Assessment of Reinforcement Learning for CubeSat concept generation

B. Krijnen

Master of Science Thesis 2020

DELFT UNIVERSITY OF TECHNOLOGY

FACULTY OF AEROSPACE ENGINEERING

Assessment of Reinforcement Learning for CubeSat concept generation

by

B. Krijnen

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Thursday December 17, 2020 at 09:30 AM.

Student number:	4232143	
Project duration:	November, 2019 – December, 2020	
Thesis committee:	dr. A. Cervone,	TU Delft, Committee chair
	dr. J. Guo,	TU Delft, Daily supervisor
	ir. M.C. Naeije,	TU Delft, External examiner

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Summary

Advances in miniaturised components and electronics has driven the miniaturisation of satellites, the products of which are being seen as capable replacements or additions for missions that have traditionally been performed by large satellites. The miniaturisation of satellites has also enabled relatively small companies and institutions to launch their own space systems. Delft University of Technology (DUT) has been able to set up the Delfi program using the CubeSat standard. The CubeSat standard is a form factor standard that is seeing widespread use for the development of nanosatellites. An increasing number of commercial-off-the-shelf (COTS) components for CubeSats offers an increasingly large design space for CubeSat missions. It is becoming increasingly difficult to evaluate this entire design space manually.

An automated design tool with the ability to take the entire design space into account may improve the design of a CubeSat compared to manual methods. This work sets out to develop such a design tool. Through a trade-off, it is decided to implement a machine learning methodology into the design tool for the generation of CubeSat concepts. Reinforcement learning (RL) is identified as the suitable machine learning method. More specifically, the Q-learning algorithm is selected as the RL algorithm that is implemented into the design tool.

The design tool that is created in this work is able to generate CubeSat concepts by selecting components from a manually created hypothetical components database. Components can be selected by the design tool for the following component types: camera, ground station, antenna, transceiver, attitude determination, attitude control, power control unit and battery. These components make up the CubeSat concept. A training cycle of the design tool consists of 25000 episodes with each episode containing 100 iterations. At each episode, the design tool is initiated with a random configuration of components. This configuration of components is the state $[C_1, C_2, ..., C_n]$, with each index representing a component and the length of which depends on the number of component types that are implemented into the design tool. The state thereby represents a CubeSat concept, insofar that the concept exists of the component types that are available to the agent in the environment. At each iteration, the design tool's agent can take an action that changes the selected components. Whether this action is taken randomly or based on previous knowledge stored in the Q-table is based on the ϵ-greedy method, which determines whether the agent explores the design space or exploits it. At every iteration, a reward is provided to the concept based on a performance analysis of the selected components. When the design tool is exploiting the design space, it will aim to maximise the reward that it can obtain and will thereby aim to select the optimal components. Through experiments, several ways of implementing the components into the design cycle are evaluated. The method that is implemented is to let the design tool select components directly from the database.

To investigate the influence of an increase in complexity on the performance of the design tool, component type(s) are gradually added until the full setup of the design tool is achieved. The results of this investigation show that the design tool becomes less likely to select higher rewarding (and therefore performing) components during a training cycle as the complexity increases. The design tool's capability of assessing the performance of a complete CubeSat concept is investigated by examining whether the selection of components influences the analyses correctly. It is shown that the CubeSat parameters that are dependent on the complete component set such as size and power consumption are correctly implemented.

A validation of the design tool is aimed to be performed using a case study, which is the AltiCube mission study. The AltiCube mission study is a feasibility study of a Ka-band altimeter CubeSat constellation for ocean monitoring, carried out by the DUT, specifically by the Civil and Aerospace Engineering faculties, under the support of the European Space Agency. The validation is performed by means of comparison between the systems designs from the mission study itself and the design selected by the design tool for the case study's mission requirements. The components that are selected by the design

tool provide a feasible concept for the mission requirements and are generally similar to the result of the mission study itself. The difference in size and mass of the designs can be explained by the difference in the way that the size and mass are estimated by the design tool. It is found that the design tool selects certain components based on marginal differences in the concept rewards, and that the components that a trained agent selects can be components that are over-performing for the mission requirements. Using a trained agent to identify feasible concepts can result in the design tool selecting over-performing concepts. To improve the quality of the trained agent, additional system performance rewards should be implemented that provide a penalty to over-performing concepts, thereby giving the design tool the capability of selecting the optimal concept for the mission requirements. Examples of such rewards are mass and cost. The implementation of a weighted reward system is identified as a possible improvement to the design tool for enabling the user to emphasise certain performance aspects. The way in which the design tool is set up allows for a simple implementation of such additions. This also includes the addition of extra component types.

The results from a training cycle are at the moment better suitable for identifying the optimal concept, because this cycle also explores the design space and thereby does not leave out certain solutions. The concepts the design tool provides are made up of COTS components obtained from a COTS components database that is used directly in the design cycle. The usability of the design tool in its current version is limited due to the tabular nature of the Q-learning method. The tabular nature means that the method has a high computational expense in terms of RAM, which limits the number of component types and components per component type that can be implemented. Note that if a supercomputer would be used this would not be an issue. The other limitation is that a tabular method is not able to generalise over unseen states because discrete values are stored for state-action combinations. The method is not able to generalise from surrounding states to predict the action to take at the unseen state. In order to generate a fully trained agent, most, if not all state-action combinations should be visited, which results in a high sampling requirement. This limitation also inhibits the ability of a trained agent to be re-used for varying mission requirements, which would be a very valuable aspect of the design tool. A promising method that does not suffer from these limitations is the Deep Q-Network, which can be seen as an extension of the Q-learning method.

Acknowledgements

Throughout the duration of this work, a number of people have offered me support in various ways. Support that has been directly and indirectly valuable for this work. I would like to acknowledge these people here.
To start with, I would like to thank my direct supervisor, Associate Professor Dr. Jian Guo. After an internship, I came back to Delft with an idea in my head for the thesis, which was the idea to explore ways in which to automate the design of a CubeSat. Jian allowed me to explore this idea freely, while maintaining a critical overview over this work which kept my work within feasible boundaries for the amount of time that this work is allowed to take up. Next to that, during the thesis he invited me to participate in the system design for the ESA project AltiCube, which eventually would be used in this work as well. This intermezzo has given me additional practical experience and drove me to become more goal-oriented in this work. For all of this, I want to thank Jian.
In this work, I have selected to pursue a machine learning methodology. This field is a novel field for me, and the assistance and guidance of Dr. ir. Erik-Jan van Kampen at the section of Control & Simulations at Delft University of Technology has helped me significantly with the implementation of reinforcement learning in this work. I had very interesting discussions with the ISISpace System Engineer Eric Bertels, during which the practical side of automating the design of a CubeSat was discussed. These discussions provided a user point-of-view to this work, which was very helpful.
I would also like to mention several of my fellow students with who I have had useful discussions during the compilation of this work. These discussions provided valuable insights into different aspects of this work. I would like to thank Angelos Karagiannis, Jesse Hummel, Alexander Drent, Stijn Koehler, Peter van den Berg and Wessel den Ouden.

To conclude with, on a more personal note, I want to express my love to Sophie, with whom I have had the pleasure to spend more and more time with during the course of this work. You have become an immense source of joy during what have sometimes been trying times. And, I want to express my everlasting gratitude to my family: my parents Gerard and Rian, and my brother Rob and his partner Jeanine, who have supported me throughout my studies. Without their support, I would never have been able to enjoy my time in Delft such as I have.

Bas Krijnen
Delft, the Netherlands
November 2020

Contents

List of Abbreviations

Abbreviation	Description
AC	Attitude Control
AD	Attitude Determination
$ADCS$	Attitude Determination & Control System
AI	Artificial Intelligence
BOL	Beginning of Life
$CCG-Q$	CubeSat Concept Generation using Q-learing
CDF	Concurrent Design Facility
CDH	Command & Data Handling
CE	Concurrent Engineering
CEF	Concurrent Engineering Facility
$CNES$	French Space Agency
$COTS$	Commercial Off the Shelf
DEA	Design Engineering Assistant
DoD	Depth of Discharge
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DSP	Deployable Solar Panels
DUT	Delft University of Technology
EO	Earth Observation
EOL	End of Life
EPS	Electrical Power System
ESA	European Space Agency
$MBSE$	Model-Based Systems Engineering
MOO	Multi-Objective Optimisation
$MORL$	Multi-Objective Reinforcement Learning
$NASA$	National Aeronautics and Space Administration
PAY	Payload
PCU	Power Control Unit
RL	Reinforcement Learning
STK	Systems Tool Kit
TD	Temporal Difference
TTC	Telemetry, Tracking & Command

List of Symbols

Latin characters	Description	Unit
a	Semi-major axis	m
A	Action	–
Δa_{rev}	Semi-major axis decay per revolution	m
A	Surface Area	$\mathrm{m^2}$
B	Magnetic moment Earth	$\mathrm{Tm^3}$
c	Speed of light	m/s
cp	Center of pressure	m
cm	Center of mass	m
C_D	Drag coefficient	–
d_{px}	Width of a pixel	m
D	Solar cell degradation	$-/$yr
D	Magnetic dipole moment	$\mathrm{Am^2}$
e	Eccentricity	–
E	Elevation angle	°
E	Eclipse Fraction	–
$\frac{E_b}{N_0}$	Ratio of received energy per bit to noise density	dB
f	Focal length	m
f	Frequency	Hz
F	Noise figure	–
G	Gain	dB
G	Gravitational constant	$\mathrm{m^3kg^{-1}s^{-2}}$
GSD	Ground sample distance	m
h	Angular momentum	Nms
h	Orbit height	m
H	Orbit height	m
i	Inclination	°
I	Moment of inertia	$\mathrm{kgm^2}$
k	Boltzmann's constant	$\mathrm{m^2kgs^{-2}K^{-1}}$
L	Loss	dB
L	Lifetime	Yr
LM	Link Margin	dB
m	Mass	kg
M	Mass Earth	kg
M	Earth Magnetic Moment	$\mathrm{Tm^2}$
n_{px}	Number of pixels	–
P	Orbit period	s
P	Power	W
q	Reflectance factor	–
$Q(S,A)$ Q-table	–	
r	Orbit radius	m
r_e	Radius Earth	m
r_{eq}	Radius Earth's equator	m
r_{slant}	Distance between transmitter and receiver	m
R	Data rate	bps
R	Reward	–
R_E	Radius Earth	m
S	State	–
S	Swadth width	m
t	Time	s
T	Time in view	s

T	Torque	Nm
T	Temperature	K
U	Unit	–
V	Velocity	m/s
X	Efficiency of power delivery to components	–

Greek characters	**Description**	**Unit**
β	Beta angle	degrees
η	Nadir angle	degrees
η	Solar cell efficiency	–
ϵ	Pointing error	degrees
ε	Elevation angle	degrees
ε	Obliquity of the ecliptic of the Earth	degrees
Φ	Solar constant	$\mathrm{W/m^2}$
Γ	Ecliptic true solar longitude	degrees
Ω	RAAN	degrees
ρ	Angular radius of the Earth	degrees
ρ	Density	$\mathrm{kg/m^3}$
θ	Arc angle	degrees
θ	Half power beamwidth	degrees
θ	Incidence angle	degrees
θ_{LVZP}	Angle between the local vertical and the Z principle axis	degrees
θ	Slew angle	degrees
$\dot{\Omega}$	RAAN precession	degrees/s
μ	Standard gravitational parameter	$\mathrm{m^3s^{-2}}$
λ	Effective horizon angle	degrees
λ	Magnetic latitude function	–
λ	Wavelength	m
π	Number of Pi	–

List of Figures

List of Tables

1 Introduction

1.1. Introduction to the work

The growing need for CubeSats for various space missions could present strong demands for the use of automated systems during the early stage of the design cycle. An ever-increasing catalogue of COTS components for CubeSats creates an increasing amount of design options for the generation of concepts. Manual design methods make only limited use of this available design space because they rely on the engineer's input for design choices and are therefore constrained by the engineer's knowledge and experience. Automated systems that are able to incorporate the entire design space offered by an ever-increasing catalogue of COTS components may potentially improve the design of a CubeSat, compared to manual methods.

Artificial Intelligence (AI) could provide a solution for automated systems for the generation of concepts based on the entire catalogue of COTS components. In this work, the feasibility of a design tool that makes use of RL for automated CubeSat concept generation is presented. The design tool that is created is able to generate CubeSat concepts using a RL algorithm which is able to select subsystem components from a hypothetical hardware database, thereby generating a complete CubeSat concept. The final performance of generated concepts is evaluated using a reward system that provides rewards based on subsystem and system performance against user entered requirements, thereby keeping the engineer in the loop. The appeal of using RL to identify feasible concepts for user requirements is that knowledge from earlier runs can be re-used to provide faster results for new missions.

This assessment is structured as follows. In chapter 2, the reasoning for selecting a machine learning methodology is performed. Part of the reasoning is a trade-off between several methodologies. A review of machine learning based design methods is also performed and the scope of the work is presented, stating the research objective and questions. chapter 3 goes into depth on the implementation of RL into the design tool that is created in this work. Experiments are performed on a low complexity version of the design tool in order to study the design tool's settings in chapter 4. The results of these experiments are also shown and discussed in this chapter. Next, the final setup of the design tool is presented in chapter 5. Validation of the design tool is done through a case study, which is elaborated upon in chapter 6. The results of the design tool are presented through three investigations in chapter 7, where these results are also discussed. The three investigations are the following. First the design tool is increased gradually in complexity ending up with the final setup, where the effect of this increase in complexity on the performance of the design tool is studied. The next investigation is into the performance of the final setup of the design tool. Finally, the design tool is applied to the case study and a comparison between the two methods is performed. Conclusions for the work that is performed are drawn and recommendations for future work are made in chapter 8 and chapter 9, respectively.

1.2. General context

The complexity of space systems design has historically been tackled with the use of extensive systems engineering and management methods. Relatively recent in the overall history of space systems design, the combination of concurrent engineering and systems engineering is being applied to the design approach with the goal of shortening the design time of space systems. These design approaches are being applied in concurrent design facilities (CDF) all over the world, with DUT also being in the process of setting up such a facility.

Concurrently, in the last decades, the advancement of miniaturised components and electronics has driven the miniaturisation of satellites. This development is aided by the increase in COTS components from the consumer electronic market. Nano- and microsatellites are more and more capable of replacing missions traditionally performed by large satellites or complementing such missions. The miniaturisation development has also enabled relatively small companies and institutions to launch their own space systems due to the relatively low costs. The CubeSat standard is a form factor standard that is seeing widespread use for the development of nanosatellites. First used primarily for educational purposes, the use of the CubeSat standard for space systems became rapidly popular within governmental and private instancies as well [54]. CubeSats are seen as capable platforms for carrying out scientific, observation and technology demonstration missions at significantly reduced costs [43].

DUT is also making use of the CubeSat standard for their own projects, primarily being the Delfi program. Education within the space field is geared towards nano- and microsatellites evidenced by courses such as Microsat Engineering and Micropropulsion at the Space Engineering department. The department is currently in the process of setting up a CDF. This work could contribute towards that process.

The growing need for CubeSats has lead to an increasing amount of COTS components. This is demonstrated by the number of components that is available on online marketplaces such as satsearch[1] and CubeSatshop, which still do not encompass the full number of components that exist on the market. The design space that is presented by the catalogue of COTS components is becoming increasingly difficult to evaluate manually simply because of the sheer number of components. This incomplete exploration of the design space is also stressed by [43]. If it is assumed that the satellite configuration consists of 6 distinct subsystems for each of which a single component should be selected, and 10 components are available for each subsystem, the number of design options is already 1 million. It can be seen that even with a relatively low amount of component options the number of design options grows rapidly. Being able to take into account the entire design space may potentially improve the design of a CubeSat compared to manual methods.

Reducing the cost of low cost space missions such as nano- and microsatellite missions is commonly aimed for by minimising the amount of work that have to be performed during the early phases of a project (Pre-Phase A and Phase A) [9]. An important aspect of this is using existing hardware with known specifications. These known specifications, coupled with a standard platform allows for much more accurately assessment of the performance at system level. The use of COTS components directly in the design cycle is stressed by literature, among others by Surrey Space Center & Surrey Satellite Technology Ltd [46], by the CubeSat focused company GOMspace [4], by Tyvak Nanosatellite Systems [3], and by the French Space Agency (CNES) [40]. The development of automating at least parts of the design processes of a space system by various software tools is detailed by among others Valispace [27], by CNES [23] [12], and by satsearch [21]. The software tools that are reviewed show a similar shift in moving from a document-centric approach to a data-centric approach. These software tools aim to increase the efficiency of current space system design method but seem to still be majorly people based design tools where the design choices are still performed manually. The growing need for CubeSats for various space missions, coupled with the desire to take into account the entire design space offered by the catalogue of COTS components, could present strong demands for the use of fully automated systems during the early stage of the design cycle[2].

[1] satsearch currently already lists 57 distinct products for cameras alone[2]

[2] During talks with an ISISpace engineer interest was also shown for automated tools that could assess the sensitivity of a CubeSat design for certain parameters

Such an automated system would be able to select components from the catalogue of COTS components and analyse the performance of such a configuration of components. The performance for certain user requirements will allow the automated system to identify feasible configurations. This work focuses on first identifying what a novel approach would be to design such an automated system and afterwards perform a feasibility study with said approach. In other words, in this work it is attempted to create a design tool that enables an automated CubeSat concept generation.

1.3. Scope of the work

Literature has indicated the shift of industry towards the use of automated design methods for the design of space systems. The automated design tools that are currently in development however, do not provide the complete automation of the design cycle. Therefore, there is a need for a design tool that can provide such a capability. An important factor for the design of nano- and microsatellites is the use of COTS components and a standard platform. A widely used standardised platform is the CubeSat standard, which is also actively being used by DUT. The focus of this work is on this type of satellite. One of the fundamental aspects of the design tool is deemed to be the capability of using COTS components directly in the design cycle. In this work, a design tool that is able to automate the design of a CubeSat concept is created. The method that is used for the generation of concepts is a RL method, the reasoning for which will be further elaborated upon in chapter 2. The scope of this work is framed by formulating a research objective based on the needs identified in this chapter. It is attempted to realise this objective by answering the research questions that are stated below.

RESEARCH OBJECTIVE:

assessing a design tool for automated CubeSat concept generation using RL and COTS components implemented directly in the design cycle for the preliminary design of a CubeSat mission, with the aim of exploring the complete design space offered by the COTS components catalogue

RESEARCH QUESTIONS:

- RQ1: What RL method can be used for automated CubeSat concept generation?
- RQ2: How can COTS components be implemented directly in the design tool?
- RQ3: Is the design tool able to identify feasible CubeSat concepts for a set of mission requirements?
- RQ4: Is the design tool capable of re-using data from previous runs?

2

CubeSat Concept Generation Methodology

2.1. Concept Generation Methodology Trade-Off

The methodology of the user-driven cubesat concept generation software consists of two parts. Firstly, there is the question on how a single concept is generated, this means how the different subsystem models in the software interact and thereby influence the components' feasibility.

Secondly, there is the question on how a complete set of concepts is generated. This question deals with how a set of concepts is generated from a set of components. It should be decided as to what the set of concepts actually entails (e.g. do we want the full design space or just the 'feasible' solutions), and in what manner and for what reason components could be included or excluded from the concept generation. It could also be said that the first aspect of the methodology deals with the procedure of the concept generation, while the second question is about what kind of in- and output we need and want to generate with the software.

