be to perform multiple long trainings of the RL agent with probably several millions of timesteps. With the current performance of the DIFODO implementation used in this thesis, a long training with millions of timesteps would take weeks due to the low parallelization in the DIFODO implementation. Thus, either enough time for training should be planned in or a more performant implementation of the DIFODO algorithm should be used.

Once long trainings have been recorded, the proposed figures should be used to evaluate the same. The training should be extended until no further training improvement can be observed. The performance of the toolchain using the trained RL agent should then be compared to the toolchain which does not use RL in order to fully evaluate the usage of RL for depth image enhancement in the RACOON-Lab's toolchain.

The depth image filters used in this thesis have parameters which were fixed as part of this thesis, such as the maximum percentile of the *Percentile filter*. The developed toolchain does not offer any option to tune these. Since they are heavily influencing the performance of the RL agent, future studies of the RACOON-Lab should develop a framework to tune these parameters or let the RL agent choose them too.

Another proposal besides the evaluation of RL for the RACOON-Lab is to enhance the metric currently used in the RACOON-Lab by a orientation measurement. In an application of the toolchain not only the coordinates but also the orientation will be of interest. An adoption of the S_{trans} component of the reward function used in this thesis could be a valid candidate for that orientation measurement.

Bibliography

[1] Donald J. Kessler and Burton G. Cour-Palais. “Collision frequency of artificial satellites: The creation of a debris belt”. In: *Journal of Geophysical Research: Space Physics* 83.A6 (1978), pp. 2637–2646. DOI: 10.1029/JA083iA06p02637.

[2] Donald J. Kessler et al. “The kessler syndrome: implications to future space operations”. In: *Advances in the Astronautical Sciences* 137.8 (2010), p. 2010. URL: https://www.threecountrytrustedbroker.com/media/kesseler_syndrome.pdf (visited on 3.12.2021).

[3] Jonathan Amos. “Russian anti-satellite missile test draws condemnation”. In: *BBC.com* (16.11.2021). URL: https://www.bbc.com/news/science-environment-59299101 (visited on 3.12.2021).

[4] ESA Space Debirs Office. *ESA’S ANNUAL SPACE ENVIRONMENT REPORT*. URL: https://www.sdo.esoc.esa.int/environment_report/Space_Environment_Report_latest.pdf (visited on 3.12.2021).

[5] Martin Dziura, Tim Wiese, and Jan Harder. “3D reconstruction in orbital proximity operations”. In: *2017 IEEE Aerospace Conference*. Piscataway, NJ: IEEE, 2017, pp. 1–10. ISBN: 978-1-5090-1613-6. DOI: 10.1109/AERO.2017.7943679. (visited on 10.12.2021).

[6] Lucio Franceschini. “Experimental Analysis of Pose Tracking Performance of Visual and Infrared Stereo Cameras under Low Orbital Light Conditions”. Masterarbeit. Technical University of Munich, 2020. URL: https://mediatum.ub.tum.de/doc/1543085/1543085.pdf (visited on 11.12.2021).

[7] Andrew Tatsch, Norman Fitz-Coy, and Svetlana Gladun. “On-orbit servicing: A brief survey”. In: *Performance Metrics for Intelligent Systems Workshop* (2006), pp. 276–281. URL: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication1062.pdf#page=278 (visited on 11.12.2021).

[8] Enrico Stoll et al. “On-orbit servicing”. In: *IEEE Robotics & Automation Magazine* 16.4 (2009), pp. 29–33. ISSN: 1070-9932. DOI: 10.1109/mra.2009.934819.

[9] Toru Kasai, Mitsushige Oda, and Takashi Suzuki. “Results of the ETS-7 Mission-Rendezvous docking and space robotics experiments”. In: *Artificial Intelligence, Robotics and Automation in Space* 440 (1999), pp. 299–306.

[10] G. S. Aglietti et al. “RemoveDEBRIS: An in-orbit demonstration of technologies for the removal of space debris”. In: *The Aeronautical Journal* 124.1271 (2020), pp. 1–23. DOI: 10.1017/aer.2019.136. URL: https://hal.inria.fr/hal-02406580/document (visited on 11.12.2021).

[11] *Astroscale Celebrates Successful Launch of ELSA-d - Astroscale*. 23.03.2021. URL: https://astroscale.com/astroscale-celebrates-successful-launch-of-elsa-d/ (visited on 13.7.2021).

[12] *Northrop Grumman and Intelsat Make History with Docking of Second Mission Extension Vehicle to Extend Life of Satellite*. 12.04.2021. URL: https://news.northropgrumman.com/news/releases/northrop-grumman-and-intelsat-make-history-with-docking-of-second-mission-extension-vehicle-to-extend-life-of-satellite (visited on 13.7.2021).

[13] Robin Biesbroeck et al. "e. Deorbit-ESA's active debris removal mission". In: *Journal of the British Interplanetary Society* (2017), pp. 143–151. URL: https://conference.sdo.esoc.esa.int/proceedings/sdc7/paper/1053/SDC7-paper1053.pdf.

[14] Robin Biesbroeck et al. *THE CLEARSPACE-1 MISSION: ESA AND CLEARSPACE TEAM UP TO REMOVE DEBRIS*. 2021. URL: https://conference.sdo.esoc.esa.int/proceedings/sdc8/paper/320/SDC8-paper320.pdf (visited on 11.12.2021).

[15] Heike Benninghoff et al. "European Proximity Operations Simulator 2.0 (EPOS) - A Robotic-Based Rendezvous and Docking Simulator". In: *Journal of Large-Scale Research Facilities JLSRF* 3, A107 (2017). DOI: 10.17815/jlsrf-3-155. URL: https://elib.dlr.de/111852/1/Final_Version_Online.pdf (visited on 11.12.2021).

[16] Flavio Rehn. "Experimentelle Studie mit RACOON und EPOS zur quantitativen Bewertung von Einflussfaktoren auf die Qualität einer 3D-Objektrekonstruktion im Erdorbit". Bachelorarbeit. Technische Universität München, 2017. URL: https://mediatum.ub.tum.de/doc/1443240/1443240.pdf (visited on 11.12.2021).

[17] Ariadna Pregel Hoderlein. "Implementation and Evaluation of Neural Networks for Enhancement of Depth Images in Orbital Proximity Operations". Semester's Thesis. Technical University of Munich, 2021.

[18] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. URL: https://arxiv.org/pdf/1312.5602.pdf?source=post_page (visited on 11.12.2021).

[19] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. Second. Adaptive computation and machine learning. Cambridge, Massachusetts: MIT Press, 2018. ISBN: 9780262352703.

[20] Ryosuke Furuta, Naoto Inoue, and Toshihiko Yamasaki. "PixelRL: Fully Convolutional Network With Reinforcement Learning for Image Processing". In: *IEEE Transactions on Multimedia* 22.7 (2020), pp. 1704–1719. DOI: 10.1109/TMM.2019.2960636. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8936404 (visited on 11.12.2021).

[21] Debang Li et al. "A2-RL: Aesthetics Aware Reinforcement Learning for Image Cropping". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 8193–8201. URL: https://openaccess.thecvf.com/content_cvpr_2018/papers/Li_A2-RL_Aesthetics_Aware_CVPR_2018_paper.pdf (visited on 11.12.2021).

[22] Jongchan Park et al. “Distort-and-Recover: Color Enhancement Using Deep Reinforcement Learning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 5928–5936. URL: https://openaccess.thecvf.com/content_cvpr_2018/papers/Park_Distort-and-Recover_Color_Enhancement_CVPR_2018_paper.pdf (visited on 11.12.2321).

[23] Qingxing Cao et al. “Attention-Aware Face Hallucination via Deep Reinforcement Learning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 690–698. URL: https://openaccess.thecvf.com/content_cvpr_2017/papers/Cao_Attention-Aware_Face_Hallucination_CVPR_2017_paper.pdf (visited on 11.12.2021).

[24] Ke Yu et al. “Crafting a Toolchain for Image Restoration by Deep Reinforcement Learning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 2443–2452. URL: https://openaccess.thecvf.com/content_cvpr_2018/papers/Yu_Crafting_a_Toolchain_CVPR_2018_paper.pdf (visited on 11.12.2021).

[25] F. Sahba and H. R. Tizhoosh. “Filter fusion for image enhancement using reinforcement learning”. In: *Proceedings / CCECE 2003, Canadian Conference on Electrical and Computer Engineering*. Ed. by Guy Olivier. Piscataway, NJ: IEEE Operations Center, 2003, pp. 847–850. ISBN: 0-7803-7781-8. DOI: 10.1109/CCECE.2003.1226027.

[26] David Silver. *Introduction to Reinforcement Learning*. University College London, 2015. URL: https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver.

[27] David Silver. *Lecture 2: Markov Decision Processes*. University College London, 2015. URL: https://www.davidsilver.uk/wp-content/uploads/2020/03/MDP.pdf.

[28] David Silver. *Lecture 1: Introduction to Reinforcement Learning*. University College London, 2015. URL: https://www.davidsilver.uk/wp-content/uploads/2020/03/intro_RL.pdf.

[29] David Silver. *Lecture 5: Model-Free Control*. University College London, 2015. URL: https://www.davidsilver.uk/wp-content/uploads/2020/03/control.pdf.

[30] David Silver. *Lecture 6: Value Function Approximation*. University College London, 2015. URL: https://www.davidsilver.uk/wp-content/uploads/2020/03/FA.pdf.

[31] David Silver. *Lecture 4:Model-Free Prediction*. University College London, 2015. URL: https://www.davidsilver.uk/wp-content/uploads/2020/03/MC-TD.pdf.

[32] David Silver. *Leture 7: Policy Gradient*. University College London, 2015. URL: https://www.davidsilver.uk/wp-content/uploads/2020/03/pg.pdf.

[33] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. First. Cambridge, Massachusetts: MIT Press, 1998. ISBN: 0262193981.

[34] David Silver. *Lecture 8: Integrating Learning and Planning*. University College London, 2015. URL: https://www.davidsilver.uk/wp-content/uploads/2020/03/dyna.pdf.

[35] David L. Poole and Alan K. Machworth. *Artificial Intellignece: Foundations of Computational Agents*. Second. Camebridge University Press, 2017. ISBN: 9781107195394.

[36] M. G. Bellemare et al. “The Arcade Learning Environment: An Evaluation Platform for General Agents”. In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[37] Prafulla Dhariwal et al. *OpenAI Baselines*. Ed. by GitHub. 2017. URL: https://github.com/openai/baselines (visited on 24.8.2021).

[38] *Stable Baslines 2 Documentation*. URL: https://stable-baselines.readthedocs.io/en/master/index.html (visited on 18.8.2021).

[39] *Stable Baselines 2 Documentation: Which algorithm should I use?* URL: https://stable-baselines.readthedocs.io/en/master/guide/rl_tips.html#which-algorithm-should-i-use (visited on 18.8.2021).

[40] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: *CoRR* abs/1707.06347 (2017). URL: https://arxiv.org/pdf/1707.06347.pdf (visited on 16.8.2021).

[41] John Schulman et al. “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. In: *Proceeding of the International Conference on Learning Representations 2016*. URL: https://arxiv.org/pdf/1506.02438.pdf (visited on 18.7.2021).

[42] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *International Conference on Machine Learning* (2016), pp. 1928–1937. ISSN: 1938-7228. URL: http://proceedings.mlr.press/v48/mniha16.pdf (visited on 11.12.2021).

[43] Yuhuai Wu et al. “Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation”. In: *Advances in Neural Information Processing Systems* 30 (2017), pp. 5279–5288. URL: https://proceedings.neurips.cc/paper/2017/file/361440528766bbaaaa1901845cf4152b-Paper.pdf (visited on 11.12.2021).

[44] James Martens and Roger B. Grosse. “Optimizing Neural Networks with Kronecker-factored Approximate Curvature”. In: *International Conference on Machine Learning* (2015), pp. 2408–2417. ISSN: 1938-7228. URL: http://proceedings.mlr.press/v37/martens15.pdf (visited on 11.12.2021).

[45] Ziyu Wang et al. “Sample Efficient Actor-Critic with Experience Replay”. In: *Proceeding of the International Conference on Learning Representation (ICLR) 2017*. URL: https://arxiv.org/pdf/1611.01224.pdf (visited on 16.8.2021).