The design space is made up of all possible concepts created by these components. However, the number of concepts becomes increasingly large with an increasing number of component options. This is because there are multiple points during the design process where component selection for a specific subsystem occurs. Component selection occurs a total of 6 times during the design process. Assuming every subsystem has the same amount of component options, the amount of concepts this results in is given by Equation 2.1. This means that if there are 10 available components for each subsystem to choose from, numbering a total of 60 components for the complete design, this would already result in a million possible concepts. The question then is whether this is an issue for the software, and if it is actually beneficial for the software to evaluate every possible concept.

$$n_{concepts} = n_{components}^{6} \tag{2.1}$$

In theory, the amount of concepts should not pose a problem for the software with regards to the capability to evaluate the concepts. The amount of concepts does however, influence the time it takes to complete the design cycle, since a larger amount of concepts means a larger amount of evaluations have to take place. The amount of data can also slow down the speed and efficiency of the evaluations.

2.1.1. Methodology Options

The goal of the design tool is to provide the user feasible concepts. In order to identify what a feasible concept is, the tool will have to analyse the performance of a configuration of components for a set of mission requirements. The problem can then essentially be identified as a design optimisation problem that can be solved using an optimisation scheme. In this section, several methodologies that can provide the capability of identifying feasible concepts will be reviewed.

Top-Down or Bottom-Up Process
A basic choice that can be made with respect to the methodology for exploring the design space and generating concept sets is whether this process should be top-down or bottom-up. For this application, a top-down method is said to be a method where concepts are eliminated early on the design process, thereby reducing the design space and eliminating subsequent options further down in the design process as well. Conversely, a bottom-up process explores and evaluates the full design space, thereby evaluating by means of sampling the entire design space and finding the attractive design space in this way.

The desired process for this application is thought to be a bottom-up process, since this will not eliminate any concepts early on the design process, potentially losing promising concepts. A traditional design method based on requirements and constraints can be said to be a top-down process. It is still reviewed in this thesis, due too the design method being a traditional methodology, which will provide a useful baseline for the eventual trade-off.

Objective Function
To evaluate a concept, it has to be known what to evaluate for. This is where the system requirements come into play. A concept's performance can be evaluated to the requirements numerically and given an objective function evaluation.

Requirement & Constraint Based Methodology
Reducing the size of the design space using a requirements and constraints based method will reduce the amount of concepts that could be further evaluated. This does not however, contain an approach to then evaluate the reduced design space and the way in which feasible designs are found.

In order for the requirement and constraint based reduction to function, the performance of subsystem components will have to be evaluated against the performance requirements for that subsystem. In this way it is possible to eliminate components that do not comply to the requirements. This means that the design space is reduced at each subsystem design phase. Nonetheless, for every subsystem every possible component has to evaluated, taking into account the previous subsystem's feasible components. What is important to note here, is the fact that the feasibility of subsequent subsystem components is dependent on the previous combination of components. The specific combinations that lead to the elimination of other components has to be stored, in order to evaluate the full concepts further on in the design process. Whether that is achievable with an average system for the amount of data that is created is an important consideration for this method. The feasible concept set for the subsystem requirements does automatically result from this method. The system performance can then be evaluated for these feasible concepts.

If the amount of feasible concepts is still significantly large, depending on the amount of calculations that are necessary to evaluate the system performance, evaluating this for every concept remains impractical. In that case, other (design space reduction) methods should be used to narrow down the design space even further or to explore the remaining design space efficiently.

Fuzzy Clustering Based Methodology
Fuzzy c-means clustering method (FCM) can be used to identify optimal clusters. The components themselves are not evaluated for their subsystem performance. Instead, the clusters that are found to be feasible are used to identify the feasible components that are part of these clusters. In this way, the concepts that should be evaluated for their system performance can be discovered.

The size of the clusters determines how many concepts are feasible concepts for the mission. Similarly to the requirement and constraint based method, this design space could still be significantly large, and therefore the same strategy is valid for this method. For this specific method, the reduced design space could be divided into clusters again, iterating through the same process until a manageable sized concept set is obtained.

Meta-Model Based Methodology
A meta-model approach can be viewed as an optimisation-scheme and is aimed to improve the efficiency of the optimisation process by iteratively reducing the size of the design space. This is stated

by [8], [48], [47], [25] and [52]. In short the iteration loop of the method works as follows to find the global optimum. First, the design space is sampled using a specific sampling method, after which these samples are evaluated with increasingly a model of the engineering problem. Using the sample evaluations, meta-models are constructed on which a design space reduction method is used to find the attractive design space. The process then starts another iteration until the global optimum is identified with a high certainty. Increasingly higher-fidelity models are used in increasingly smaller design spaces in order to efficiently approach the global optimum. This method does involve creating the different fidelity models.

To explore and evaluate the design space several methods are used in the literature. Sampling and employing various optimisation algorithms are the most common methods found in the literature. After evaluating the design space, a method is used to perform the design space reduction. A common method is to use a clustering based reduction method, as described by, among others, [47], [25] and [52].

It can be noted that the meta-model based concept set generation methodology combines different methods in order to find the global optimum. The meta-model is combined with a design space reduction method in order to find the global optimum efficiently. The clustering method for the purpose of design space reduction to provide a certain amount of 'best' solutions around the global optimum, and thereby providing the desired concept set.

Machine Learning Based Methodology

A method that could also solve the complex problem of finding the best design(s) is the use of machine learning. Machine learning methods learn from data in such a way that the method is able to solve future tasks. The machine learning methods are generally divided into three categories; supervised, unsupervised and RL. The method depends on the kind of data that is available [28]. Where examples of data with both the inputs and outputs are available a supervised learning method is applicable. Unsupervised learning is applicable when only the inputs of the example data is available. Finally, RL is applicable when the correct output is not available, but the measure of quality of a certain output resulting from a certain input is obtainable.

For this thesis, the output can be obtained by evaluating a certain concept input for the mission requirements. However, due to the large amount of possible concepts, sampling the design space to provide a useful learning data set might prove to be cumbersome. Because of this, supervised learning might not be the best machine learning method for this application. Next, unsupervised learning aims to find patterns in the input data. Since this will not help us to achieve the goal of finding the best feasible concept(s), this machine learning method is not suited for this application. The method which seems to be the most promising approach for this kind of application is the RL method. This approach is able to independently generate and learn from input and its corresponding output. It is therefore not necessary to sample the design space and generate a learning data set manually.

In more detail, RL is the learning of a mapping from states to actions with the goal of maximising the reward through these actions [44]. The actions to perform are not specified to the learning algorithm, in contrast to the other machine learning methods. Instead, the algorithm learns which action to take by testing them and evaluating the resulting reward. How far the learning algorithm should look into the future for the accumulative reward of subsequent actions depends on the application and can be varied. The actions in this approach would translate to the action of choosing a component for a concept when the RL approach is applied to this thesis' application.

The main approaches for solving RL problems are divided into two by [18]. The first approach is *"to search in the space of behaviours in order to find one that performs well in the environment"* [18, pg. 1], which is the approach used in genetic algorithms and programming. The second approach is to use statistical techniques and dynamic programming methods to approximate the reward of taking an action in certain situations. After this basic definition of RL, the applicability of RL for engineering design, and more specifically for this thesis is reviewed.

Today, the use of artificial intelligence (AI) and machine learning methods for engineering is becoming more and more widespread. Using machine learning for design optimisation problems can be seen across industries and applications. This is evidenced by the current available literature, among others:

[35], [10], [29], [7]. More specifically using RL by [26], [53], and learning optimisation algorithms itself using RL by [24]. This literature is used to demonstrate the relevance of machine learning and more specifically reinforcement for (design) optimisation problems. The similarities between multi-objective optimisation (MOO) and multi-objective reinforcement learning (MORL) and other RL methods are described by [30], as well as common strategies for both methods.

Furthermore, the use of AI to aid complex design problems is described by [20]. The use of AI for space applications is outlined by [14], and proposed specifically for the design of space systems by [41], [5], [32] and [36]. It seems that there is a big gap between the first use of AI in space systems design and the more recent activities. While [36] specifies the use of specific AI methods, [32] outlines the use of AI through various stages of the early design of space missions, but does not specify the AI methods that will be used. Moreover, the paper indicates that the use of AI is targeted towards a support role for the engineers. This means that the actual engineering design is still left to the engineers.

The interesting prospect of using machine learning over other optimisation schemes is that the knowledge that is gained during runs can potentially be re-used for new runs. Using a machine learning method, an agent is trained during the use of the method. This means that the knowledge gained during such runs is not discarded and can be re-used to more efficiently use the method during new runs. This also means that when the method is not actively used for any problem, it can be trained to perform these functions faster when it is required.

To conclude, literature demonstrates the usability of machine learning for (design) optimisation problems. RL shows the most promise for this thesis' application due too the decision processes that are present in this type of machine learning. Since the concept set generation problem that is treated in this thesis can also be seen as a multi-objective complex problem, the applicability of RL for this problem could be justified. However, literature does not indicate the use of machine learning, and more specifically RL for the selection of hardware components, which is integrated into a design tool. This application is not prevalent in either space systems design or any other industry. The literature does indicate that the use of machine learning in an engineering optimisation problem is effective, and can achieve better results than traditional approaches, as described by [53], and [10]. It seems that applying this methodology for the kind of application that is required for this thesis is uncharted territory, at least as far as literature shows.

2.1.2. Methodology Trade-Off

In the previous section, several methodology options have been identified. In this section, a trade-off of these options will be performed in order to select one of them for implementation into the design tool. The trade-off methods that will be used are taken from [13]. The methods that will be used are the Pugh matrix and a graphical trade-off method, as it is thought these two methods will supplement each other.

- Applicability: This is the question whether the method is suitable for the kind of application that it should be used for during the thesis, and also includes the ease with which the method can be implemented.
- Efficiency: Answering this should give an indication as to how efficient the method is for this kind of application.
- Completeness: The question of whether this approach is a one off method. Is it necessary to combine the approach with other methods to be able to perform the concept set generation efficiently.
- Transparency: This criterion handles the fact whether it is difficult for the user of the design tool to gain insights into the design process. Some design methods are very much black-box processes while other allow for a transparent process.
- Novelty: Is applying the method in this way during the thesis a novel approach, and how is it a novel approach.
- Bottom-up/top-down: This criterion deals with the question of whether the methodology can be viewed as a bottom-up or top-down process.

Pugh Matrix

The first trade-off method that is used is the Pugh matrix. The result can be seen in Table 2.1. In this matrix the weighting factors range from 1 to 3, with 1 being the least important and 3 the most important. The scoring ranges from -1 to 1, with -1 having a negative connection with the selection criterion, 0 a neutral connection, and 1 a positive connection. In this trade-off, the machine learning methodology has the best score and would therefore be the most promising method.

Table 2.1: Pugh matrix for the concept set generation methodology

Selection Criteria	Weighting Factor	Requirements/Constraint	Fuzzy Clustering	Meta-Model	Machine Learning
Applicability	2	0	1	0	1
Efficiency	3	-1	0	1	1
Completeness	1	-1	0	1	1
Transparency	2	1	1	0	-1
Novelty	3	-1	-1	0	1
Bottom-Up/Top-Down	2	-1	1	1	1
	Sum(+)	1	3	3	5
	Sum(0)	1	2	3	0
	Sum(-)	4	1	0	1
	Result	-7	3	6	9

A sensitivity analysis is also done in order to see the effect of changing weights and scores to this result. The sensitivity analysis of the Pugh matrix for 3 different scoring entries can be seen in Figure 2.1. Next to that, a sensitivity analysis is performed by changing the weighting factors and keeping the scores the same as in Table 2.1. 4 different weighting factors entries are used, among one where all the weights are set equal to 1. The result of this can be seen in Figure 2.2.

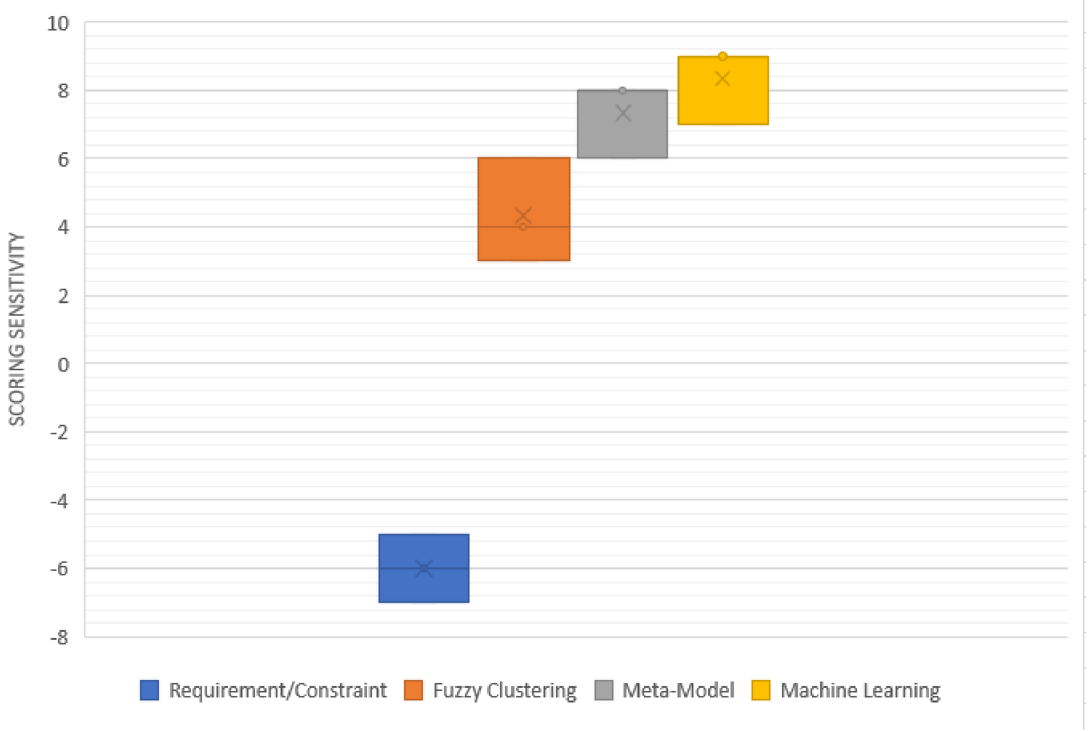


Figure 2.1: Sensitivity analysis of Pugh matrix for 3 different scoring entries

It can be seen in Figure 2.1 that the meta-model and machine learning scores are relatively close. A larger difference between the two can be observed in Figure 2.2, where the scores for machine learning have a larger spread. In both sensitivity analyses the machine learning method has the best average performance. It is therefore concluded that the best option to pursue based on the Pugh matrix trade-off is the machine learning option.

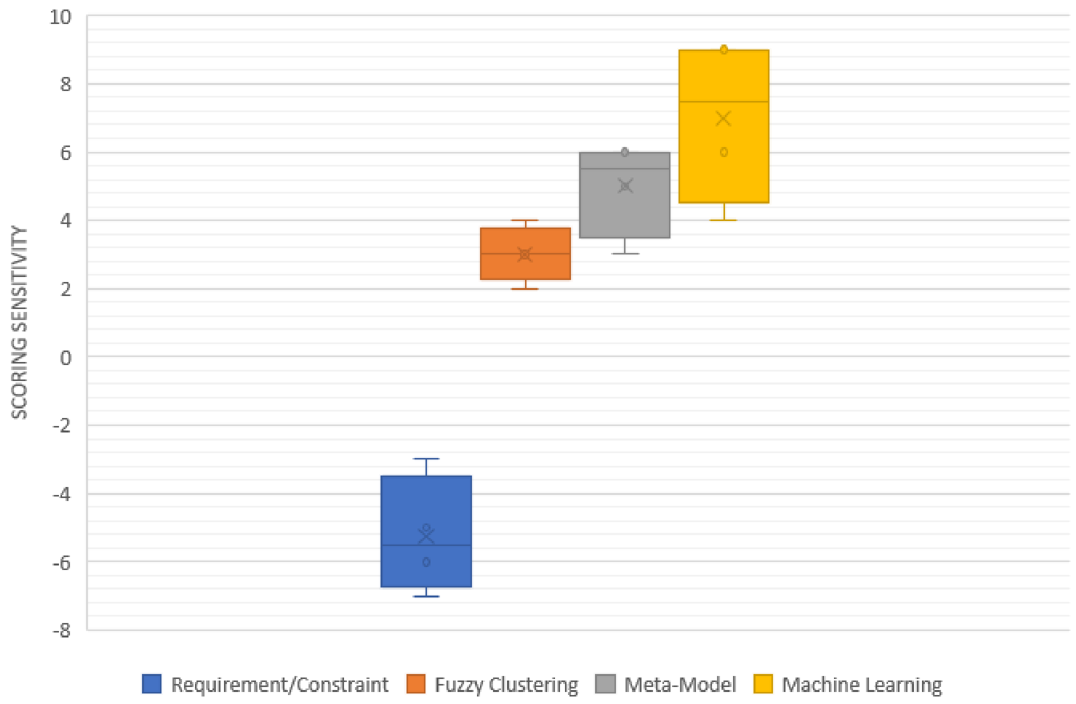


Figure 2.2: Sensitivity analysis of Pugh matrix for 4 different weighting factor entries

Graphical Trade-Off
The second trade-off method is the graphical trade-off method. The scoring is performed in a simple manner, namely scoring by assigning a colour. The colouring scheme that is can be seen in the legend underneath the table in Table 2.2. In this table, the graphical trade-off end result can be seen. The width of the columns, in other words of the selection criteria, indicates the weight of the criterion visually. Combined with the colouring scheme, the scoring is indicated visually. From Table 2.2, the graphical trade-off also indicates that the machine learning method is most suited for this application. Both the meta-model and machine learning methods score a 'correctable deficiencies' for the transparency criterion. This criterion was found to be difficult to score for these methods, as there is a difference in transparency between the two. It is thought that the meta-model method allows for a more transparent approach as apposed to the machine learning method, and is therefore more easily correctable. However, since it is thought that at their basic level the two method are black-box processes, the methods are given the same colour and score. An important factor that weighs into this conclusion, is the fact that novelty plays an important role in the trade-off.

2.2. Review of Machine Learning Based Design Methods

In subsection 2.1.2, a trade-off has been performed on different methods that could be used for the generation of concepts for the design tool. From the trade-off, it resulted that machine learning is the most promising approach for this application. Moreover, in subsection 2.1.1, it was identified that of the different machine learning approaches, RL is seen as the most applicable approach for the type of design process that is envisioned in this thesis. This is because RL deals with a decision process, which the selection of components during the design process can also be seen as. In this section, the implementation of such a RL based approach is investigated.

An integral part of the design tool is the ability to provide a solution set to the user, in order to allow the user the ultimate decision of picking the 'best' solution. Because in all learning problems there is some parameter to minimise or maximise, one can consider all learning problems as optimisation problems [17]. The review into the implementation of a RL based approach will therefore be performed in two directions. A more general RL approach to generate a concept set is reviewed. Next to that, the use of RL based optimisation approach is reviewed. Since the problem at hand can be seen as a

Table 2.2: Graphical trade-off matrix for the concept set generation methodology

	Applicability	Efficiency	Completeness	Transparency	Novelty	Bottom-Up / Top-Down
Requirement/ Constraint	green proven design process	yellow cumbersome process	yellow will need additional methods	green transparent process	red traditional method	red top-down
Fuzzy Clustering	blue applicable	yellow cumbersome process	yellow might need additional methods	green transparent process	yellow proven method	green bottom-up
Meta-Model	blue applicable, component selection not encountered	blue efficient process	green provides end-solution	yellow black-box process	blue recent method	green bottom-up
Machine Learning	blue applicable, component selection not encountered	green very efficient after training	green provides end-solution	yellow black-box process	green new method	green bottom-up

green	excellent, exceeds requirements	yellow	Correctable deficiencies
blue	good, meets requirements	red	Unacceptable

multi-objective problem, this approach should incorporate MOO. The MOO approach using RL is found in literature to be called MORL, which is reviewed as the second direction. First off however, a review is performed on current implementation of artificial intelligence for the fast design of satellites. This is done to provide the reader with an overview of related developments in this works field.