[46] Kaleigh Clary et al. “Let’s Play Again: Variability of Deep Reinforcement Learning Agents in Atari Environments”. In: *NeurIPS 2018 Critiquing and Correcting Trends Workshop* (2019). URL: https://arxiv.org/pdf/1904.06312.pdf (visited on 21.8.2021).

[47] Yuhuai Wu et al. *OpenAI Baselines: ACKTR & A2C*. 2017. URL: https://openai.com/blog/baselines-acktr-a2c/ (visited on 16.8.2021).

[48] Peter Henderson et al. “Deep Reinforcement Learning that Matters”. In: *CoRR* abs/1709.06560 (2017). URL: https://arxiv.org/pdf/1709.06560.pdf (visited on 17.8.2021).

[49] Tanwi Mallick, Partha Pratim Das, and Arun Kumar Majumdar. “Characterizations of Noise in Kinect Depth Images: A Review”. In: *IEEE Sensors Journal* 14.6 (2014), pp. 1731–1740. ISSN: 1530-437X. DOI: 10.1109/jsen.2014.2309987. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6756961 (visited on 25.8.2021).

[50] Hengshuang Zhao et al. “Pyramid Scene Parsing Network”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017. DOI: 10.1109/cvpr.2017.660. URL: https://openaccess.thecvf.com/content_cvpr_2017/papers/Zhao_Pyramid_Scene_Parsing_CVPR_2017_paper.pdf (visited on 3.9.2021).

[51] Xin Yang et al. “Where Is My Mirror?” In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019. URL: https://openaccess.thecvf.com/content_ICCV_2019/papers/Yang_Where_Is_My_Mirror_ICCV_2019_paper.pdf (visited on 3.9.2021).

[52] Jiaying Lin, Guodong Wang, and Rynson W.H. Lau. “Progressive Mirror Detection”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2020. DOI: 10.1109/cvpr42600.2020.00375. URL: https://openaccess.thecvf.com/content_CVPR_2020/papers/Lin_Progressive_Mirror_Detection_CVPR_2020_paper.pdf (visited on 3.9.2021).

[53] *OpenCV-Python*. URL: https://github.com/opencv/opencv-python (visited on 2.9.2021).

[54] *OpenCV Documentation*. URL: https://docs.opencv.org/master/ (visited on 2.9.2021).

[55] Jason Ku, Ali Harakeh, and Steven L. Waslander. “In Defense of Classical Image Processing: Fast Depth Completion on the CPU”. In: *2018 15th Conference on Computer and Robot Vision (CRV)*. 2018, pp. 16–22.

[56] A. Geiger, P. Lenz, and R. Urtasun. “Are we ready for autonomous driving? The KITTI vision benchmark suite”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012. DOI: 10.1109/cvpr.2012.6248074.

[57] Thomas Simonini. *On Choosing a Deep Reinforcement Learning Library*. 2020. URL: https://blog.dataiku.com/on-choosing-a-deep-reinforcement-learning-library (visited on 30.8.2021).

[58] Sergio Guadarrama et al. *TF-Agents: A library for Reinforcement Learning in TensorFlow*. 2018. URL: `https://github.com/tensorflow/agents` (visited on 11.12.2021).

[59] Antonin Raffin. *Stable Baselines: a Fork of OpenAI Baselines — Reinforcement Learning Made Easy*. 2018. URL: `https://towardsdatascience.com/stable-baselines-a-fork-of-openai-baselines-reinforcement-learning-made-easy-df87c4b2fc82` (visited on 30.8.2021).

[60] Ashley Hill et al. *Stable Baselines*. 2018. URL: `https://github.com/hill-a/stable-baselines` (visited on 30.8.2021).

[61] Greg Brockman et al. *OpenAI Gym*. 2016. URL: `https://github.com/openai/gym` (visited on 30.8.2021).

[62] Antonin Raffin et al. *Stable Baselines3*. 2019. URL: `https://github.com/DLR-RM/stable-baselines3` (visited on 30.8.2021).

[63] *Stable Baselines 3 Documentation*. URL: `https://stable-baselines3.readthedocs.io/en/master/index.html` (visited on 30.8.2021).

[64] Antonin Raffin. *Performance check (Continuous Actions) #48*. 2020. URL: `https://github.com/DLR-RM/stable-baselines3/issues/48` (visited on 30.8.2021).

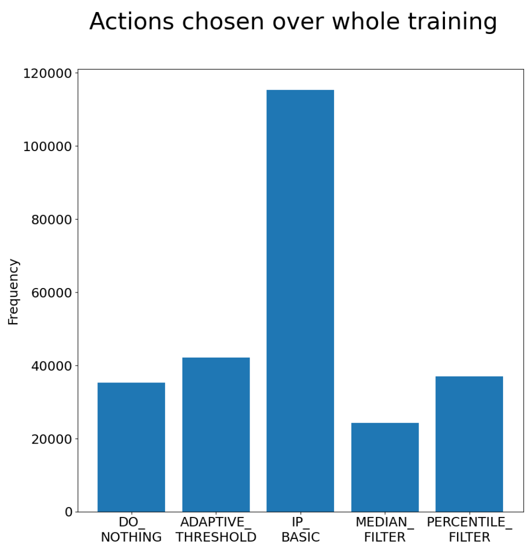
[65] Alexander Kuhnle, Michael Schaarschmidt, and Kai Fricke. *Tensorforce: a TensorFlow library for applied reinforcement learning*. 2017. URL: `https://github.com/tensorforce/tensorforce` (visited on 30.8.2021).

[66] *Tensorforce Documentation*. URL: `https://tensorforce.readthedocs.io/en/latest/index.html` (visited on 30.8.2021).

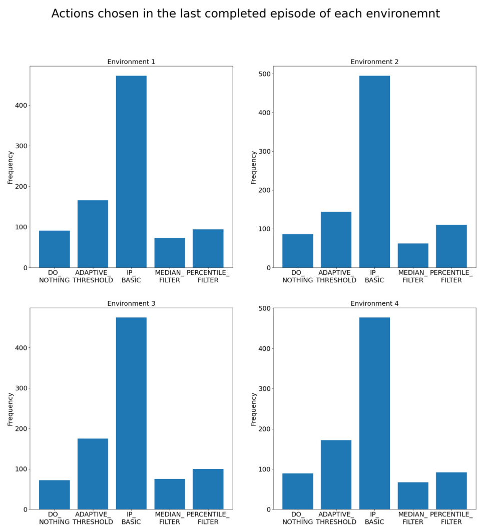
[67] *OpenAI Gym Documentation*. URL: `http://gym.openai.com/docs/` (visited on 30.8.2021).

A Results of all datasets

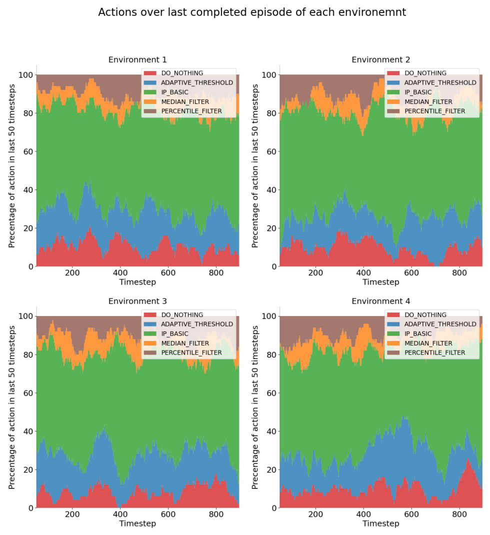
A.1 Dataset 21-11-11 04-41-55



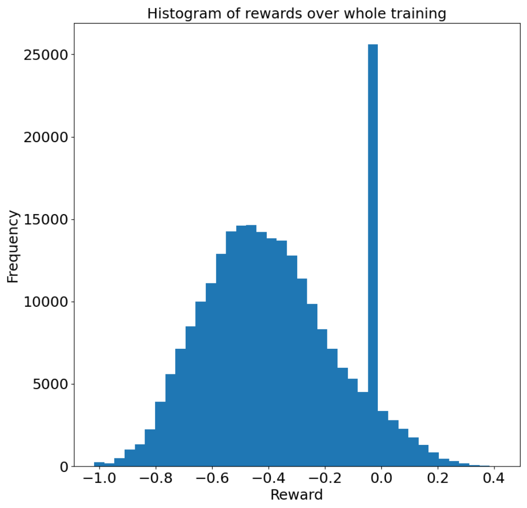
Actions chosen over whole training



Actions chosen in last episode

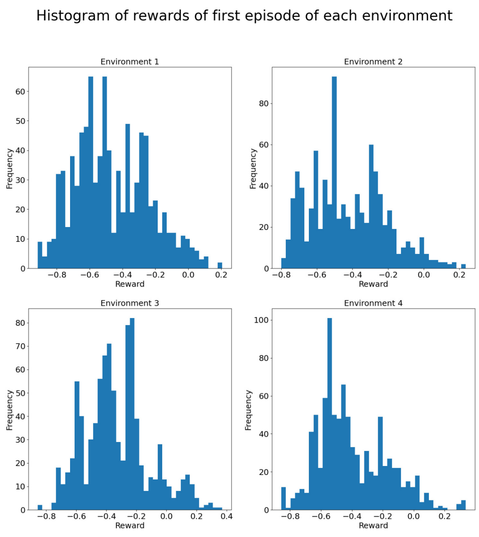


Actions over last episode

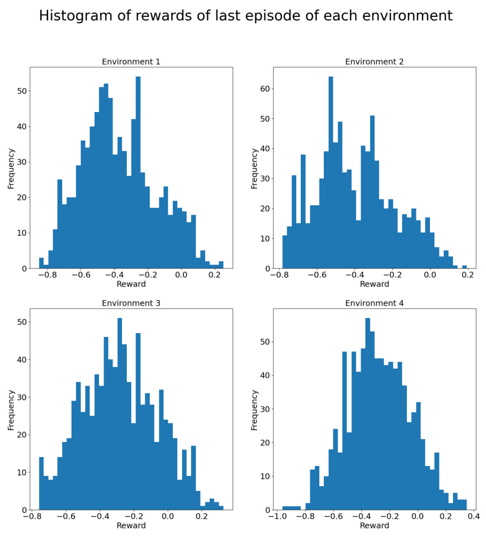


Reward histogram over whole training