2.2.1. Use of Artificial Intelligence for Fast Design of Satellites

A tool for automation of satellite design and creating highly accurate design concepts in a fast and cheap way is introduced by [16]. The design tool, called SPIDR, is developed to automate the evaluation of ripple effects during the design process, without having to have user input. Ripple effects are caused by changing parameters during the design process which result in re-evaluations of the design having to occur. Automating this is aimed to allow faster evaluations of designs, which would allow a more expansive exploration of the design space and thereby finding higher quality concepts.

The use of AI in this design tool is detailed by [19]. In general, it is understood that the design process depends on the use of rules. These rules set local constraints between components in the system, and thereby signify the connections between these components. An example of such a rule is that when a torque is included, by definition a magnetometer should be included as well. The design process is then seen as a planning problem, where a goal condition is met through a feasible sequence of actions. The use of rules allows the tool to generate concepts and evaluate these concepts, thereby eliminating infeasible concepts. This process allows for a large design space exploration, and combined with optimisation methods the exploration can be focused.

Another tool for rapid design of satellites is SATBuilder, which is described by [41]. SATBuilder, using the open source OpenSAT architecture, has been developed in response to the Responsive Space Program, and is therefore aimed at enveloping the entire satellite design process. The use of AI in this process is found for the generation of feasible design solutions. SATBuilder uses the Bayesian Artificial Intelligence approach for this purpose. This approach uses user input and historical data, in the form of established satellite designs, to determine feasible design solutions to new user requirements. The Bayesian network is first trained for these inputs, after which it is used in the SATBuilder tool.

A design engineering assistant (DEA), which is an AI-based agent, is introduced by [32]. The aim of the DEA is to assist experts with knowledge management support in feasibility studies during concurrent engineering sessions. This support can come in the form of automatically obtaining knowledge from previous designs that are based on similar requirements to the current mission. An expert system is able to perform this function more efficiently than when it would be done manually.

Under the DEA project, the use of a unsupervised machine learning methodology for analysing documents is outlined by [6]. The analysis consists of identifying, learning and extracting topics from a set of documents using topic modelling, which is an unsupervised learning method. This will enable an expert to more easily kick start a study, since relevant information for the mission is more efficiently found, allowing the engineer to more accurately estimate value ranges and validated architectures.

The DEA, as outlined by [32] and [6], is as the name implies an engineering assistant. The actual design process and evaluation itself, is still performed manually by the engineer. The DEA plays a supporting role in this design process, and delivers the engineer better insights into relevant information for the mission. The DEA can therefore not be seen as a design tool, in contrast to SPIDR and SAT-Builder. Even though these examples of the use of AI for satellite design should not be seen as fully encompassing this field, it is interesting to see the gap in time between the design tools SPIDR/SAT-Builder and the more recent approach by [32], which also applies AI in a different way than the design tools.

The literature study that was performed in the relevant field for this thesis did not discover the use of machine learning in a design tool to generate satellite concepts. The use of the more broad AI was found in design tools. [32] and [6] describe the use of machine learning, but this is not used in a design tool, as previously described. Next, in further sections, the use of RL for concept generation in other fields is described. This will provide an insight into the possibilities of RL for this application and can identify any gaps that exist in this approach.

2.2.2. Concept Generation Using Reinforcement Learning in Literature

An optimisation framework is described by [53], where deep reinforcement learning (DRL) is implemented in a design optimisation process for the optimisation of an airfoil angle of attack. It is identified that a long waiting time exists for repeating optimisation tasks. Using this waiting time to train a machine learning method for the optimisation process would allow solutions to be obtained faster when the optimisation process is carried out.

Because contour images of the flow field are used in the optimisation process, deep learning is used to extract valuable information from the images. The proposed framework makes use of a deep Q network (DQN), which can be seen as a Q-learning based RL approach. The RL algorithm is divided into a training and searching phase. During the training phase, the algorithm is trained to generate the Q function, which enables the algorithm to use this Q function during the searching phase. The action that the RL algorithm can take is whether to increase or decrease the angle of attack. The contour image of the flow field is the state, from which a reward is determined using deep learning. The optimisation process in this particular study can be seen as single objective optimisation.

Interesting research in the use of RL for the design of the agent is described by [15]. It is noted that in a RL problem, the design of the agent is seldom optimised for the problem. Instead, the only part that is optimised is the RL's policy. The work that is performed by [15] deals with optimising the design of the agent simultaneously to the policy. The application for which this is done is a navigation problem, in other words a control problem, with a physical organism. The agent can then vary the design of its body, with the goal of increasing the performance in the navigation problem. Changing the design is part of the learning process for the agent, next to the learning process for the control policy.

In order to change the design of the agent's body, the body is parameterised, which creates an extra parameter vector that is a learnable parameter. The learnable weights w then consist of the policy network parameters and the environment parameters. The aim of the RL framework is to learn w as such to get the maximum reward. Finding w is done using a population-based policy gradient method, from [51]. Exploration of the design space is encouraged using rewards for difficult designs.

A similar approach is described by [42], who also describe the joint optimisation of the physical design and control network for a robot. Due too the time consuming process of training separate policies for different physical designs, the need for an approach that simultaneously optimises the physical design and control of the agent. [42] use an approach for allowing better exploration of the design space which is of interest for this thesis. During the training process, designs that are performing positively are examined and optimised further, while designs that are not are eliminated from the process. It is claimed that this approach allows better exploration of the design space and prevents the algorithm from getting stuck in local optima. While this approach is not directly applicable to this thesis' work, the thought behind the approach is attractive. This could be used to improve the exploration/exploitation trade-off as well.

The use of RL for the design of RNA molecules is described by [39]. RNA molecules are the information-carrying biopolymers in living organisms' cells. The design of RNA deals with finding an RNA sequence which folds into a target secondary structure. The paper describes a DRL methodology for this problem. There are two elements in this methodology that train a policy, namely LEARNA and Meta-LEARNA.

LEARNA deals with the problem of finding the RNA sequence for a certain target secondary structure. The error for the RL agent is calculated by measuring the distance between the obtained secondary structure of the generated RNA sequence, and the target secondary structure. Meta-LEARNA is trained to obtain a policy from different RNA design tasks, which can be applied to new design tasks. This should decrease the time it takes to find RNA sequences for new RNA design tasks.

[38] use RL for the design generation of two-dimensional layout schemes of single-family housing units. Shape grammars are used as a design construction mechanism. Shape grammars can be seen as implementing rules into the automatic design process. If this is done manually however, these rules will cause the obtained design space to be very narrow, since the implemented rules will be created from the experience of the engineer. On the other hand, if the rules are created very liberally, called naive shape grammars, the obtained results are broad but mostly unfeasible.

To obtain feasible solutions which still span the entire design space, [38] introduces the implementation of RL for this problem. A methodology of RL combined with naive shape grammars is proposed. The RL algorithm is used to learn a policy which should construct the naive shape grammars such that feasible solutions are obtained from them. To learn this policy, the design requirements are used as rewards, so that the learned policy will generate optimised design solutions. The learning problem is solved using a generalised version of the Q-learning method, namely Q(λ).

The implementation of RL for this problem is described by [38]. Actions in this problem are the implementation of shape grammar rules to the ongoing shape. Rewards are then evaluated based on the performance of the shape for the design requirements. By dividing the design process into distinct phases, only the relevant requirements have to be assessed. The rewards are then given during these phases for the subset of requirements. Weights are applied to signify the individual importance of requirements and allow the engineer to implement their own experience into the rewarding system. Two practical issues are identified for the application of Q-learning to naive shape rules.

The first issue is the trust in the current policy, where the exploration-exploitation strategy is important. The usual course of the algorithm is to exploit the current policy. Since it is desired to also explore areas of the design space that would be left unvisited by such an approach, an ∈-greedy strategy is used. This strategy implements a random action at each step with probability ∈.

The second issue is about generalisation. It is established that the amount of state-action pairs, the possible shaped combined with the possible transformations, is too large to be saved in a table. A function approximator is used to learn the Q-values of the state-action pairs. The pairs are characterised by certain parameters, for which the Q-values are posited as a function of. A linear function is used for this approach, and is learnt using a gradient-descent rule, that can be seen in Equation 2.2. The advantage of using this function approximation is that pairs that have never been visited can be assigned appropriate Q-values based on similarity to other pairs.

$$\theta_i \leftarrow \theta_i + \alpha \times \delta \times f_i(s,a) \tag{2.2}$$

2.2.3. Concept Generation Using Multi-Objective Reinforcement Learning

In subsection 2.1.1, several sources were cited about the use of machine learning & RL for design optimisation purposes. In this section, these papers are elaborated upon, by performing a more in-depth review of the sources. Machine learning is used in optimisation to improve the efficiency of the process by learning from previous attempts of the optimisation effort, which traditional optimisation methods do not. Machine learning methods are thereby able to more easily adapt to new situations, and will increase the efficiency of the process.

The RL methodology for ship design optimisation is outlined by [10]. The methodology uses machine learning to provide a direction for the optimisation and enhances the flexibility of the optimisation for a changing design environment. The method that is used is developed to resemble natural human learning, which is used to assist the optimisation process.

A brief introduction on learning theory is given, and further elaborated upon to demonstrate the applicability of the different learning theory elements to the optimisation process. The learning theory that is accepted in this study is based on the memory model of Atkinson and Shiffrin, in which the memory process is split into three elements. MOO is introduced as well, to introduce the idea of conflicting objectives, where the aim of the optimisation process is to find solutions that are acceptable for decision makers. The goal of the optimisation process is then to find good trade-offs in the solutions between the different objectives. Pareto-optimality is introduced for this, which is the approach being used in the machine learning method.

The first element, sensory memory, is used in the method described by [10] to explore the design space, and learns from the exploration by trial and error. This is done because for different design processes, the design space might respond in other ways to this exploration. The exploration can then be used to identify the design task and to determine rules from the data. The second element, short-term memory, uses a 'central executive' center to check the in- and outflow of rules in the learning method, and determines the maturity of the rules. Finally, the third element of learning theory, the long-term memory, is used to store the mature rules and regulations.

The most important aspect of the method is described to be the ability to establish the rules through real-time learning. For this aspect, the Q-learning RL approach is used. The approach that is used can be seen as a MORL method. Q-learning is used because it is a suitable method for discrete environments and is able to implement real-time learning, finding hidden rules during the design process.

MORL is introduced by [26], to optimise the power system dispatch and voltage stability. The method is developed to cope with high-dimensional optimisation problems, because of a performance drop of other multi-objective optimisation methods for high-dimensional problems. In the study, MORL is compared to multi-objective evolutionary algorithm based on decomposition (MOEA/D). The result of this comparison is that MORL outperforms MOEA/D, both in quality of Pareto fronts and computation time. The quality of Pareto fronts is evidenced by the fact that MORL finds more accurate, broad, and evenly dispersed Pareto fronts. This can be read as such that MORL will offer a greater variation in the solutions that are found.

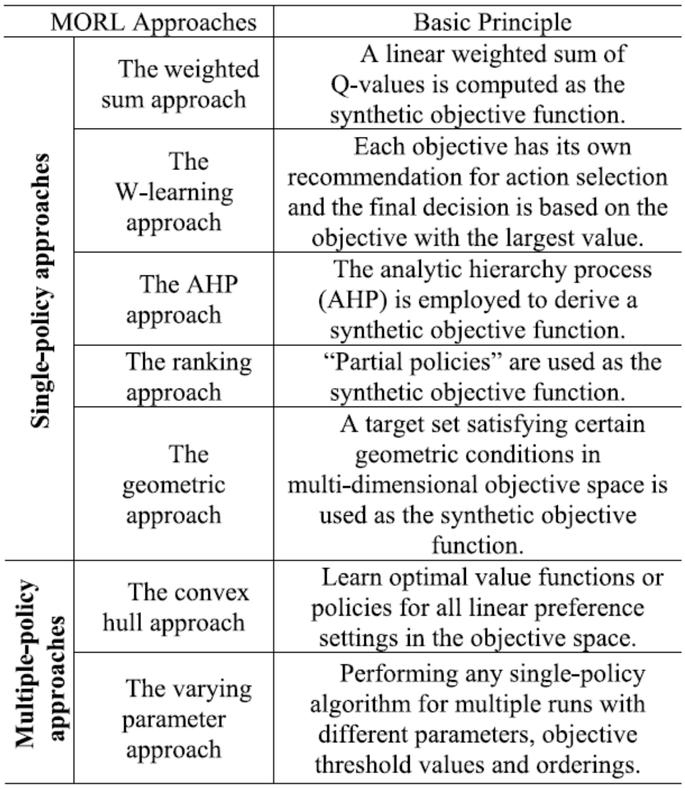
An overview of MORL methods is presented by [30]. The difference between traditional RL and MORL is that MORL needs a learning agent that can obtain action policies which in turn are able to optimise multiple objectives at the same time. This means that in MORL, each objective has its own reward signal, which results in the final reward not being a scalar value but a vector. When the objectives are conflicting, any policy of the learning agent cannot optimise these objectives, and will only be able to optimise one of them, or obtain a trade-off. It is stated that because of this, MORL can be seen as the combination of MOO and RL. Comparable to the MOO approach, MORL algorithms are divided into two techniques. The first technique is made up of single-policy MORL algorithms. A single policy aims to represent the ranking of the different objectives, for which the ranking can be user specified or obtained from the problem domain. Using this ranking and scoring based on this ranking, single reward is obtained for multiple objectives. The second technique is made up of the multiple-policy MORL algorithms. In these algorithms, multiple policies are used to approximate the Pareto front. The biggest variation between different multiple-policy algorithms is the approximation approach for the Pareto front.

Multiple solutions to the MORL problem are also described. First, [30] make use of the term naïve solutions, which aim to solve the problem by designing a synthetic objective function. One of such solutions is the use of an algorithm similar to the Q-learning algorithm. Representative single- and multiple-policy approaches to the MORL problem are explained, shown in Table 2.3, which is obtained from [30].

Multi-Objective Machine Learning

An overview of Pareto-based multi-objective machine learning is created by [17]. The overview is restricted to the Pareto-based multi-objective supervised and unsupervised learning approaches. This paper is reviewed because it might provide useful insights for MORL.

Table 2.3: MORL problem representative approaches, obtained from [30]



MORL Approaches		Basic Principle
Single-policy approaches	The weighted sum approach	A linear weighted sum of Q-values is computed as the synthetic objective function.
	The W-learning approach	Each objective has its own recommendation for action selection and the final decision is based on the objective with the largest value.
	The AHP approach	The analytic hierarchy process (AHP) is employed to derive a synthetic objective function.
	The ranking approach	"Partial policies" are used as the synthetic objective function.
	The geometric approach	A target set satisfying certain geometric conditions in multi-dimensional objective space is used as the synthetic objective function.
Multiple-policy approaches	The convex hull approach	Learn optimal value functions or policies for all linear preference settings in the objective space.
	The varying parameter approach	Performing any single-policy algorithm for multiple runs with different parameters, objective threshold values and orderings.

The objective learning algorithms are divided into three categories, based on the way the objectives are evaluated. The three categories are single-objective learning, scalarised multi-objective learning, and Pareto-based multi-objective learning. Single-objective learning uses algorithms where a single objective function is optimised. For supervised learning, the mean squared error (MSE) method is often used for training, while for unsupervised learning the k-means clustering algorithm is most often used. It is noted that learning inherently is a multi-objective problem. This is because using a single objective for the training data could result in overfitting of this data, which would result in poor performance of the model on unseen data. Several weaknesses of using a scalarised multi-objective function for a multi-objective optimisation problem are identified. Selection of the hyperparameter, which represents the user's intent in the optimisation process, is difficult. Next to that, scalarised multi-objective functions will only obtain a single solution from the multi-objective optimisation problem. This presents issues if in reality the objective conflict eachother, for which not a single optimal solution exists. Pareto-based multi-objective learning on the other hand, is able to find a multitude of solutions, the Pareto-optimal solutions. Learning problems in Pareto-based multi-objective learning is handled similar to Pareto-based multi-objective optimisation, according to [17]. The advantage of this approach is identified as being able to extract knowledge from the Pareto-optimal solutions, by having a better picture of the design space, and thereby being able to choose a final solution with increased knowledge about the problem.

An overview of the research that is being performed using Pareto-based multi-objective supervised and unsupervised learning is also provided. The research using supervised learning is divided into three categories, based on their focus, which is either generalisation improvement, interpretability enhancement in rule extraction, and finally diverse ensemble generation.

Generalisation improvement deals with the fact that the learning models should have a good functionality on unseen data, and not only on the training data. To achieve this, [17] states several approaches. Taking other objectives next to the training error into account, improving the generalisation performance of neural networks by minimising error measures that could be conflicting. Pareto-based multi-objective learning can be used to optimise this behaviour.

Interpretability enhancement in rule extraction is aimed to increase the interpretability of logic or fuzzy rules that are extracted with the model from data or the trained models. Important elements of the interpretability of the rules are the compactness and the consistency. The advantage of a Pareto-based approach here is identified as the fact that the user is able to choose from a set of Pareto-optimal solutions.

Finally, the research on diverse ensemble generation is performed because the use of an ensemble of learning models has a better performance than a single learning model. The research deals with what models to choose for use in the ensemble of models, because there exists a trade-off between accuracy and diversity of the ensemble of models. Here, Pareto-based multi-objective learning can be used to optimise these objectives.

In multi-objective unsupervised learning, research on Pareto-based approaches is mainly limited to data clustering. Pareto-based approaches can be used to identify significant features and the right amount of clusters for the problem. It could also be used to automatically identify the amount of clusters.

I

Design Tool Setup

3 Design Tool Reinforcement Learning Algorithm Selection

In chapter 2, it is identified that a machine learning method should be used for the generation of CubeSats concepts, and more specifically a RL method should be used. In this part, Part I, a RL algorithm is selected and the design tool that uses this algorithm in order to generate CubeSats concepts is detailed. In this chapter, chapter 2, the basics of the selected RL algorithm is explained. After that, in chapter 4, experiments are performed using a preliminary design tool. Building on the results of these experiments, the final setup of the design tool is defined in chapter 5.

3.1. Reinforcement Learning Algorithm Selection

After the selection of RL as the method that should be implemented in the design tool, a RL algorithm should be selected. A broad division can be created for RL algorithms, which is the division between a model-based or a model-free system. In model-based systems a model of the environment is constructed which is used to foresee the effect of actions on the environment. Using this capability, model-based systems choose the optimal policy based on this model. Model-based systems are statistically more efficient than model-free systems because of a better knowledge of the environment [11]. However, the models in model-based systems have to be reasonably accurate to be useful [45]. If it is difficult to construct a sufficiently accurate environment model, model-free systems can still have advantages over the more complex model-based systems. Due to the high complexity of the design of a space system, constructing an accurate model of the design space created by a catalogue of COTS components is thought of as infeasible at this point. It is reasoned that the algorithm that will be selected for this work should therefore be a model-free system.

Model-free RL makes use of experience to learn one or both of two different quantities, the state/action values or policies. This can also provide the optimal policy but without the construction of a model of the environment. The quantities have values that will indicate the promise of actions taken from the states. The design tool is at this point envisioned to be able to implement components directly in the design cycle. This means that the design tool should be able to select components from a COTS components catalogue. Components in such a catalogue create a discrete design space for the design tool, as the components specifications will not be evenly distributed over the design space. The RL algorithm that is selected should be able to deal with such a discrete design space or in terms of a RL system, a discrete state space. The actions that the RL algorithm can take are whether to select a different component or not and the actions of the RL algorithm are also discrete.

Algorithms that can handle this kind of design and action space directly are tabular solution methods. Tabular solution methods are divided into three fundamental classes: dynamic programming, Monte Carlo methods and temporal difference learning [45]. Because dynamic programming requires a complete and accurate model of the environment, this class of methods is not taken into account further. To solve the prediction problem, the Monte Carlo and temporal difference (TD) learning classes use experience. The difference between the classes lies in the timing when the policy is updated. Monte Carlo methods wait until the end of an episode, while TD methods will wait until the next step. The fact that TD methods are implemented in an online and incremental way is an advantage of TD methods over Monte Carlo methods. Both methods are proven to converge asymptotically to the correct predictions, but TD methods usually converge faster than Monte Carlo methods on stochastic tasks [45].

The selection of a RL algorithm for the design tool is now between the Sarsa and Q-learning algorithms, which are TD tabular model-free methods. The difference between the algorithms is that Sarsa is on-policy TD control while Q-learning is off-policy TD control. The off-policy approach is a more straightforward approach, where two policies are used to differentiate between exploration of the design space and for exploitation. The on-policy approach will always explore to improve the decision making policy. The possibility exists that the behaviour policy is unrelated to the target policy, which on-policy methods do not differentiate. The on-policy methods are simpler, but less powerful and general than off-policy methods [45]. It is therefore decided that the Q-learning algorithm is the most suitable algorithm of the two for this work.

It should be noted that the selection of the RL algorithm was performed with the knowledge on RL methods at that time. During the course of the work the limitations of the currently selected algorithm became visible, which will be discussed in later parts of this work. Furthermore, the knowledge on RL methods was also developed during the duration of this work.

3.2. Reinforcement Learning Elements

3.2.1. General Reinforcement Learning Elements

The general application of reinforcement learning for generating concepts can be explained by looking at the basic elements of a reinforcement learning problem. For these elements, they can be explained in the way they translate to this work's problem. In Figure 3.1, a basic reinforcement learning scenario can be seen. The following elements are identified; agent, policy, reinforcement learning algorithm, action, environment, reward, and observation. Below, the elements of this scenario are translated to the problem in this thesis.

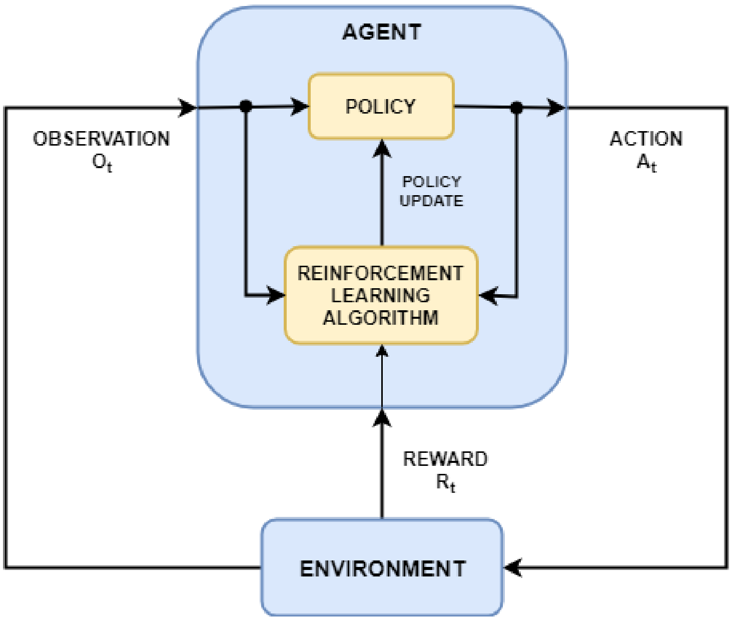


Figure 3.1: Basic reinforcement learning scenario, from [1]

- **Agent**: The agent is both the learner and determines the actions to take in the process. The agent can be seen as the embodiment of the design process.
- **Policy**: The policy determines what action is most favourable to take, in order to gain the best

reward. While the training of the agent is ongoing, the policy is updated at each iteration to reflect the best route to take in the reinforcement learning problem. It therefore determines the action that the agent will take.

- **Reinforcement learning algorithm**: The RL algorithm of the agent uses the rewards from actions taken by the agent in order to determine policy updates.
- **Action**: The action in the reinforcement learning problem dictates the route that the agent will take in the environment. For the thesis the action determines the components that are chosen for the satellite design.
- **Environment**: The environment can be seen as the design space of the problem. In the case of a control reinforcement learning problem, the environment could for example be the terrain over which an agent is trying to navigate. The environment is where the agent learns and takes actions. In the thesis' case, the environment consists of the component configuration options which make up the design space.
- **Reward**: The performance of the action in the environment produces the reward. The reward provides an indication of how well the action has performed, and can take a range of values which have to be determined. In the thesis, the performance of the action can be determined by evaluating the performance of the component selection, the action, to the mission requirements.
- **Observation/States**: The observation or state is the current location of the agent in the environment. For the thesis this means that the state is the current configuration of components.

The reinforcement learning approach that will be used for this work will need to be further defined in terms of the state space, action space, reward model, and what algorithm should be used. It was identified that the promise of using a machine learning approach is the ability to re-use knowledge from previous runs. The work should reflect on this capability of the used approach.

3.2.2. Q-learning Algorithm

Q-learning was developed by [49] and is an off-policy TD control algorithm. Using this algorithm, the optimal action-value function is directly approximated by the learned action-value function [45]. The policy that is used determines which state-action pairs are visited and for which the Q-values are thereby updated. The algorithm is defined in Equation 3.1. Q-learning establishes Q-values for every state-action combination. These Q-values determine the optimal action at each state, where the Q-values are updated using Equation 3.1. In this equation, Q is the Q-value, S the state, A the action, α is the learning rate, R the reward and γ the discount factor. This equation takes into account the maximum possible future Q-value to determine the updated Q-value for the current state-action pair.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \underset{a}{max} Q(S_{t+1}, a) - Q(S_t, A_t) \right] \tag{3.1}$$

The state space determines in what situation the agent can find itself in. For the generation of CubeSat concepts, this situation can be seen as what components are selected by the agent. It is envisioned at this point that it is possible to create three kinds of state spaces for the design tool, which are listed below. The main distinction is whether the state space is made up of the components, which is done in state spaces 1 and 2, or whether the state space is independent of the components, which is done in approach 3. The different state spaces are further elaborated upon in subsection 4.1.3. The performance of each state space using the Q-learning algorithm is investigated in chapter 4, and a decision is made as to which state space kind will be used for the final setup of the design tool.

1. **State space 1:** Using components directly in the state space.
2. **State space 2:** Using components indirectly in the state space.
3. **State space 3:** Learning on a state space which is independent from the components.

The action space determines what actions the agent can take when it is in a certain state. Depending on the kind of state space that is used, the action will either directly or indirectly change the components that are selected or change other parameters. How greedy the agent takes actions is driven by the parameter ϵ. ϵ determines whether a random action should be taken or an action indicated by the maximum value at that state in the q-table. The epsilon will decay during a training cycle, which will ensure that the state-action space is explored at the beginning, but exploited towards the end of a training cycle.

The next element is the reward system. This system determines in what situation what reward should be given during an episode. An episode is one run through the environment. For this basic scenario a system is used which gives a positive reward when the user requirements are met by the state-action pair, and then also exits the episode. A move penalty is given when the user requirements are not yet met, and an action should therefore be performed to achieve the next state.

The learning rate α and discount factor γ both range between 0 and 1. The learning rate is an indication of how important new information is over old information. A learning rate of 1 will mean the agent only considers new information. The discount factor controls the importance of future rewards. A value of 1 means the future rewards are emphasised over short-term rewards. The influence of these values on training agent for the basic scenario will be further researched.

The pseudo code of the Q-learning algorithm that is implemented in the design tool can be seen in Algorithm 1.

Algorithm 1 Q-learning algorithm

```
Initialise $Q(S, A)$
Initialise $\epsilon$

for each episode do:
    Initialise state $S$
    for each iteration of episode do:
        Observe state $S$
        Choose action $A$
        Take action $A$
        Determine state performance
        Determine reward $R$
        Observe reward and state $R, S'$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$
        Update $Q(S, A)$
    end for
    Decay $\epsilon$
end for
```

4

Design Tool Setup Experiments

In chapter 3, three different kinds of state spaces were identified for the implementation of the Q-learning algorithm in the design tool. In this chapter, these state spaces are investigated and a selection is made as to which kind of state space will be used in the final setup of the design tool. The setup of the design tool used to experiment with the different kind of state spaces will be explained first in section 4.1. A relatively simple setup is used for these experiments, which could identify future issues and help set RL parameters, such as the training parameters, for the final setup. The results of the experiments using the different state spaces are presented and discussed in section 4.2 and the kind of state space that will be continued to be used in this work is selected. Some follow up experiments are further defined and presented in full in section 4.3.

4.1. Experiment Settings

The implementation of the Q-learning method into the design tool for the experiments is detailed in this section. First, the environment of the experiments is characterised, after which the Q-learning settings will be defined. Finally, the state space options, which where identified in subsection 3.2.2, will be defined in more detail.

4.1.1. Experiment Environment

The thesis' full problem has to be translated to a relatively simple scenario to experiment on with RL methods. A simple scenario that can be used for implementing different RL methods should have a relatively small state-action space. A small state-action space means that the RL agent will have a limited number of directions to explore and will therefore be trained faster. In this way, results can be generated for a large amount of agents using different methods and settings. Next to that, it should be identified what the basic scenario should have as output or goal.

A small state-action space is created by limiting the number of states that the agent is able to attain and the number of actions it is able to take at each state. The states of the RL problem is indicated by the size of the environment. To create a simple scenario, the environment should be kept simple as well. The number of actions that the agent is able to take at each state, depends on how the environment is set up as well. An example of this is a scenario where the agent has to choose two components for two subsystems. A state in this scenario could be the component index from the list of components for each subsystem, where the possible actions for the agent could be to go up or down the lists to choose different components.

For the experiments in this chapter, the design tool is set up as a problem in which the RL agent has the goal of identifying components that satisfies user provided requirements. To simplify the problem, this is performed for a single subsystem of a satellite, which for the scenario is the payload and more specifically a camera. Next to that, the subsystem itself is simplified in such a way that the amount of parameters that the payload consists of.

For the scenario, it is assumed that the payload components are characterised solely by the parameters focal length and number of horizontal pixels. The user requirements for this payload are the ground sample distance (GSD) and the horizontal swath (S) on the ground. Next to that, the user is able to specify the orbital altitude of the satellite. Finally, an assumed parameter for the scenario is the diameter of a single pixel, which is assumed to be 3 micrometer. In order to identify whether a component satisfies the user requirements, some simple calculations are performed. The GSD is calculated using Equation 4.1, and horizontal swath is calculated using Equation 4.2. For this scenario, an orbital altitude of 400 km is assumed. The parameters are summarised in Table 4.1. A threshold of 10% between the performance of a components and the user requirements is used to determine whether a component satisfies the user requirements.

$$GSD = \frac{Hd_{px}}{f} \tag{4.1}$$

$$S_h = 2tan^{-1}\left(\frac{d_{px}}{2f}\right)n_{px}H \tag{4.2}$$

Table 4.1: Basic scenario parameters

Parameter	Value	Unit
Orbital altitude	400	km
Pixel diameter	3	micrometer
Focal length	Variable	m
Number of pixels	Variable	-
Required GSD	1	m
Required Swath	10	km

4.1.2. Q-learning Settings

For the experiments, the Q-learning settings that are identical between the different approaches are established in this section. A training cycle consists of a certain number of episodes. For each episode a certain number of iterations is performed, where at each iteration an action is taken. The number of episodes and iterations per episode for the basic scenario is 25000 and 100, respectively. These numbers are perceived to be sufficient for training the agent and finding a solution within the number of iterations. This will be shown graphically in section 4.2

A state space of size 100 for each single component or parameter is used. This means that for the third approach, which uses an independent state space consisting of the parameters, the state space consists of 2 times 100 options. This means that a total number of $100x100$ state space possibilities is created.

The reward system that was described previously will give a reward when the user requirements are met and a move penalty otherwise. The user requirements are said to be met when the performance of a state is within a 10% difference of the user requirements. This reward is set as 25 while the move penalty is 1. For a 100 iterations this results in a minimum and maximum episode reward of -100 and 25, respectively.

The parameter that determines the greediness of the agent is ϵ. It is decided to decay this value during a training cycle, so that a greedy approach is used at the beginning and a non-greedy at the end of the training cycle. The decay equation used for this can be seen in Equation 4.3. Because a certain level of randomness is still desired near the end of the training cycle, the decay has a minimum value, which is 0.1. The evolution of ϵ is shown in Figure 4.1.

$$\epsilon_{new} = \epsilon_{old} * \epsilon_{decay} \tag{4.3}$$

The values for the learning rate α and discount factor γ, together with the previously described elements are summarised in Table 4.2.

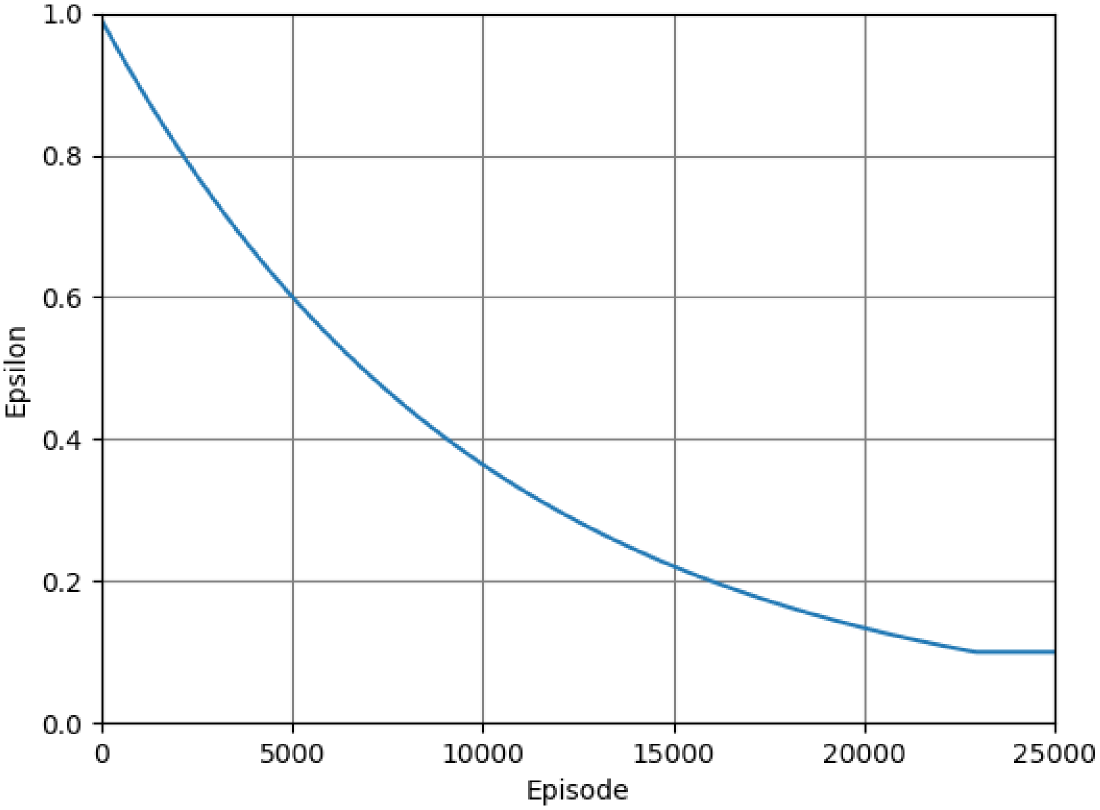


Figure 4.1: Epsilon decay for training cycle

Table 4.2: Experiments Q-learning settings

Setting	Value
Number of Episodes	25000
Number of Iterations	100
Requirement Reward	25
Movement Penalty	-1
Epsilon Initial	0.99
Epsilon Decay	0.9999
Epsilon Minimal	0.1
Learning Rate	0.3
Discount Factor	0.9

4.1.3. State Space Options

In section 3.2, the general aspects of implementing Q-learning for the basic scenario are explained. Next to that, three different options for creating the state space of the basic scenario were identified. From these three different options, three different approaches for implementing RL for the basic scenario are formed. This section will go into detail for each of these different approaches.

State Space 1: Component Directly

The next approach uses components directly in the state-action space. For the basic scenario, 100 randomly generated components are used, with the same maximum and minimum range as previously specified for their parameters. Because comparability and reproducibility are the main goals of this basic scenario, the same set of randomly generated components are used for every training cycle.

This set of components can be seen in Figure 4.2.

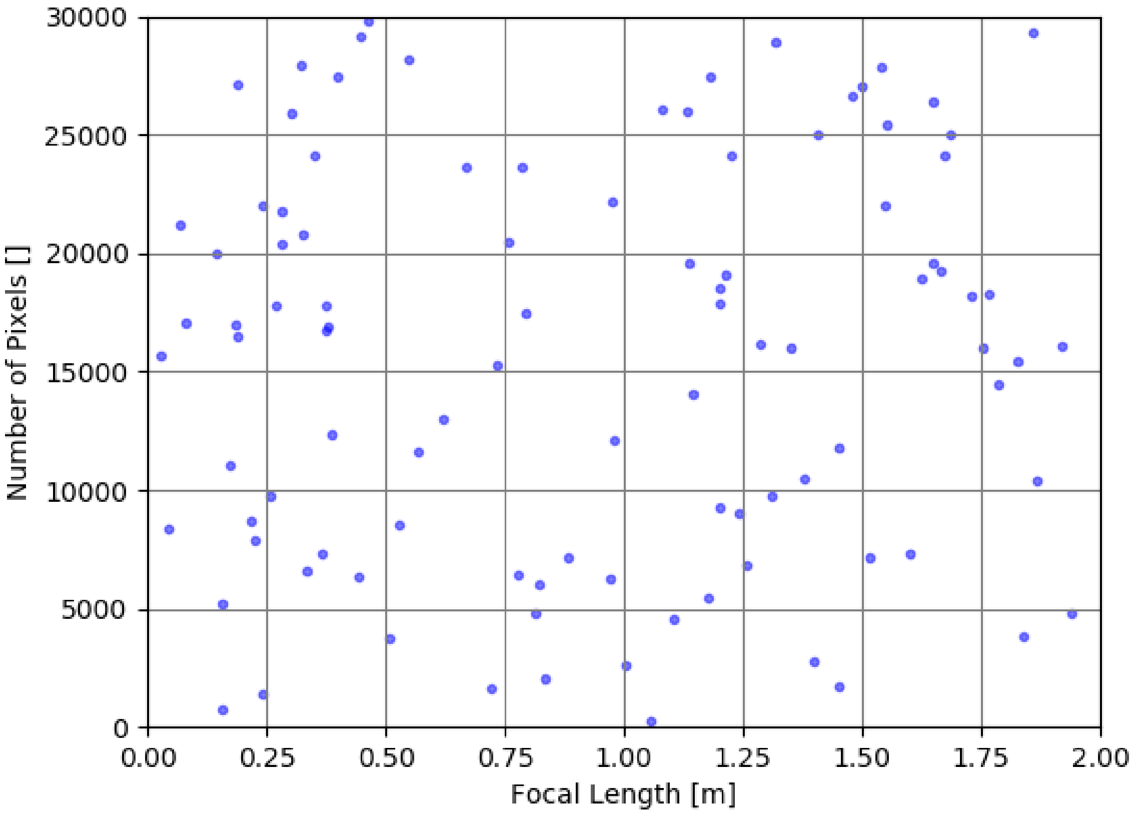


Figure 4.2: Component set for basic scenario

To prepare for the use of value function approximation in this approach, the components are sorted in a list based on the value of their parameters. The lower the combined relative value of the parameter, the lower the component will be sorted in the component set. This is illustrated in Figure 4.3, where the line from lower left to upper right indicates the order in which the component set is sorted.

The state space of this approach is the component set, where the state takes the form of an index in this set, indicating the component being used at that iteration by the agent. The action that the agent can take is to either go up, stay at the same, or go down in the component set list. In this way, the action that is taken determines the component that is being used. The state-action space is generated from the combination of every action at each possible state. It can be noticed that the size of this state-action space is significantly lower than for the independent parameter approach described in Figure 4.1.3.