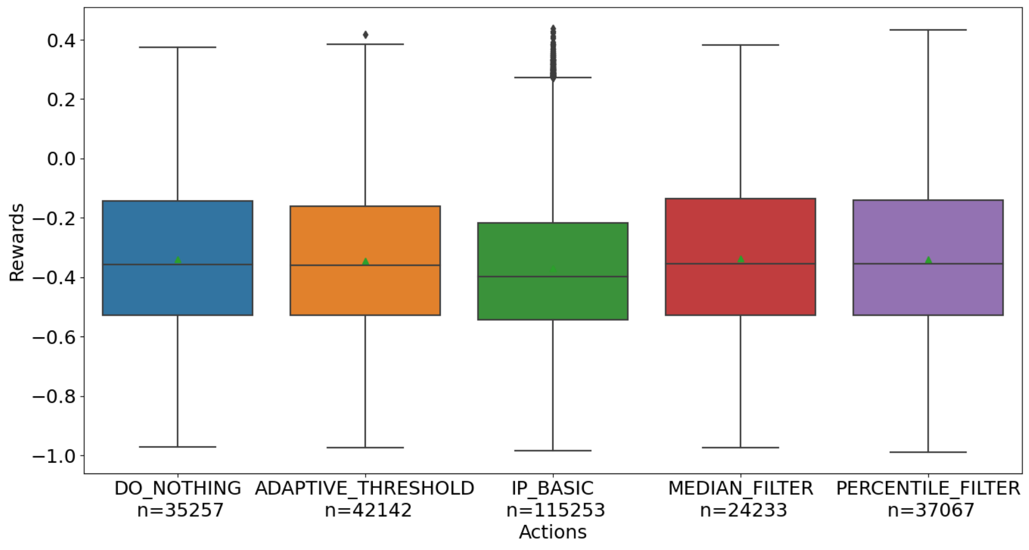
Results of all datasets



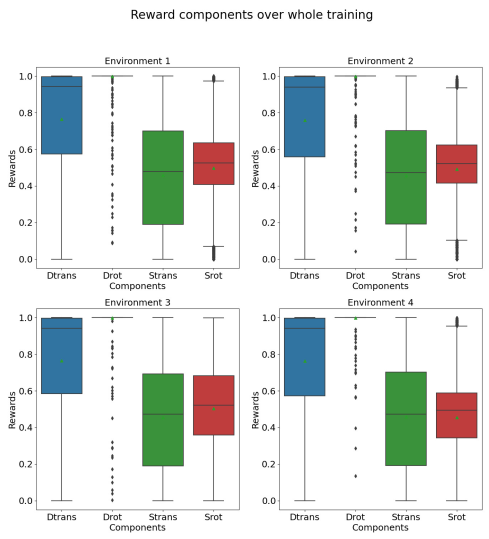
Reward histogram first episode



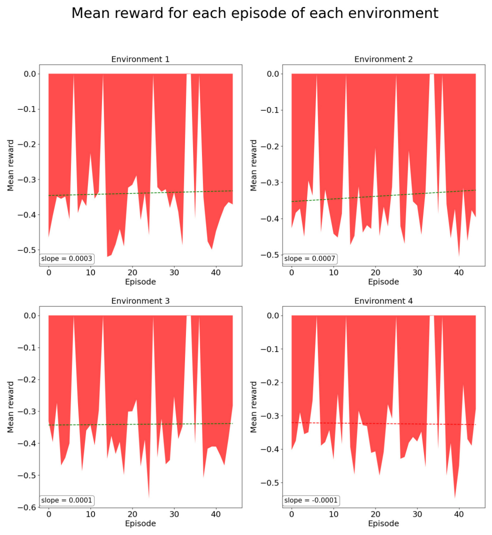
Reward histogram last episode



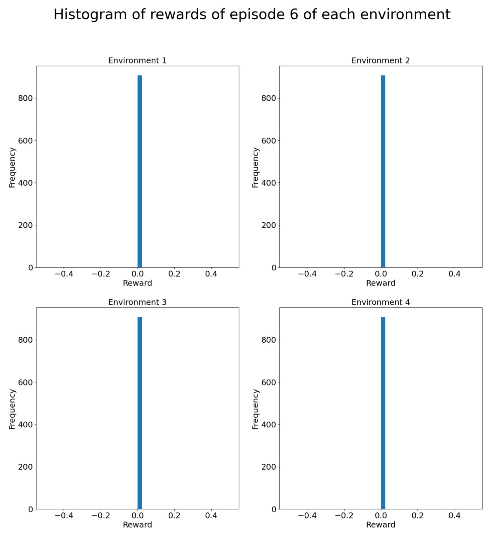
Boxplot of rewards over actions



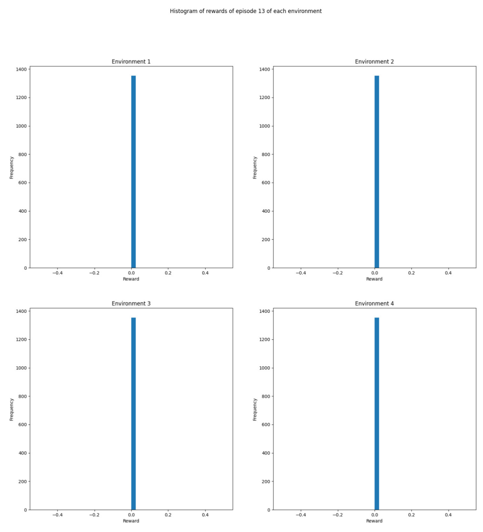
Boxplot of reward components over whole training