The performance that is achieved at each iteration can be calculated by calling the component parameter values of the component that is being used at that iteration. This is possible because the sorted component set is saved and does not vary during training. In this way the reward system that can be used is similar to the previous approach in Figure 4.1.3.

State Space 2: Components Indirectly

The difference of this approach as compared to the direct components approach of subsection 4.1.3 is in the state space that is generated by the components. Instead of sorting the components and using these directly, a fit is generated through the components parameters. This fit is then used as the state space for the basic scenario. To create the fit a polynomial regression line of degree 3 is used. The fit for the basic scenario component set can be seen in Figure 4.4.

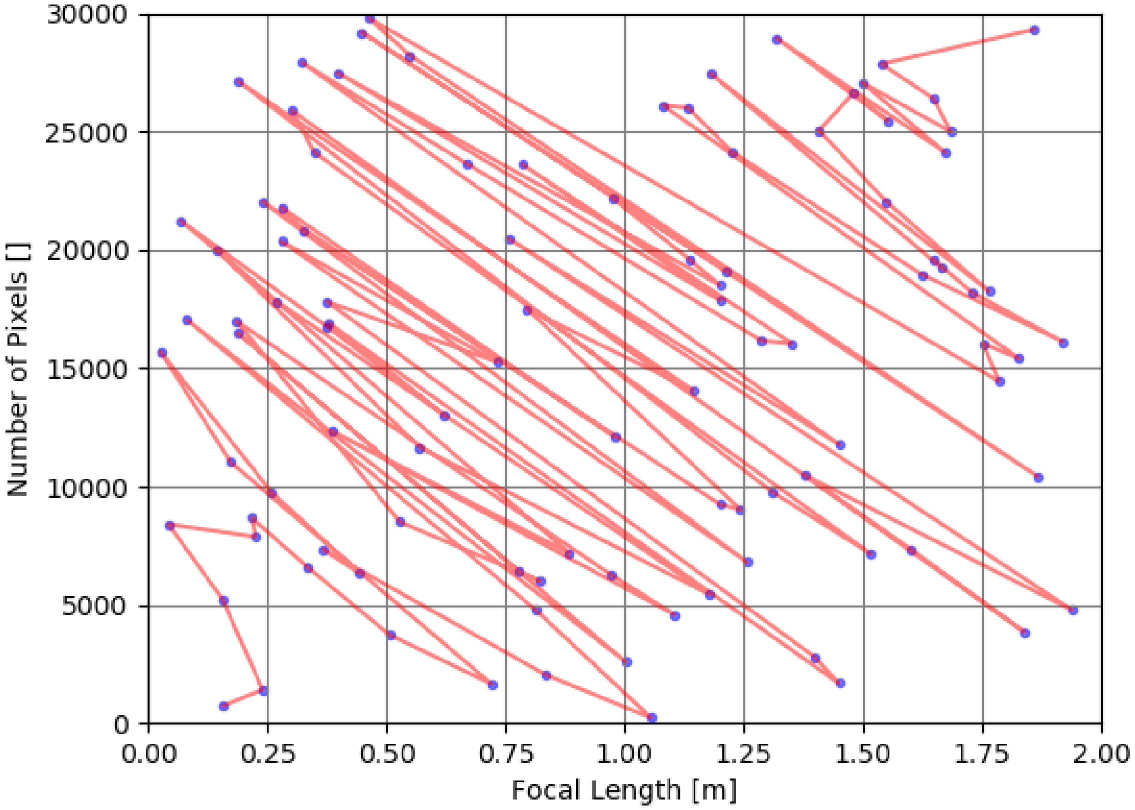


Figure 4.3: Sorted component set for basic scenario

The state space is created by generating a list of elements on the fit, which consists of the same amount of elements when compared to the original component set. For this scenario, the elements are constructed by calculating the number of pixels for a range of focal lengths, ranging from the minimal to maximal focal length, using the established polynomial regression line. The actions in this approach are similar to the actions in subsection 4.1.3, and the resulting state-action space is therefore of equal size to the approach of this section.

State Space 3: Independent Parameters
The first approach that is discussed, is the approach that uses independent parameters for the state-action space. This approach is the most basic of the three, as it is independent of any components. The main feature of this approach is that the environment is made up of the components' parameters, instead of the components themselves.

For the basic scenario there is only a single subsystem, which is made up of two parameters, focal length and number of pixels. For this approach, rather than looking at the components, the state space is made up of all possible combinations of the components' parameters. The range of the parameters are assumed, and the state space can take all possible values in this range. The step size of an action taken is defined by Equation 4.4.

$$STEP_{ACTION} = \frac{(PARAM_{MAX} - PARAM_{MIN})}{100} \tag{4.4}$$

What this results in, is that the state can take values for every parameter between the maximum and minimum value of these parameters. The complete state space then consists of the combinations of these values. The actual size of the state space depends on the magnitude of the action step size. The

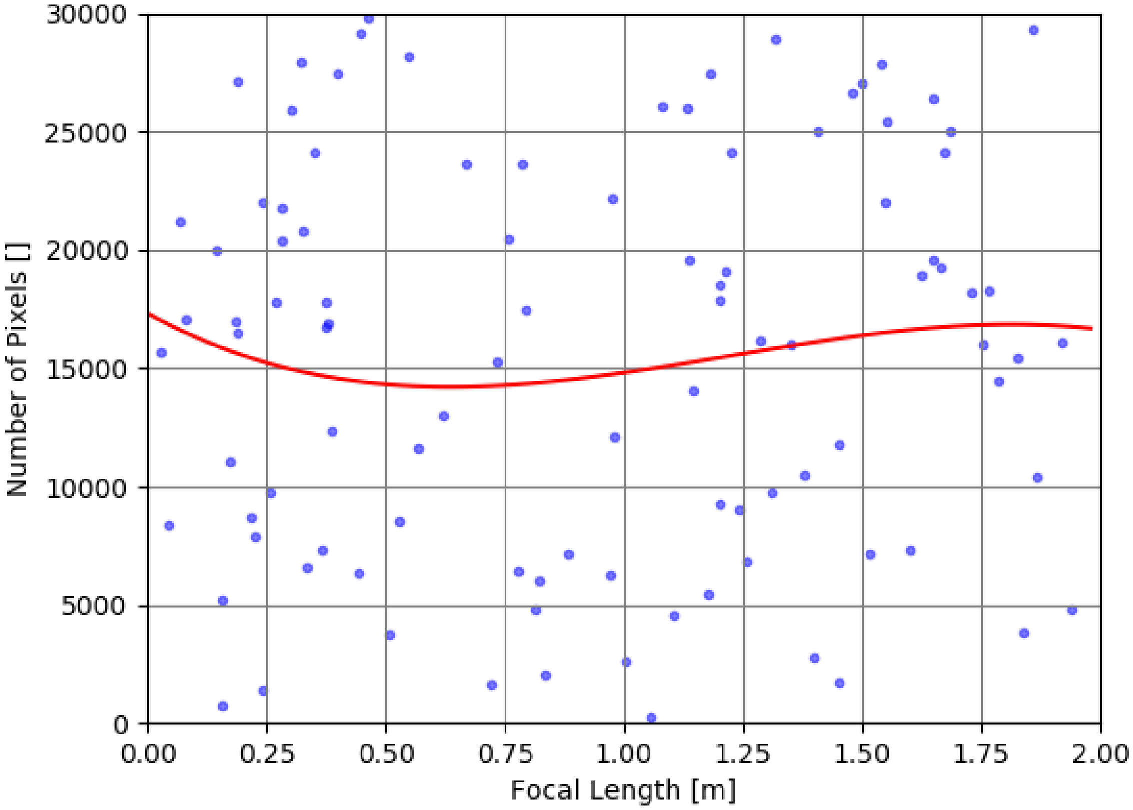


Figure 4.4: Polynomial regression line for basic scenario component set

action space then also consists of every combination that can be made for changing the state. Because the possible actions for a single parameter is to increase, to stay, or to decrease the parameter, the number of actions of the complete action space for this kind of approach is given by Equation 4.5. Since the number of parameters for the basic scenario is 2, the complete action space consists of 9 different actions. It can already be noticed that for an increase in parameters, the size of the state-action space becomes considerable quickly. This means that this approach might become relatively computationally expensive as the size of the scenario is increased.

$$NUM_{ACTIONS} = 3^{NUM_{PARAM}} \tag{4.5}$$

4.2. Experiments Results

The three different state spaces have been established in subsection 4.1.3. In this section, the achieved results with the settings from section 3.2 for the three state space options are given. Because there is a random factor during each training cycle, a total of 50 training cycles are performed for each approach to generate results that can be averaged over. The results are also discussed in this section. For the sake of this discussion, the ’Components Directly’ state space will be referred to as state space 1, the ’Components Indirectly’ state space as state space 2, and the ’Independent Parameters’ state space will be referred to as state space 3.

Several similar figures are presented for each state space option. First, there is the average reward over the last n amount of episodes. The second and third figure represent the maximal and minimal reward of the last n amount of episodes, respectively. Because a total of 5 training cycles is used, it is desired to show the average of these training cycles, and the variation between all cycles. The average of the training cycles is indicated by the bold line in the graph, while the shaded area shows the maximal and minimal values that are reached during the cycles.

4.2.1. State Space 1 & 2: Components Directly & Indirectly Results

The results for the components directly state space are shown in Figure 4.5a to Figure 4.5c. The results for the components indirectly approach are shown in Figure 4.6a to Figure 4.6c.

For state spaces 1 and 2, the patterns in the average, maximal and minimal graphs are what might be expected of a RL agent. In both Figure 4.5a and Figure 4.6a, the average reward converges to slightly above 0 at the end of a training cycle. The maximal rewards, shown in Figure 4.5b and Figure 4.6b, almost immediately converge to the maximal possible reward value, 25. This indicates that the agent is able to find a feasible solution relatively quickly early on in the training cycle. It appears that for state space 2, in Figure 4.6b, a maximal reward value of 25 is almost immediately present. It is thought that this can be explained by the fact that the amount of feasible solutions is higher when compared to state space 2, as can be seen in Figure 4.8. An episode reward of 25 can only achieved when the episode initialises inside the feasible region, otherwise a move penalty would be deducted, giving a lower episode reward value. Due to the higher amount of feasible elements for state space 2, the chance that the agent initialises on one of these elements is also higher, thereby explaining the behaviour in Figure 4.6b. A similar behaviour for the minimal episode reward is observed for state spaces 1 and 2, in Figure 4.5c and Figure 4.6c. A slightly higher final average minimal episode reward is observed for state space 1, indicated by the bold line. Because of the similarities in the behaviour between the graphs, it is not thought this shows any significance at this point in time.

The similar behaviour for state spaces 1 and 2 can be expected, because these approaches mostly rely on similar methods for finding the feasible solutions. For both state space options, the size of the state-action space is similar and the RL algorithm explores and exploits this space in the same way. This can also be seen in the time that is required to perform a training cycle, which can be seen in Figure 4.9, where both approaches show a similar time required. Because state space 2 has more feasible solutions, the behaviours in the maximal episode reward graphs are slightly different.

4.2.2. State Space 3: Independent Parameters Results

The results for the independent parameters approach are shown in Figure 4.7a to Figure 4.7c. Note that in Figure 4.7c, the data that is shown does not vary from a reward value of -100, which makes the figure difficult to read.

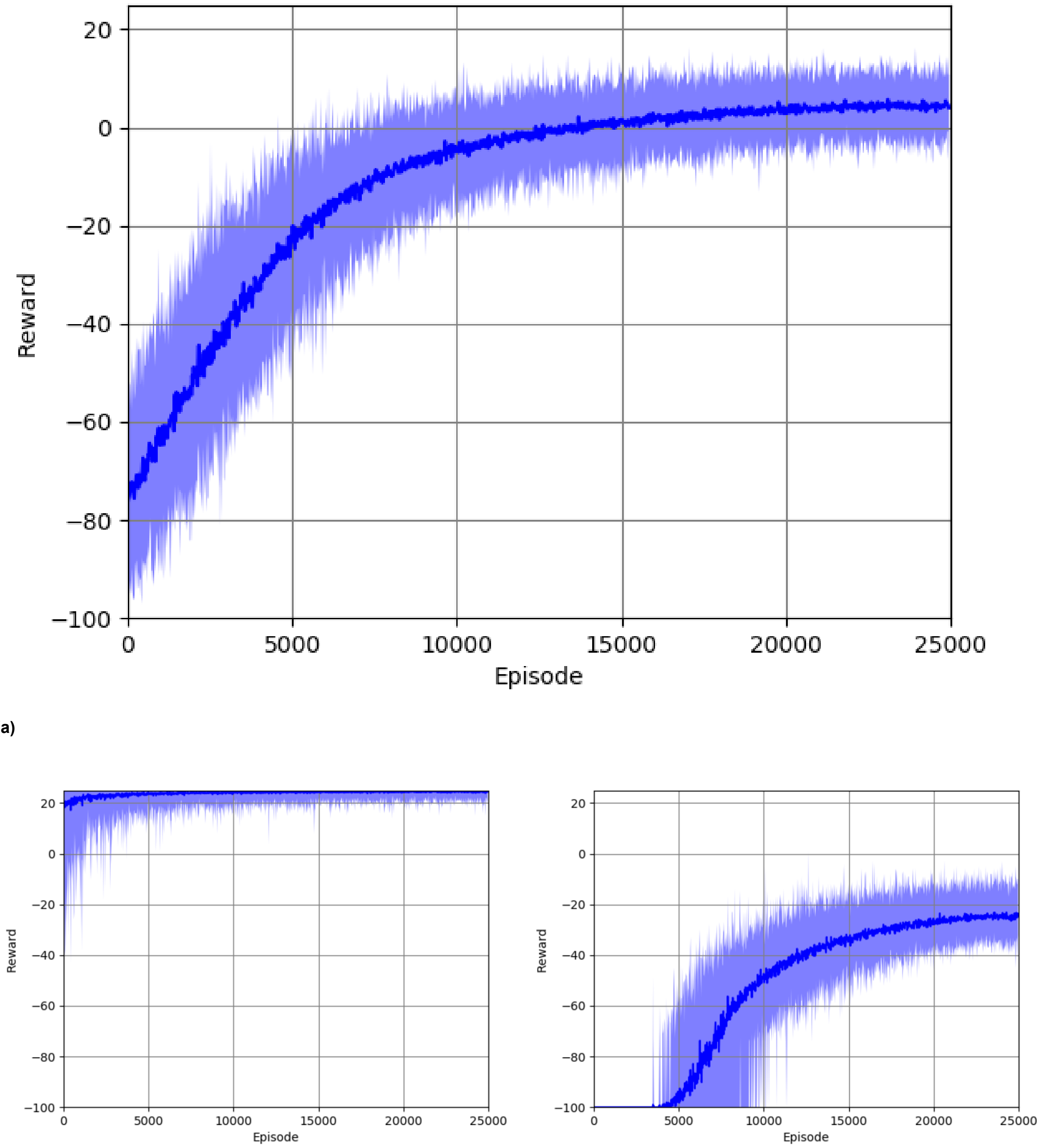
In the results of state space 3, it can be seen that the performance of this option is wanting. In Figure 4.7a, the average reward does not increase significantly with increasing episode number, meaning the agent often fails to find a parameter set that meets the user requirements. This can also be seen in Figure 4.7c, where the minimal reward is -100 throughout the full every training cycle. This means that the agent is not able to find a feasible solution and therefore obtains the worst reward possible.

This behaviour of state space 3 however, is not unexpected. Because of the increased state and action space size when compared to state space 1 and 2, the state-action space size is also significantly larger. The agent of state space 3 can perform the same number of actions, limited by the number of iterations, in this increased space as the other approaches. Whereas the agents in state space 1 and 2 are hypothetically able to traverse their full state space within this number of iterations, the agent in state space 3 is not. When the environment of state space 3 is initiated at a state that is a number of actions larger than the number of iterations away from a feasible solution, the agent is guaranteed to not find a solution. Since the initiation of the environment is done randomly, it is also guaranteed that this situation can occur during every episode of the training cycle.

4.2.3. Shared Results

Finally, for every approach the solutions that were identified by the agents during the training cycles are illustrated in Figure 4.8. The evaluation time for each training cycle is shown in Figure 4.9.

State space options 1 and 2 are both able to find feasible solutions for the user requirements. An important aspect of the evaluation of these approaches however, is to think about what would happen



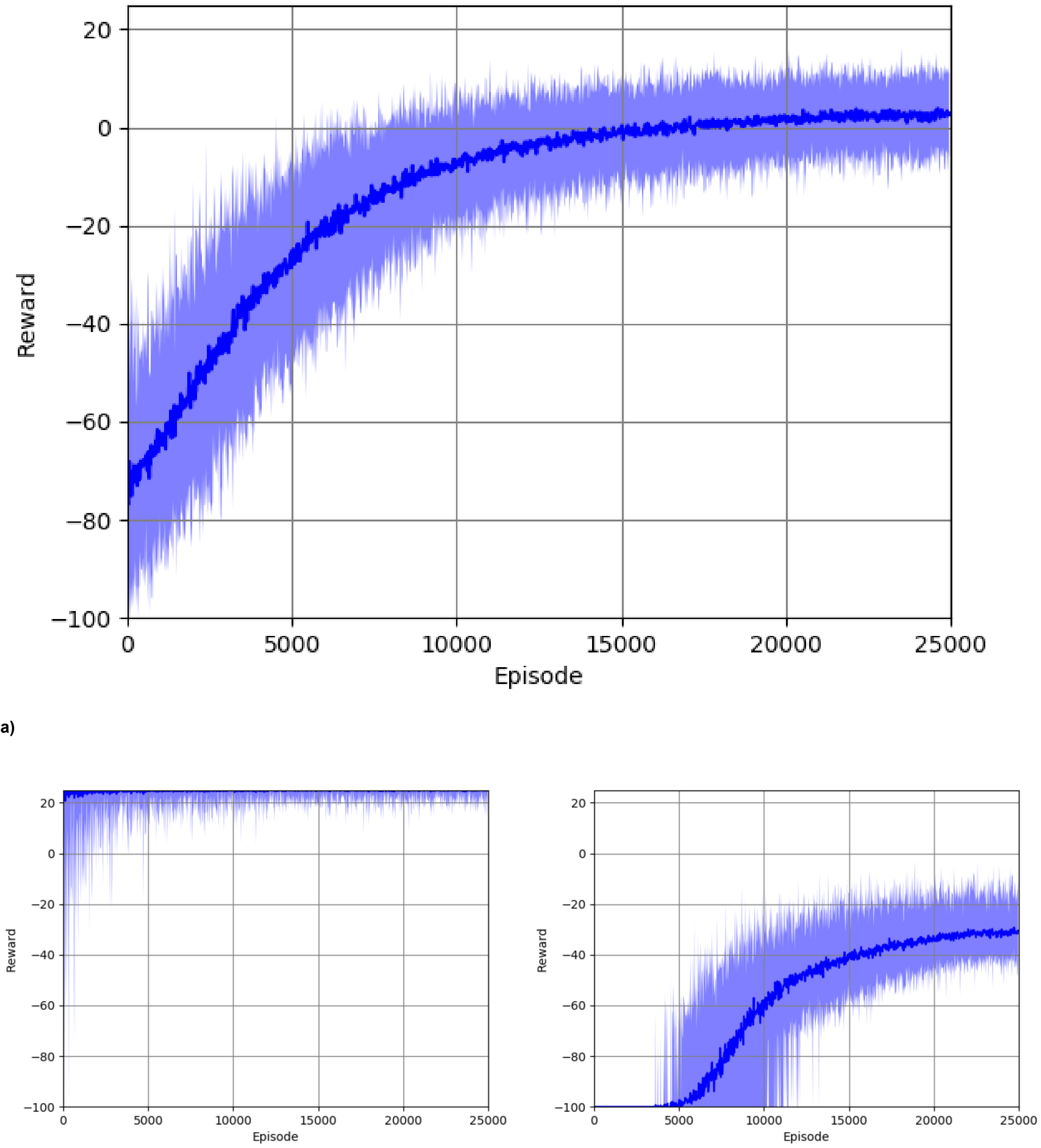
(a)

(b)

(c)

Figure 4.5: Rewards as a function of episode number for the 'Components Directly' approach to the basic scenario with (a) Average reward with variation. (b) Maximal reward with variation. (c) Minimal reward with variation.

if this were not the case. Because state space 3 is able to freely traverse the possible range of the parameters, this approach is able to find the feasible region, which is limited by the earlier specified threshold. This is the main benefit of state space 3, since it will always be able to find this feasible region, regardless of the user requirements and the components. If no components would be present inside this feasible region, and the fitted line also does not traverse this region, state space options 1 and 2 would not be able to find feasible solutions. In this case, training the agents using the same reward system as is used now will yield no functional results. It can be concluded that to be able to use



(a)

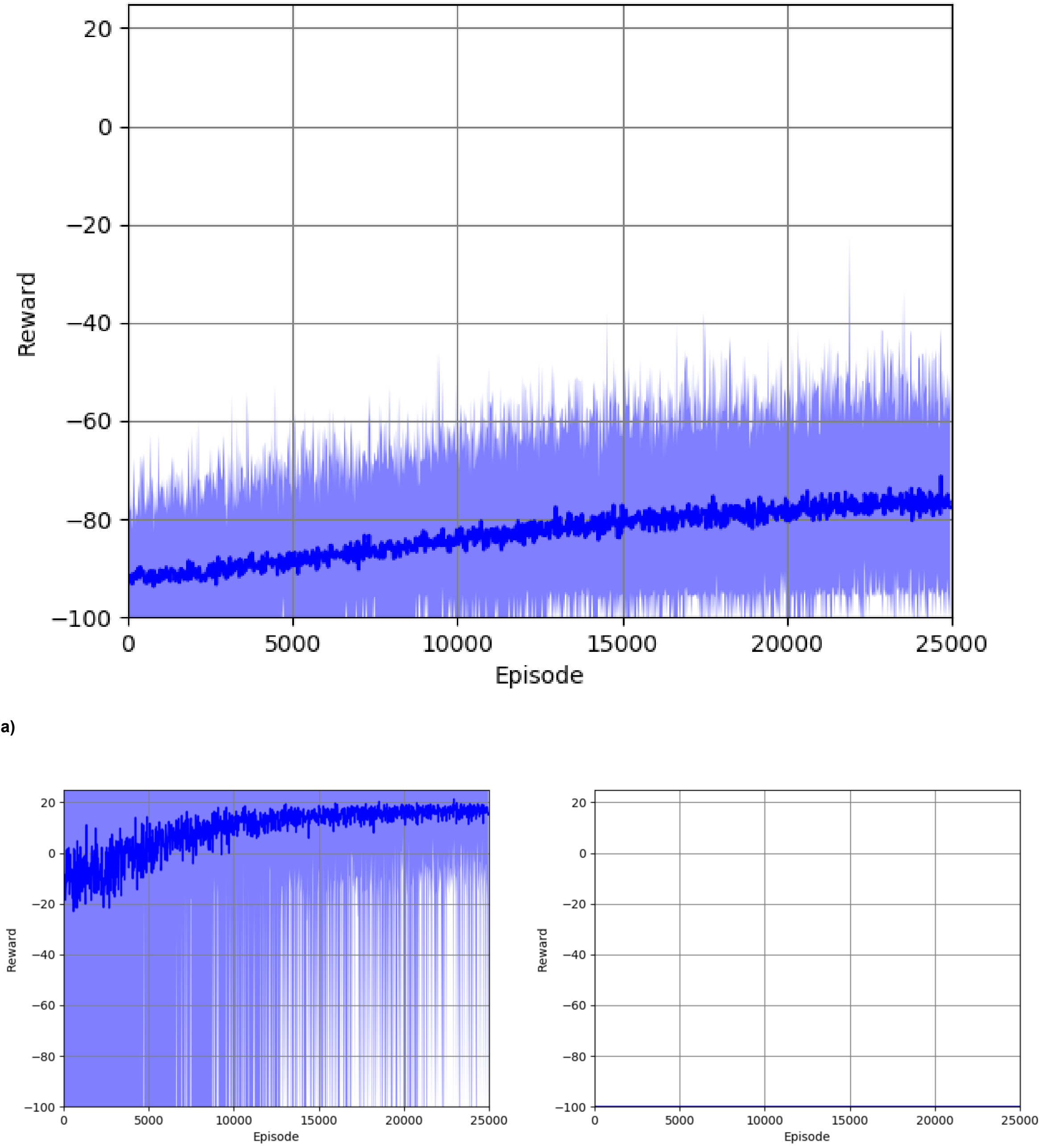
(b)

(c)

Figure 4.6: Rewards as a function of episode number for the 'Components Indirectly' approach to the basic scenario with (a) Average reward with variation. (b) Maximal reward with variation. (c) Minimal reward with variation.

state space options 1 and 2 for future experiments, a new reward system will have to be implemented in order to cope with changing user requirements and/or component sets.

Since the behaviour of state space options 1 and 2 are very similar in most aspects, these will be compared in combination with respect to state space 3. The disadvantage of state space 3 is both the episode reward performance for similar settings and the time that is required to perform a training cycle, which is illustrated in Figure 4.9. Note that these disadvantages are mutually dependent, because finding a feasible solution more quickly leads to higher rewards and shorter episodes, and thereby



(a)

(b)

(c)

Figure 4.7: Rewards as a function of episode number for the ’Independent Parameter’ approach to the basic scenario with (a) Average reward with variation. (b) Maximal reward with variation. (c) Minimal reward with variation.

improving episode reward and time performance. The advantage of state space 3 over the other state space options however, is the fact that it is able to identify the feasible region for any set of user requirements. When expanding the scenario, it might be interesting to use state space 3 in order to verify another approach.

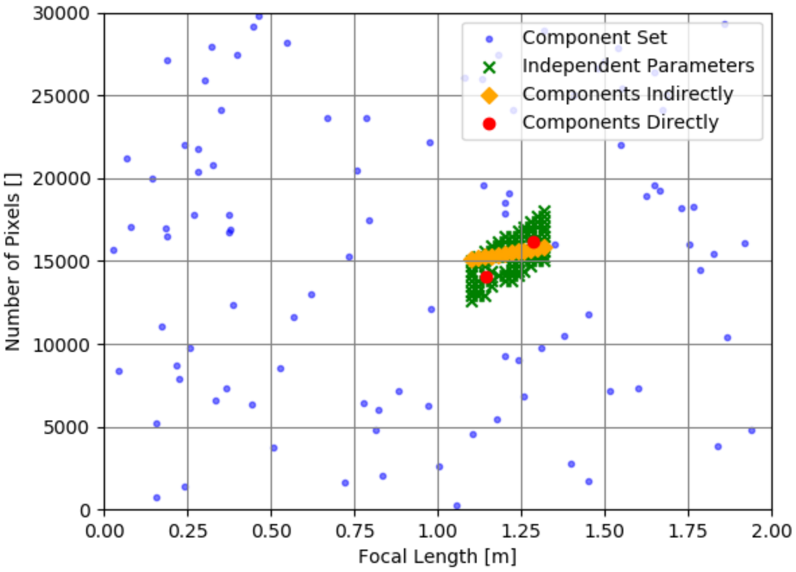


Figure 4.8: Identified feasible parameter sets for every state space option

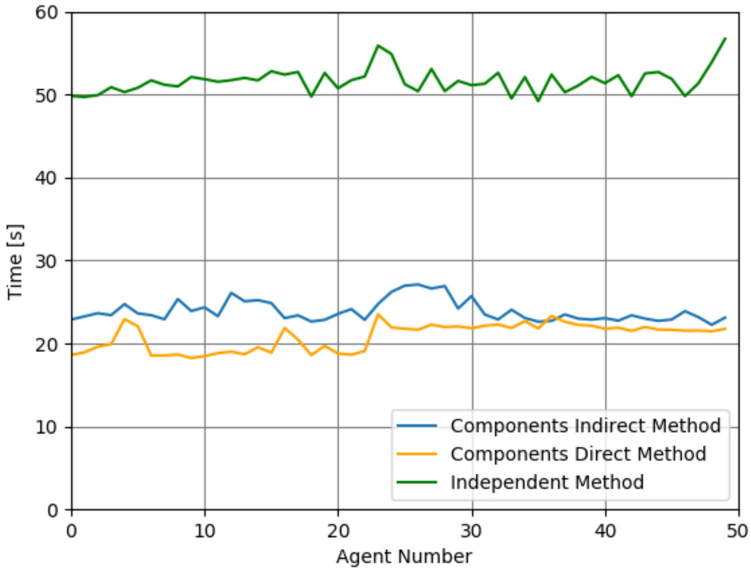


Figure 4.9: Evaluation time for each training cycle for every state space option

4.2.4. State Space Option Selection

In the setup of the design tool that is used for these experiments, the setup using state space option 3, the independent parameters state space, already has a limited performance due to the size of the state-action space that is generated by the parameters. For the final setup, this limitation will only be exacerbated since more component types will be implemented. The selected state space option for the final setup is therefore either state space option 1 or 2. At the moment, state space 1 does not have a better performance than state space 2. Therefore, for future versions of the tool, state space 1 will be used since this removes the need for generating the fit and matching components to the obtained results. These additions would impose extra computational expense while not improving the performance of the design tool. The reward system for this state space option should be reviewed and changed as such that the Q-learning algorithm is able to find solutions for any set of user requirements.

4.3. Follow Up Experiments

Previously in this section, experiments were performed using three distinct state space options. From these experiments, several of the observations that were made are treated by further experiments. The experiments that are performed will be explained and their results presented and discussed. The observations that are treated in this section are listed below.

- **Observation 1**: For state space option 3, described in Figure 4.1.3, it is observed that the number of iterations that is performed each episode will guarantee that for certain initialisation states no feasible solution will be found.

- **Observation 2**: For state space options 1 and 2, described in subsection 4.1.3 and Figure 4.1.3 respectively, it is observed that the reward system that was used for the experiments is not usable when no components or the fit passes through the feasible solutions region. A new reward system should be able to identify the ’best’ region for a set of user requirements.

4.3.1. Independent Parameters Approach’ Iterations

As was identified in section 4.2, the performance of state space option 3 is relatively low in both reward and time performance. This is due to the relatively large state-action space as compared to the other approaches. The size of this space makes it difficult for the agent to find a feasible solution in the allowed number of iterations. Using all iterations during an episode is the reason for both the relatively low reward and time performance.

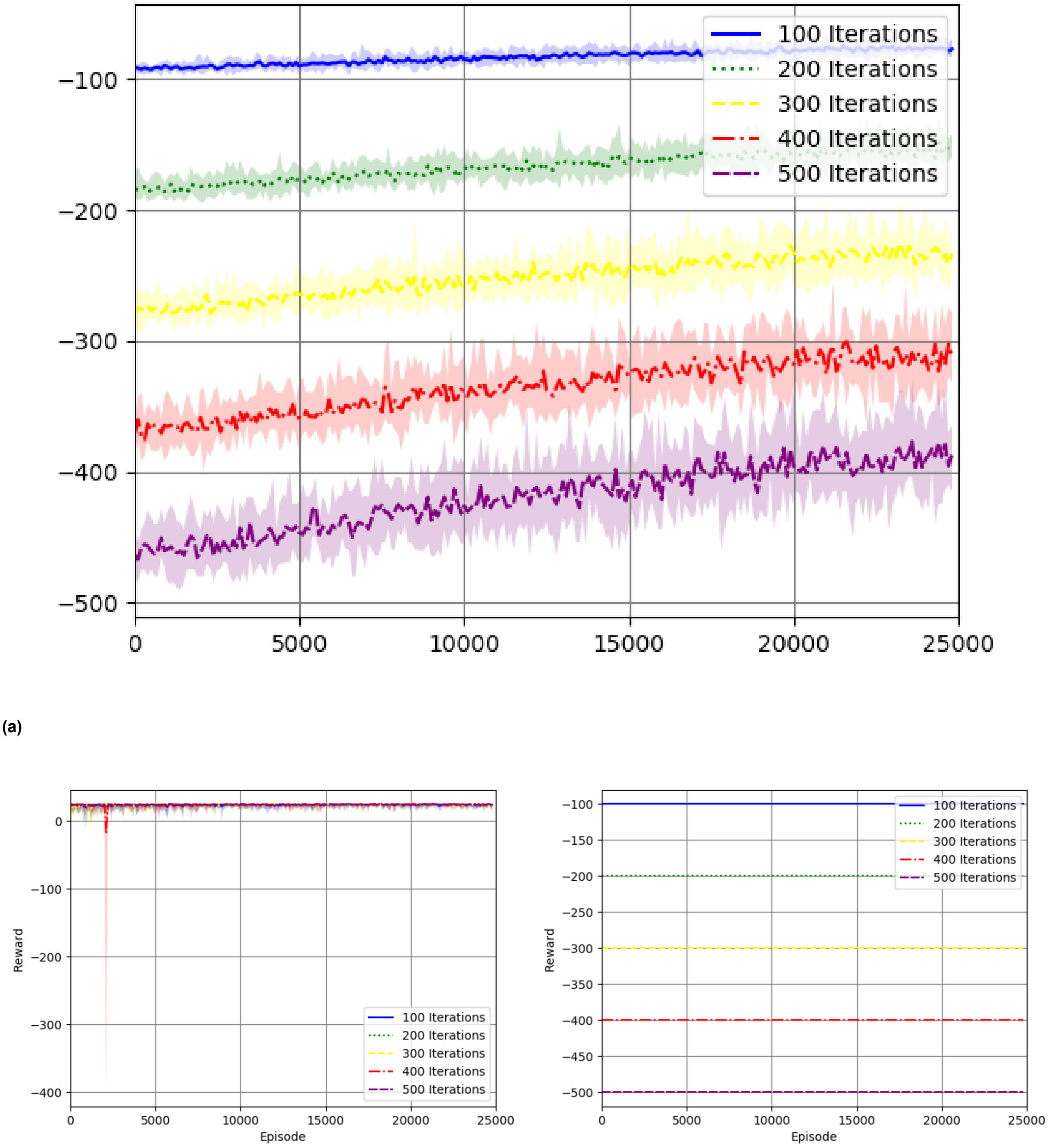
One way to increase the agents ability to find feasible solutions is to increase the number of iterations that can be performed during a single episode. An experiment is performed to research the influence of the amount of iterations for the agent’s performance. The results of these experiments are shown in Figure 4.10a to Figure 4.10c. In these figures, it can be seen that the performance behaviour is very similar between the different numbers of iterations. A clear increase in performance in reward behaviour is not evident in these graphs. For all iteration quantities, the minimal reward is equal to the number of iterations and does not vary during a training cycle. The maximal reward that is obtained converges immediately towards the maximum possible reward for every iteration quantity. These behaviours are not surprising, because this means that in the allowed number of iterations, even when this increases, the agent is still not able to reach a feasible solution due to the size of the state-action space. When the number of iterations are increased greatly, the agent will most likely be able to ultimately find a solution, but this will come at the cost of time, as can be seen in Figure 4.11.

4.3.2. New Reward System

The reward system that is used for the experiments that were performed previously in this section relies on a reward that is given when the agent reaches a ’feasible’ region, indicated by a threshold value, and a moving penalty that is given otherwise. This moving penalty can be given every iteration, so the minimal reward that can be obtained is the amount of iterations times this moving penalty.

For state space options 1 and 2, the ability of the agent to reach this ’feasible’ region depends on the components’ parameters and the path of the fitted line. If these do not intersect, the agent will not be able to find feasible solutions using this reward system. For the experiments that were done previously it was assured that this was not the case. A different reward system should be used in order to allow the approaches to cope with unknown user requirements and environments.

For this problem several continuous reward systems have been experimented with. The reward systems are based on the distance of the iteration parameters to the required parameters. This way, a reward is given every iteration based on the performance of the state the iteration finds itself in. The functions of the investigated reward systems, a solely positive and a solely negative reward system, are shown below and illustrated in Figure 4.12. These reward systems are applied to state space option 1. It is expected that for agent training purposes there should be no difference between using a negative or positive reward system.



(a)

(b)

(c)

Figure 4.10: Rewards as a function of episode number for the 'Independent Parameters' approach to the basic scenario for different numbers of iterations with (a) Average reward with variation. (b) Maximal reward with variation. (c) Minimal reward with variation.

$$R = -\frac{1}{D_{max}} \cdot D + R_{max} \tag{4.6}$$

$$R = -\frac{1}{D_{max}} \cdot D \tag{4.7}$$

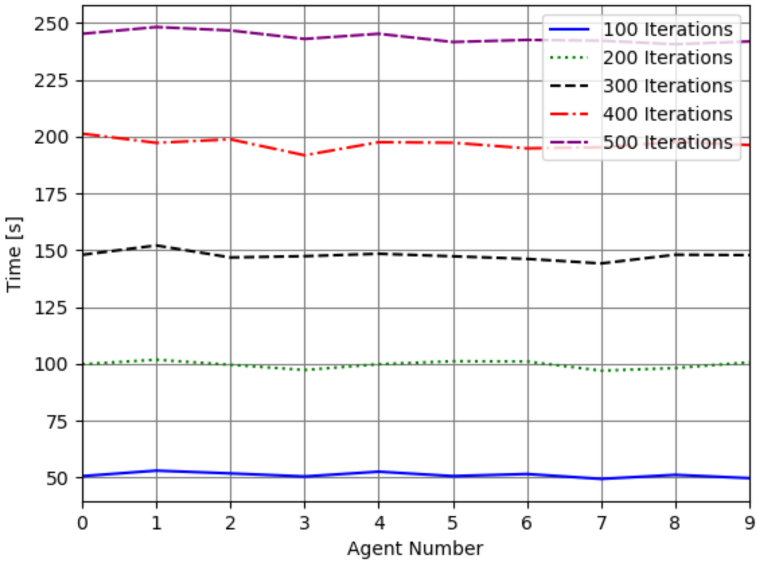


Figure 4.11: Training cycle time in seconds for different iteration quantities

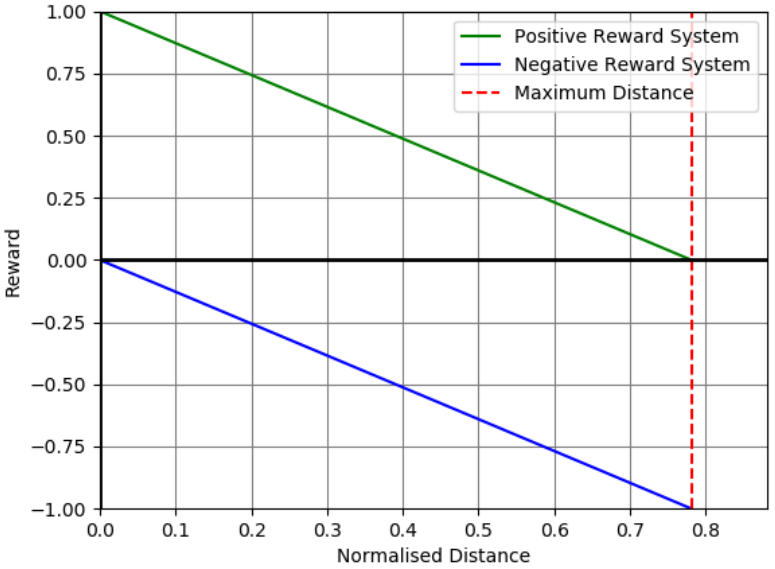


Figure 4.12: Reward functions as a function of normalised distance

Results

The reward systems agent performance can be visualised in several ways. First, there it is possible to represent the performance in a similar manner as in section 4.2. Secondly, it is possible to use the final iteration reward of every episode to represent the performance of the agents. Because a continuous reward system is used, the final iteration reward represents the last solution that the agent has found in that episode. During a training cycle, it is expected that this reward will increase as more training is done since the agent will try to find the 'best' solution. After reviewing both ways of representing the results, it is decided to use the final iteration reward. From this representation, observations are made that are not possible when using the episode performance. Next to that, it is observed that the behaviour of the final iteration reward during a training cycle is similar to the episode rewards.