Mean reward of each episode

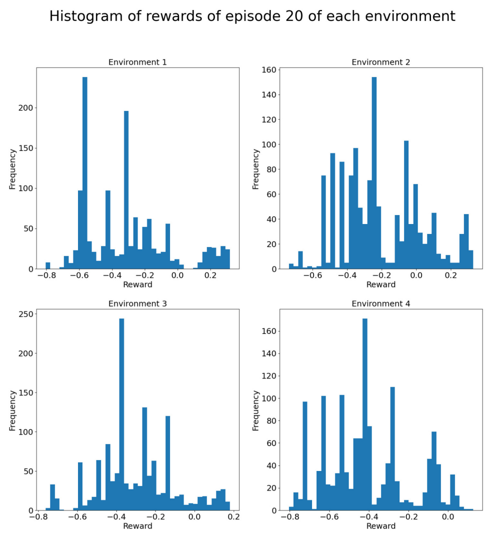


Reward histogram episode 6

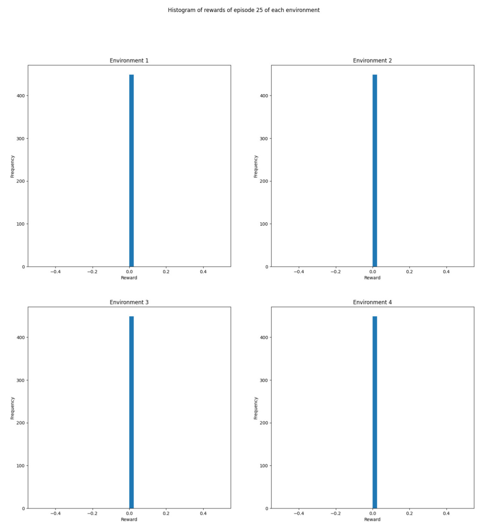


Reward histogram episode 13

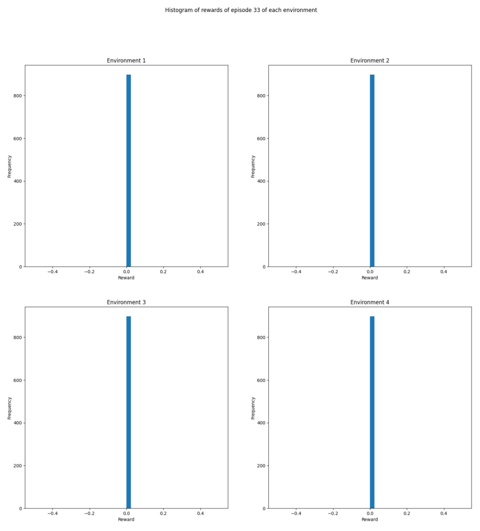
Results of all datasets



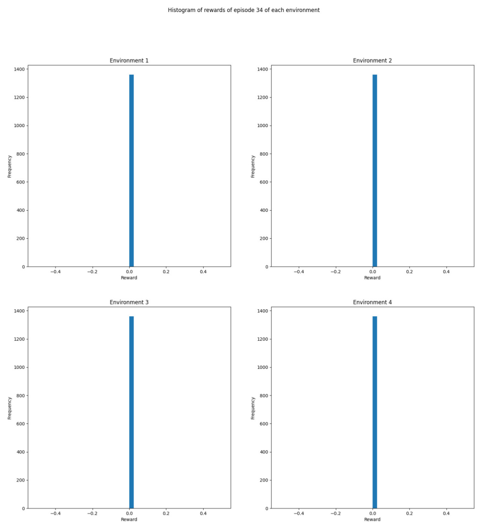
Reward histogram episode 20



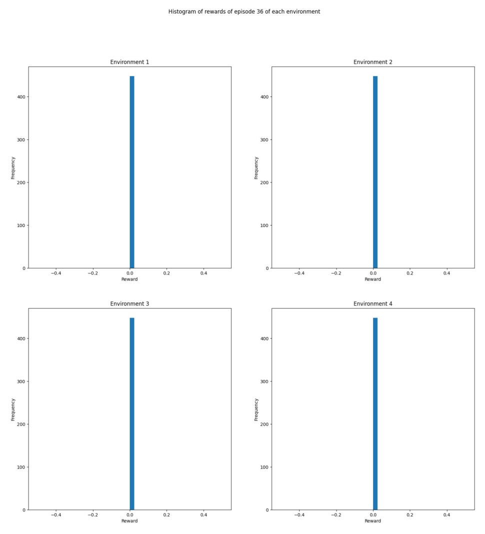
Reward histogram episode 25



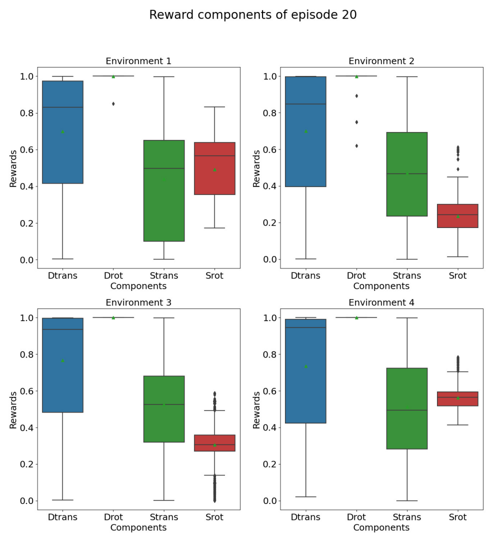
Reward histogram episode 33



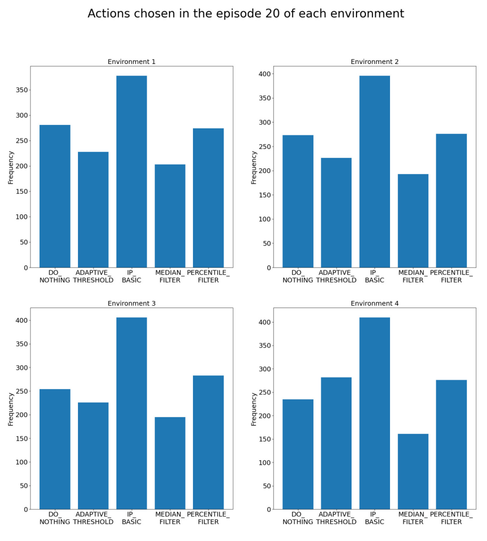
Reward histogram episode 34



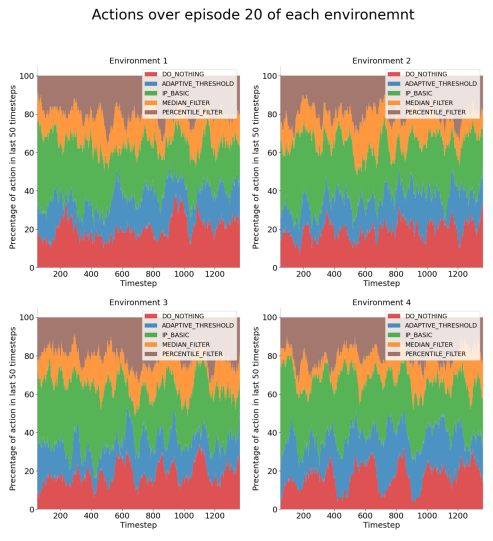
Reward histogram episode 36



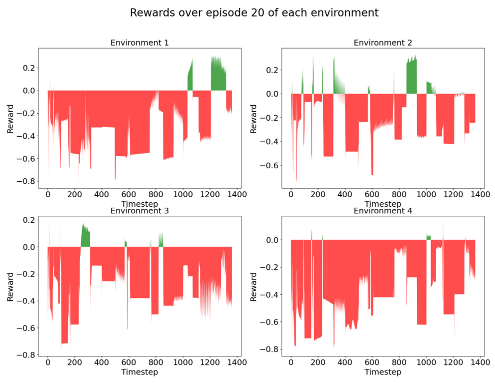
Boxplot of reward components of episode 20