The results for the positive and negative reward system are shown in Figure 4.13a to Figure 4.13e.

Discussion of the Results
The average final iteration reward and its variation for both reward systems can be seen in Figure 4.13a. In the graph, it can be seen that the negative reward system seems to learn faster, which is shown by the steeper increase in average reward in the first 10000 episodes, but converges to the same relative value. This is displayed as well in the maximal reward graphs in Figure 4.13d and Figure 4.13e.

Influence of Discount Factor
An interesting behaviour is seen in the minimal final iteration reward graphs for both reward systems, which can be seen in Figure 4.13b and Figure 4.13c. The minimal rewards for both reward systems appear to have an upper limit. For the positive reward system, this is around 0.6, and for the negative reward system, this limit is around -0.4. The agents however, are able to find more optimal solutions, which can be seen in the maximal reward graphs in Figure 4.13d and Figure 4.13e. For this scenario, the agent is always able to achieve any state due too the combination in state space size and number of possible actions. It is therefore not expected that the minimal rewards converge to a value that is lower than the maximal rewards. In other words, it is not expected that the agent ends up in a sub-optimal position at the end of the training cycle. The minimal reward graphs, in Figure 4.13b and Figure 4.13c, therefore indicate that the agent gets stuck in local optimal points.

This is confirmed in Figure 4.14, which shows the amount of times the agent ends up in a certain state at the end of an episode during a training cycle, which consists of a total of 25000 episodes, using the positive reward system. The discount factor controls the importance of future rewards for the agent, where a value of 1 will influence the agent to strive for long-term rewards, and a value of 0 will influence the agent to strive for short-term rewards. Increasing this value might pull the agent out of the local optimal points. In Figure 4.15, the results of increasing the discount factor from 0.90 to 0.95 can be seen. In this graph, it can be observed in stead of having 4 prevalent final states, this number is brought down to 2, which are also the highest reward states and therefore the optimal points.

The change in performance of an agent using a positive reward system and a discount factor of 0.90 as compared to an agent using a factor of 0.95, which will be called experiment 1 and 2, can be seen in Figure 4.16a to Figure 4.16e. The minimal reward graph in Figure 4.16c is no longer limited by local optima and is therefore able to achieve higher values. Next to that, the maximal reward graph in Figure 4.16e shows a more stable behaviour. Both of these improvements lead to a higher average reward value, which can be seen in Figure 4.16a. It is concluded that raising the discount factor influences the agent to pull out of local optima in favour of higher optimal points, which increases the agent's performance for this scenario.

Influence of Epsilon Decay
From the graphs for the agent with a positive reward system and a discount factor of 0.95, in Figure 4.16a, Figure 4.16c and Figure 4.16e, it can be deduced that the agent is able to find the most optimal solutions relatively quickly. The maximal reward converges to the feasible upper limit after 5000 episodes. Another parameter that influences the learning of the agent by determining whether an action is taken at random or determined by the Q-table, is the epsilon during the training cycle. The epsilon decay that is currently being used can be seen in Figure 4.1. Because the epsilon has a minimal value of 0.1, it is ensured that random actions are still taken at the end of the training cycle. This is not necessarily beneficial for the agent, because these actions might take it away from an optimal solution. Although this is beneficial for exploration purposes, depending on the size of the state-action space it can be more desirable to favour exploitation sooner in the training cycle. It is therefore investigated whether using a different epsilon decay function could be favorable for the agent's performance. The minimal epsilon value is removed and the epsilon is therefore able to decay to near zero. Next to that the epsilon decay value is changed from 0.9999 to 0.9997, which are referred to experiment 2 and 3. Note that both experiments still use a discount factor of 0.95, since this has already proved to increase the agent's performance. The results of experiment 2 are therefore the same results as the previous experiment 2. The comparison of the results can be seen in Figure 4.17a to Figure 4.17e. The difference in epsilon decay between the experiments can be seen in Figure 4.18.

Comparing the results of experiments 2 and 3 in Figure 4.17a to Figure 4.17e, it can be seen that the change in epsilon parameters has improved the performance of the agent. A steeper reward curve can be seen in every figure for experiment 3 when compared to experiment 2. This indicates that the change in epsilon has resulted in an increased rate of learning for the agent and requires less episodes in a single training cycle to converge to similar results, thereby reducing the time that is required to train the agent. It is thought that this is the result of earlier preference for exploitation in stead of exploration, due to the faster decay of the epsilon. This means that the agents tries to find the best solution sooner, thereby increasing the rewards that it will obtain throughout an episode.

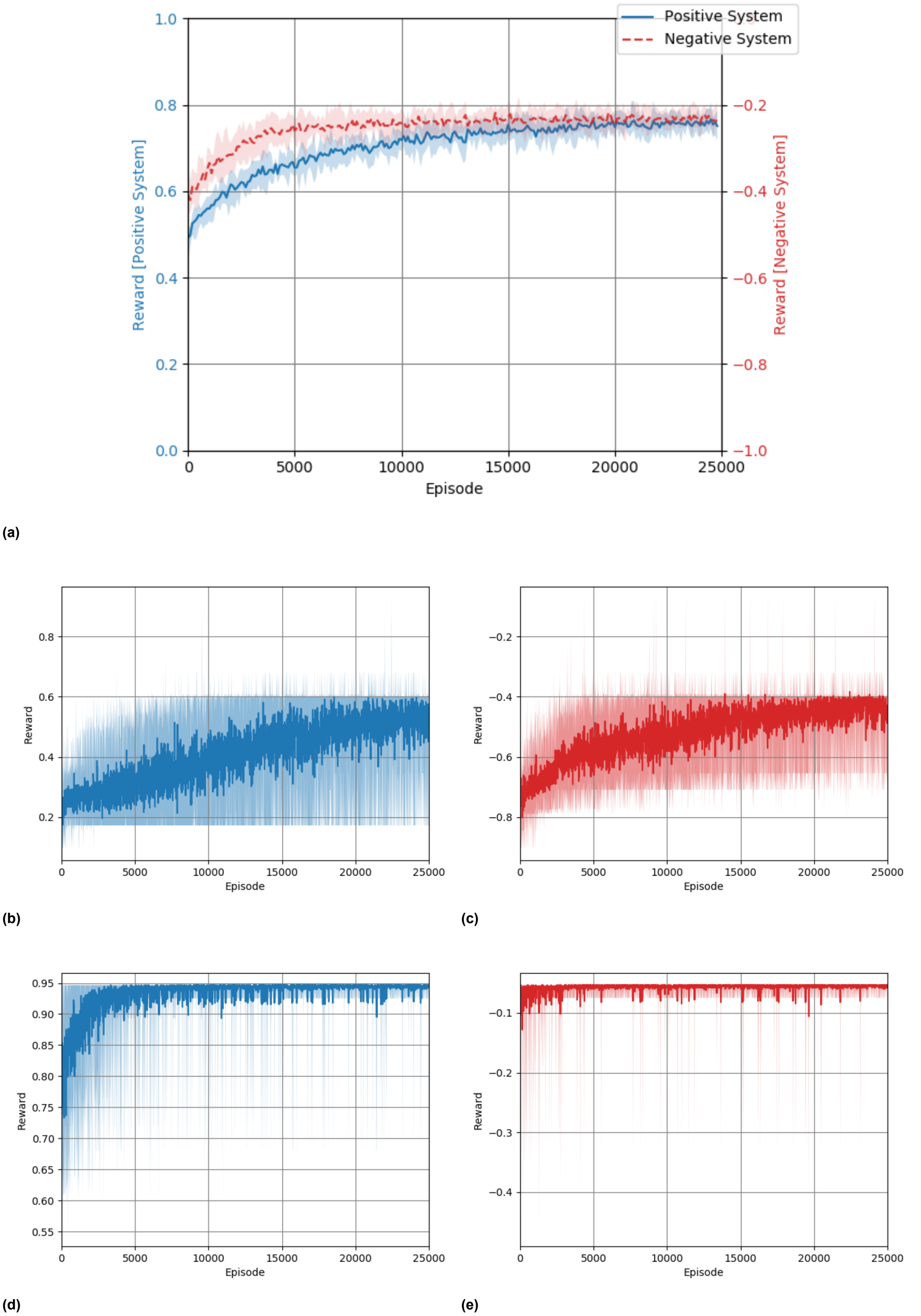
Conclusion of the follow-up experiments
Conclusions of the investigation into the effect of several agent parameters on the agent's performance for state space option 1 are made in this section. At the moment the slight increased learning rate of the negative reward system as compared to the positive system is thought to be caused by the same initialisation values for the Q-tables of both systems. The Q-table for both systems is initialised with values ranging randomly from -10 to 0. Because the Q-values in the Q-table determine the actions that the agent takes,7 and the Q-values depend on the obtained rewards, this would explain a difference in learning behaviour between the two reward systems. This is confirmed in Figure 4.19, where a positive and negative reward system for state space option 1, whose Q-tables are initialised similarly, are shown. The agent's Q-table that is using the positive reward system is initialised with random values between 0 and 1, while the Q-table for the negative reward system is initialised with random values between -1 and 0. The ranges for both reward systems coincide with the possible rewards, and are therefore similar between the two systems. In Figure 4.19, it can be seen that this results in a similar performance for both systems. It can therefore be concluded that the two different reward systems do not yield different performance results.

For the positive reward system, it can be concluded that the changes made for the discount factor and epsilon decay has resulted in increased agent performance for state space option 1, the 'Components Directly' approach. The same increase in performance is expected when these changes would be applied to the negative system. This does not necessarily mean however, that changing these parameters for state space option 2, the 'Components Indirectly' approach, will result in the same increase in performance.

Implications for state space option 2
The increase in performance due to the increase in discount factor for the agent of state space option 1 can be largely accredited to the highly variable reward line in the states of state space option 1, which can be seen in Figure 4.14. For state space option 2, because a fit has been created through the components this reward line does not lead the agent to get stuck in local optima, which can be seen in Figure 4.20. It is therefore not necessary to change the discount factor for state space option 2 when using a continuous positive reward system. Exploiting sooner by increasing the epsilon decay leads to a similar increase in performance as is observed for state space option 1. This is illustrated in Figure 4.21a to Figure 4.21e. Note that an agent of state space option 2 using a positive reward system has a better performance than state space option 1 using the 'traditional' agent settings. It is thought that this is caused by the more steady transition between states for state space option 2, where an action will only cause a slight increase or decrease in reward value, where for state space option 1 this is highly variable. This makes it more difficult to assess the long-term reward for the agent in state space option 1.

Further Implementation of the Continuous Reward System
At the moment, most experiments have been performed with the positive reward system. Using the same relative Q-table initialisation it is shown that no difference between the two reward systems exist. It is therefore decided that for further experiments on scenarios with increased complexity the positive reward system will be used. The original goal of providing the agent with the capability of coping with unknown user requirements and environments is met using such a positive reward system. This reward system is able to provide rewards to the agent regardless of the location of the components or the fitted line in the design space. This means that the closeness of any point in the state space to the user requirements can be determined, and thereby the feasibility of this point.

Unexpected behaviours of the reward systems have been investigated and treated by changing several of the agent's parameters, namely the discount factor and the epsilon decay. These changes provided increased performance for state space option 1, while for state space option 2 only the change in epsilon decay provided an increased performance. The lessons that were learned from these experiments will be used for the final setup of the design tool.



(a)

(b)

(c)

(d)

(e)

Figure 4.13: Final iteration rewards as a function of episode number for the continuous reward systems for state space option 1 with (a) Average reward with variation. (b) Minimal reward with variation for the positive reward system. (c) Minimal reward with variation for the negative reward system. (d) Maximal reward with variation for the positive reward system. (e) Maximal reward with variation for the negative reward system.

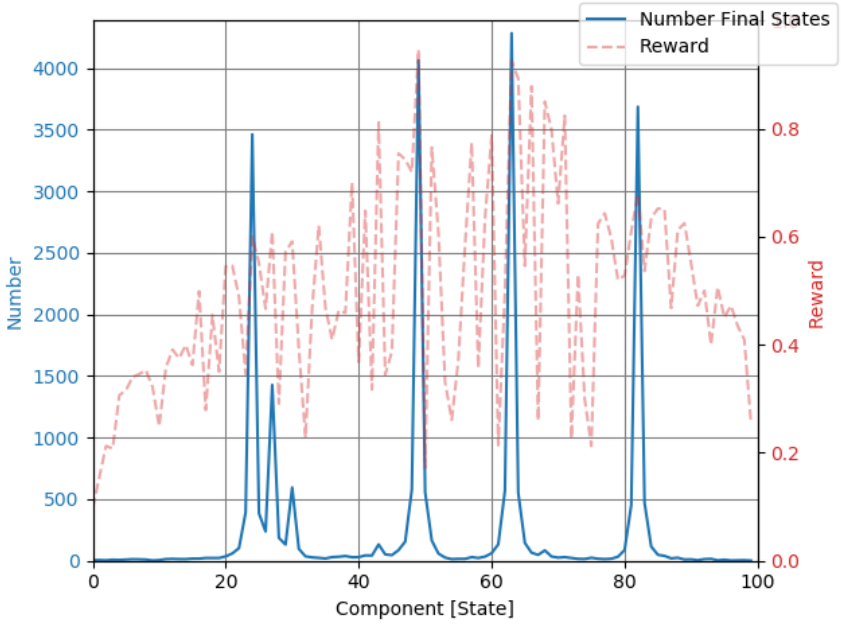


Figure 4.14: Amount of times the agent ends up in a state for state space option 1 with a discount factor of 0.90

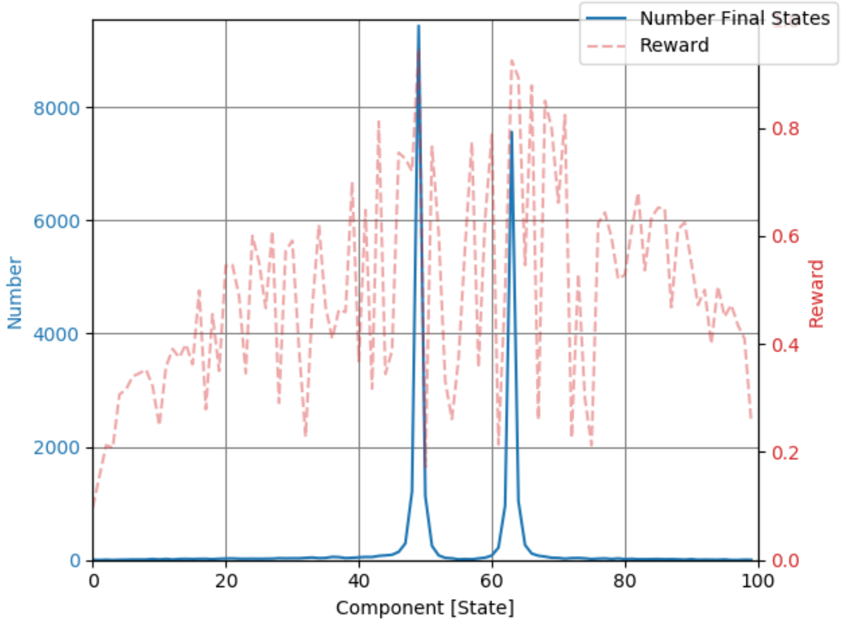


Figure 4.15: Amount of times the agent ends up in a state for state space option 1 with a discount factor of 0.95

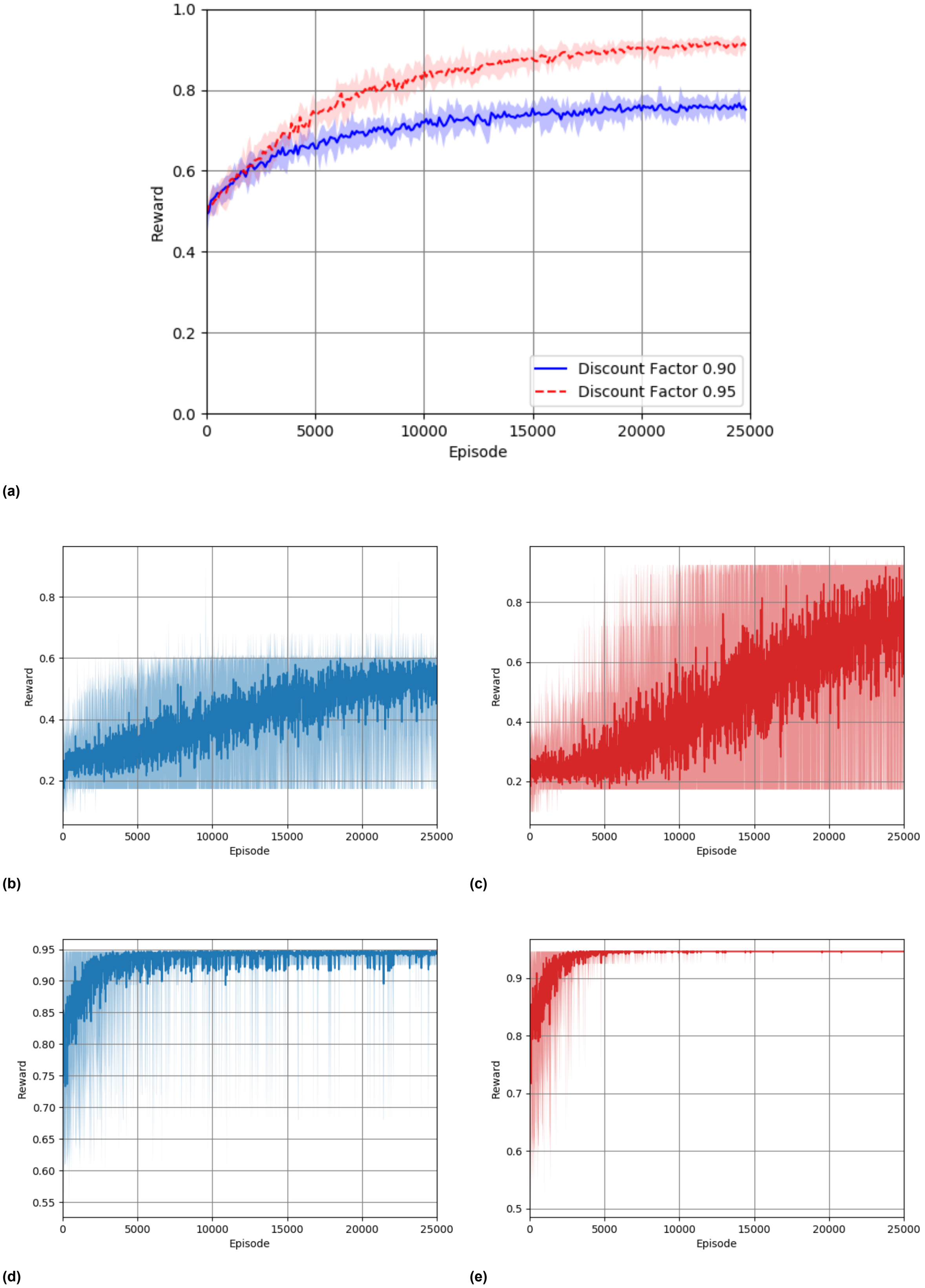
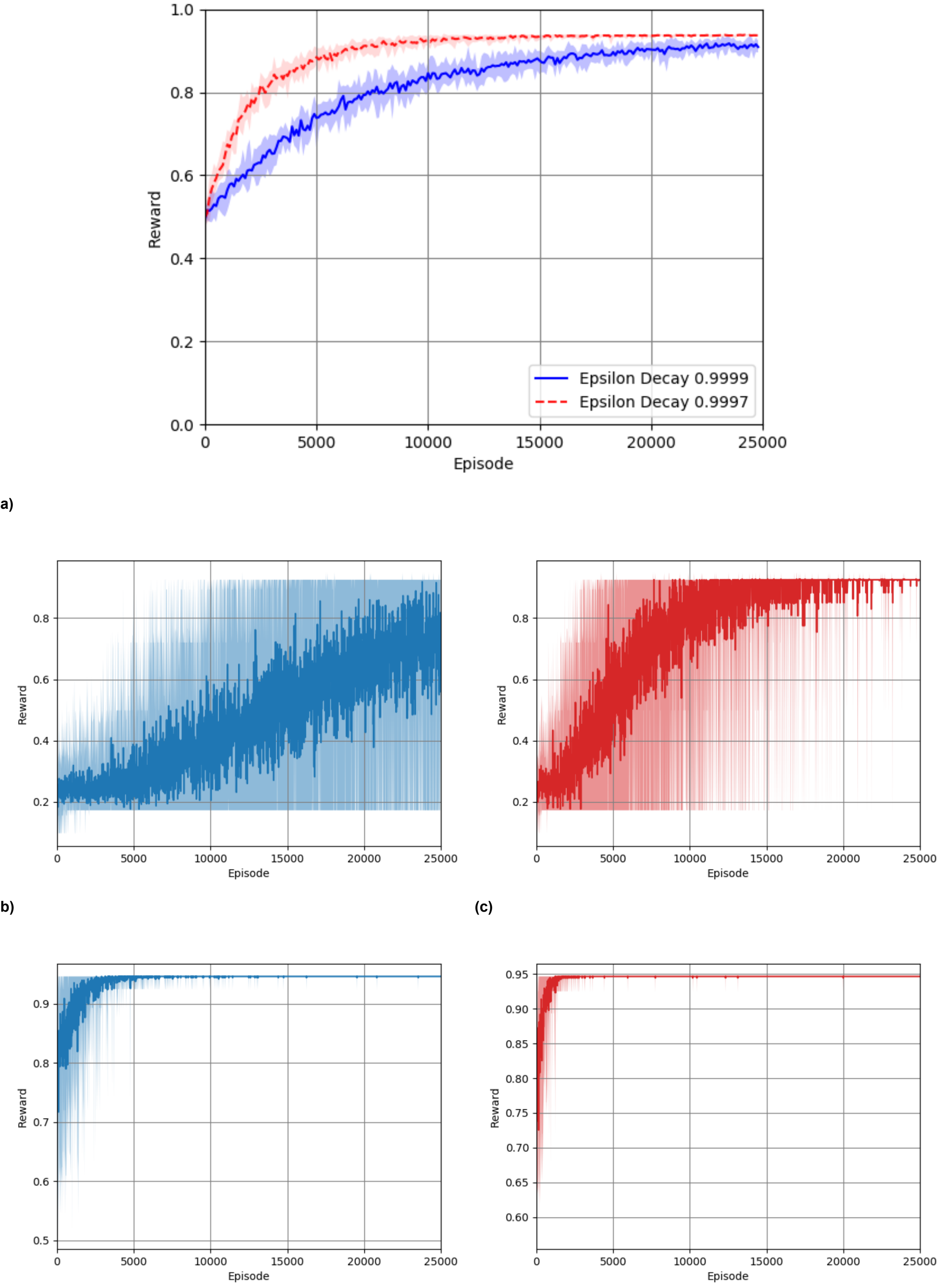