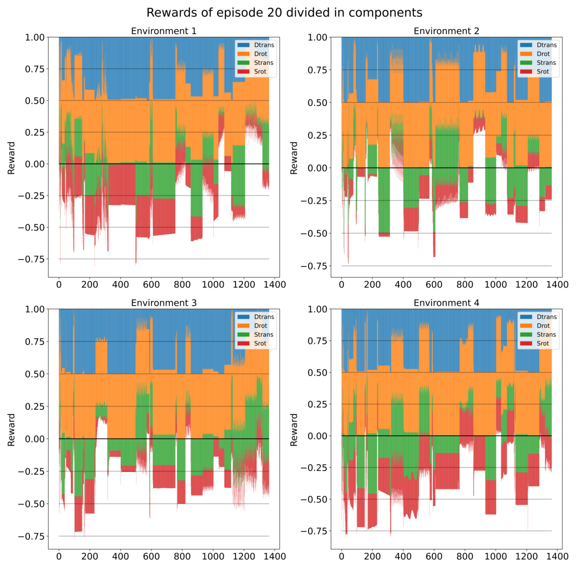
Actions chosen in episode 20



Actions over episode 20

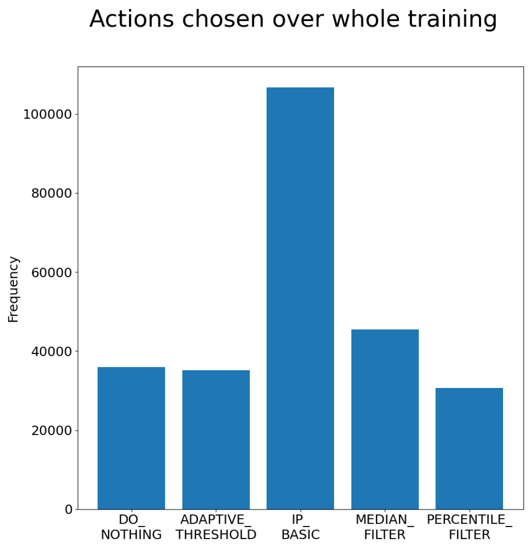


Rewards over episode 20

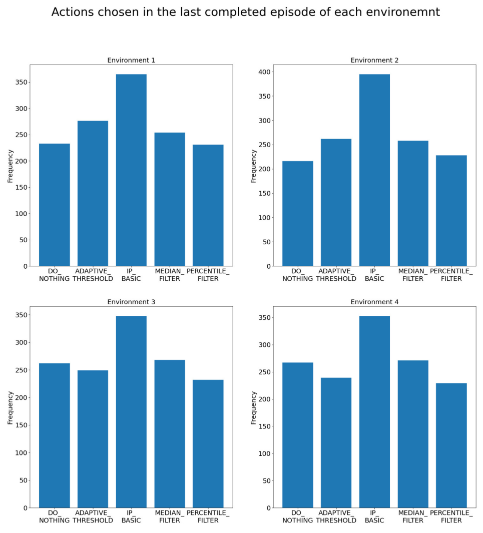


Rewards over episode 20 divided in components

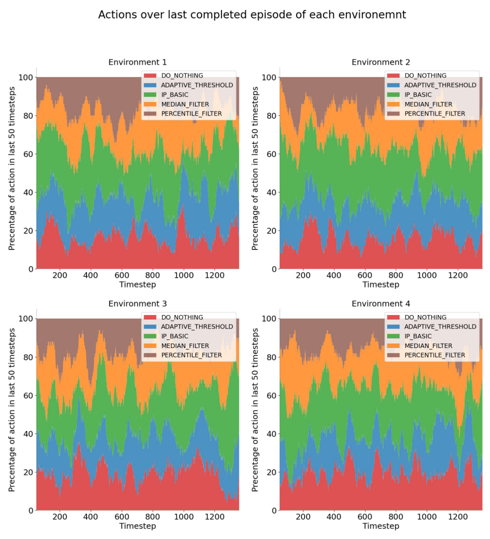
A.2 Dataset 21-11-12 14-18-56

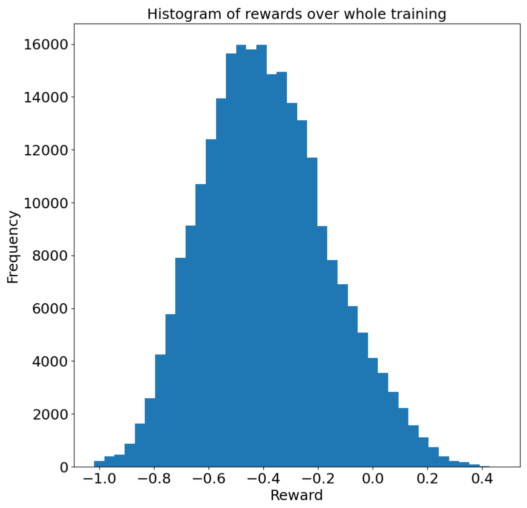


Actions chosen over whole training



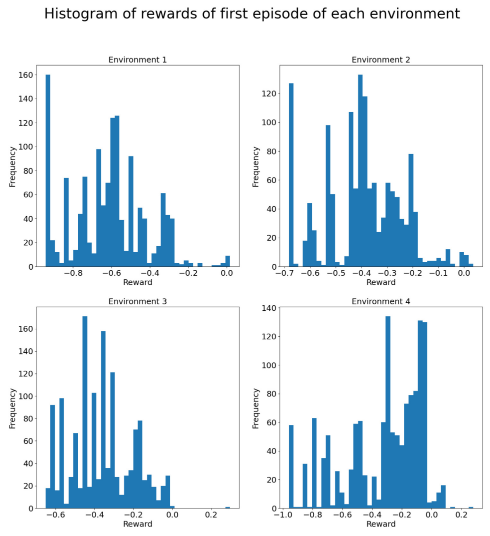
Actions chosen in last episode

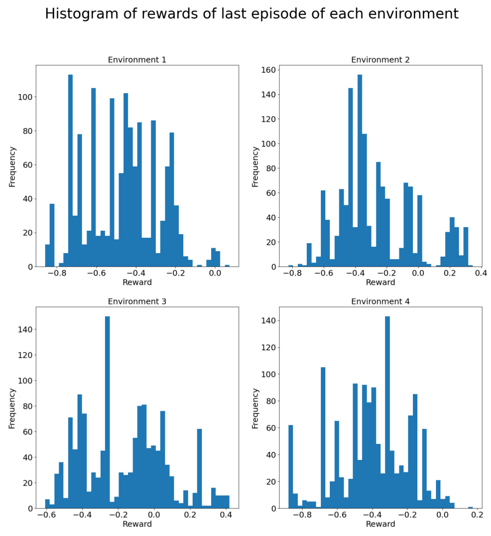




Actions over last episode

Reward histogram over whole training

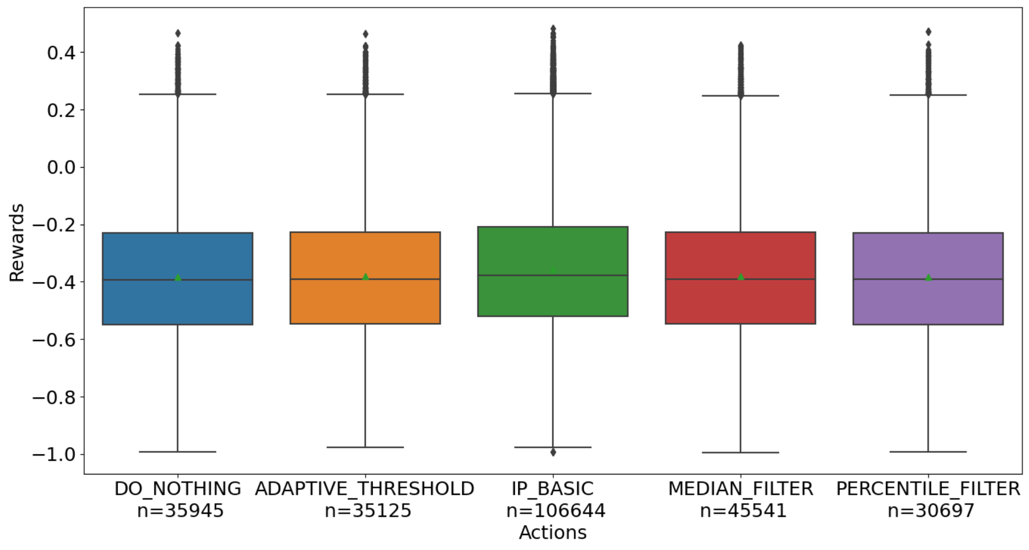




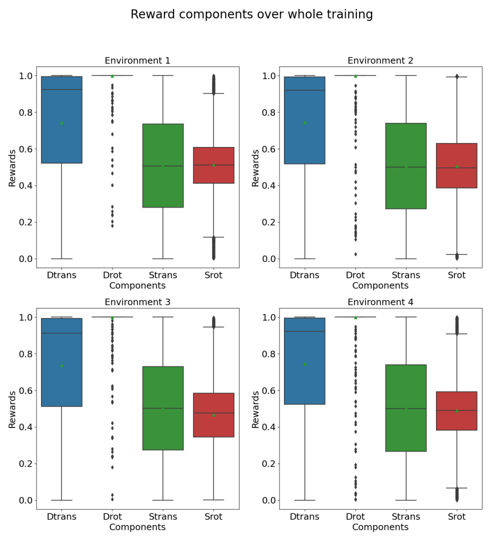
Reward histogram first episode

Reward histogram last episode

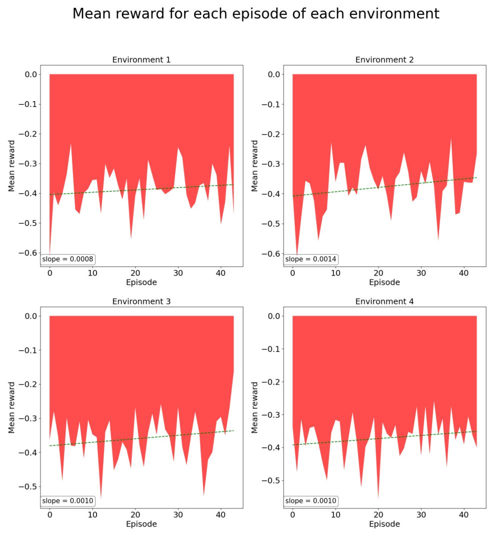
Results of all datasets



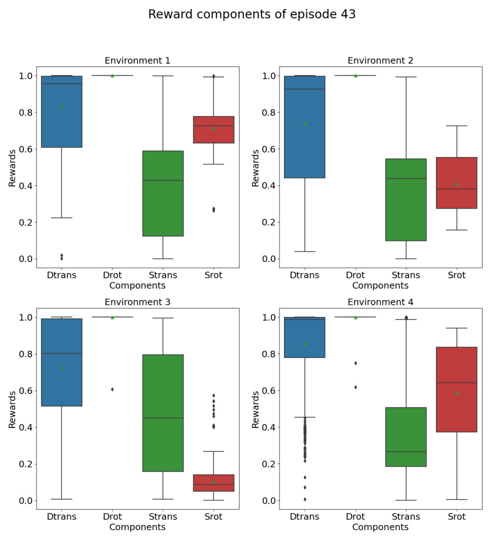
Boxplot of rewards over actions



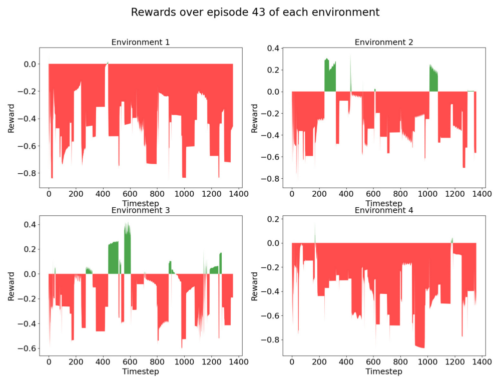
Boxplot of reward components over whole training



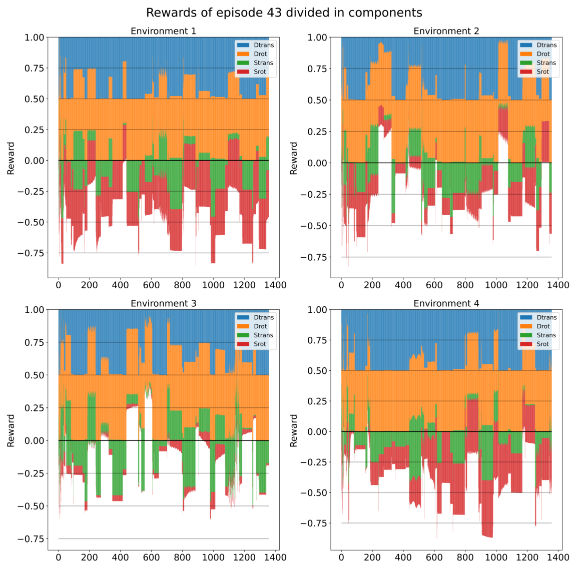
Mean reward of each episode



Boxplot of reward components of last episode



Rewards over last episode



Rewards over last episode divided in components