Figure 4.16: Final iteration rewards as a function of episode number for the positive reward systems with discount factors 0.90 and 0.95, respectively, for state space option 1 with (a) Average reward with variation. (b) Minimal reward with variation for experiment 1. (c) Minimal reward with variation for experiment 2. (d) Maximal reward with variation for experiment 1. (e) Maximal reward with variation for experiment 2.



(a)

(b)

(c)

(d)

(e)

Figure 4.17: Final iteration rewards as a function of episode number for the positive reward systems with epsilon decay values of 0.9999 and 0.9997, called experiment 2 and 3 respectively, for state space option 1 with (a) Average reward with variation. (b) Minimal reward with variation for experiment 2. (c) Minimal reward with variation for experiment 3. (d) Maximal reward with variation for experiment 2. (e) Maximal reward with variation for experiment 3.

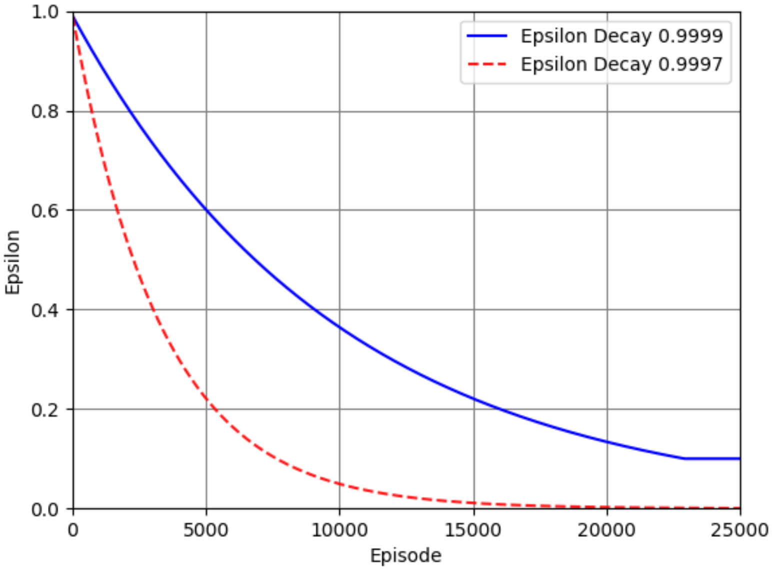


Figure 4.18: Epsilon decay as function of episode for experiments 2 and 3

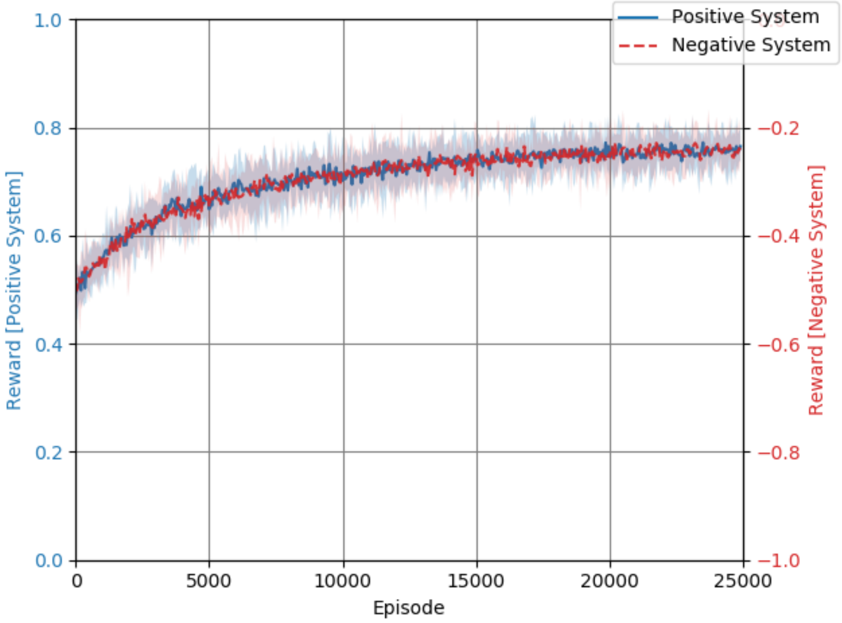


Figure 4.19: Average final iteration reward with variation as a function of episode for a positive and negative reward system with similar Q-table initiation values

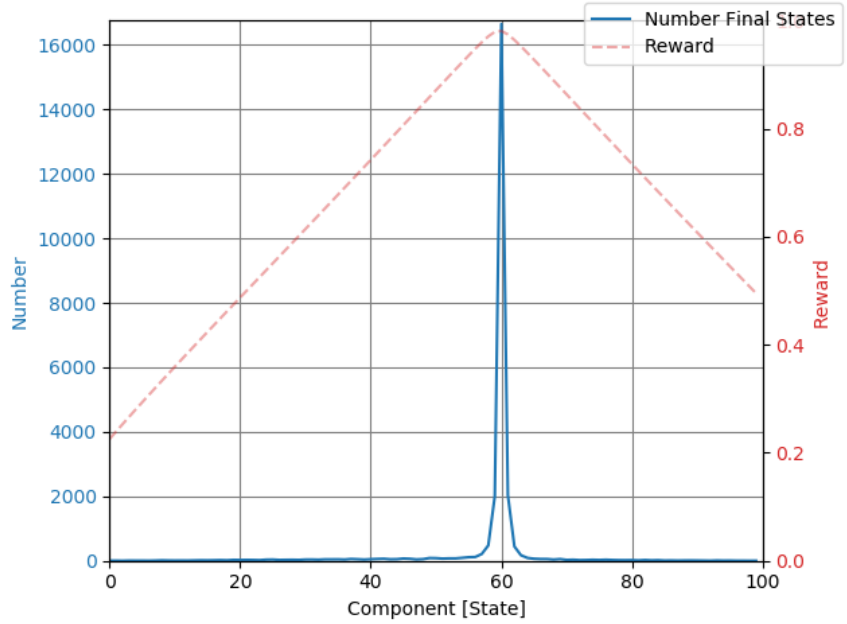
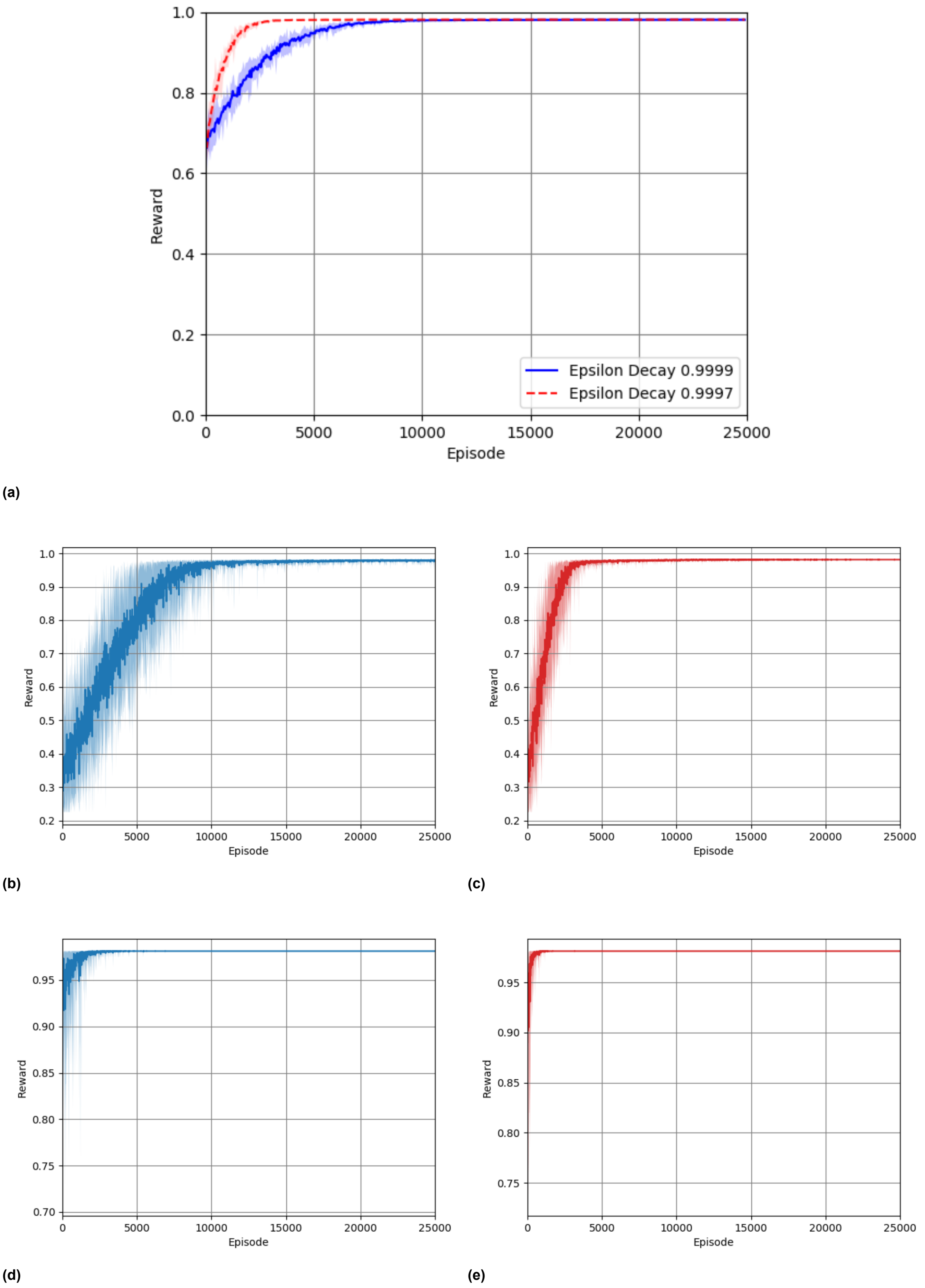


Figure 4.20: Amount of times the agent ends up in a state for state space option 2 with a positive reward system



(a)

(b)

(c)

(d)

(e)

Figure 4.21: Final iteration rewards as a function of episode number for the positive reward systems with epsilon decay values of 0.9999 and 0.9997, called experiment 4 and 5 respectively, for state space option 2 with (a) Average reward with variation. (b) Minimal reward with variation for experiment 4. (c) Minimal reward with variation for experiment 5. (d) Maximal reward with variation for experiment 4. (e) Maximal reward with variation for experiment 5.

5

Design Tool Final Setup

In this chapter, the final setup of the design tool is constructed. The complete design tool will be able to create a configuration of components. For each component type of this configuration, it is explained how this type is implemented into the design tool. Before this is defined, the general settings of the final setup will be elaborated upon. These settings are based on the results of the experiments of chapter 4.

Nomenclature From this point, the words CubeSat system design, satellite, concept, component set, state or a similar variation can be seen as synonymous and are used interchangeably. The expressions subsystems and component type(s) are also used interchangeably.

5.1. Concept Generation Method

In chapter 4, a design tool was created in a relatively simple setup in order to perform experiments with using different kinds of state space options for the Q-learning algorithm. In this setup, a single component type was used, a camera, for which the design tool had to select the best possible option for user entered requirements from a randomly created components database. The state space option that was selected for further use in the final setup of the design tool was the 'Component Directly' state space option. The algorithm should be able to select components directly from a components database and assess the performance of these components for user entered requirements. The goal of the design tool in its final setup is to automate the generation of CubeSat concepts. To present a more complete concept to the user, more component types will have to be included in the design tool. No longer will the design tool only select a single component type for specific user requirements, it will now select multiple component types for multiple requirements that are co-depending. An example of this could be the size of the concept which has an effect on the attitude control requirements. The size in turn is influenced by the size of the components that are selected by the Q-learning agent. It can be seen that the design tool will also have to coordinate the selection of components at a system level. In this section, the method behind the generation of concepts in the final setup of the design tool is explained.

5.1.1. Design Tool Overview

The design tool makes use of the Q-learning algorithm that is able to select components based on an action that is either randomly taken or based on the Q-table. This process is similar to the process in chapter 4, where the design tool was already able to select a camera component from a component database. The difference between that version of the design tool and the final version is that now the design tool is able to select components for multiple component types. The algorithm of the design tool can be seen in pseudocode in Algorithm 2.

In this pseudocode, the general elements of a Q-learning algorithm are indicated in bold, and the items that are specific for CubeSat concept generation using Q-learning (CCG-Q), are indicated in italic.

During an episode, the agent tries to take actions to maximise the reward that it can obtain. The number of actions that the agent can take during an episode is dictated by the number of iterations. At every iteration the design tool selects a component for every component type. This component set is the state. The length of the state $[C_1, C_2, ..., C_n]$ depends on the number of component types that the algorithm can select components for. The state thereby represents a CubeSat concept, insofar that the concept exists of the component types that are available to the agent in the environment. For this state, the performance of the CubeSat concept is evaluated against the user entered mission requirements, which is the input to the design tool. Based on this performance the design tool rewards the state for subsystem performance, which are subsequently combined into the system reward. The settings of the algorithm that are used are based on the results from chapter 4. The settings can be seen in Table 5.1.

Algorithm 2 CubeSat Concept Generation using Q-learning (CCG-Q)

```
CCG-Q: Obtain user entered mission requirements
Initialise $Q(S, A)$
Initialise $\epsilon$

for each episode do:
    Initialise state $S$
    CCG-Q: Initialise state with set of components $S = [C_1, C_2, ..., C_n]$
    for each iteration of episode do:
        Observe state $S$
        Choose action $A$
        CCG-Q: Action based on $Q(S, A)$ or random action
        CCG-Q: Determine CubeSat dimensions from state $S$
        CCG-Q: Determine CubeSat mass from state $S$
        Take action $A$
        CCG-Q: Action changes set of components $S$ to $S'$
        Determine state performance
        CCG-Q: Analyse subsystem performance of state $S$ for
        mission requirements
        CCG-Q: Obtain sub-rewards $[R_1, R_2, ..., R_n]$
        Determine reward $R$
        CCG-Q: Determine system reward $R_{system}$
        Observe reward and state $R, S'$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$
        Update $Q(S, A)$
    end for
    Decay $\epsilon$
end for
```

Table 5.1: Design tool final setup Q-learning agent settings

Setting	Value
Number of Episodes	25000
Number of Iterations	100
Epsilon Initial	0.99
Epsilon Decay	0.9997
Epsilon Minimal	0.1
Learning Rate	0.3
Discount Factor	0.95

Design Tool Schematic

The schematic of the design tool is detailed in the diagram in Figure 5.1. The items that are surrounded by a dotted box are further detailed in Figure 5.2. These diagrams present a schematic overview of the system coordination of the design tool.

The diagram of Figure 5.1 is basically Algorithm 2 in diagram format. The smaller darker line indicates the route which the design tool follows, while the larger brighter line indicates the in- and output connections between the processes of the design tool. The input to the design tool is the mission requirements. After these have been entered by the user the design tool generates a Q-table, either through random initialisation or through loading a previously trained Q-table. The design tool cycle is then initiated. Here the number of cycles depends on the number of episodes and the iterations per episode. Every episode the agent is allowed to take a certain number of actions, determined by the number of iterations. At every episode, the state $[C_1, C_2, ..., C_n]$ is initialised randomly. The state indicates the components that are selected and can therefore be seen as the CubeSat concept. So at every episode, the agent starts off with a random CubeSat concept. At every iteration, the agent takes an action which changes the state. This concept is the input for the concept evaluation where the performance of the selected concept is determined for the user entered mission requirements. A reward is returned by the concept evaluation which is used to calculate the Q-value for the state-action combination and with which the Q-table is updated. The agent then takes an action again, completing an iteration. The actions that the agent takes are determined by either the Q-table, where the agent would take an action that maximises the reward that will be obtained, or a random action is taken to promote exploration of the design space. The cycle can be explained as trying to map a maze and learn which direction to take at a crossroads. When the agent is taking actions purely based on the Q-table, it will take actions that have the result of retrieving the best rewards. In this way, the agent should end up with the best performing CubeSat concept for the mission requirements. The output of the design tool is obtained by storing the concept performance that is evaluated at every iteration. Hereby, after processing this data, it can be identified what is the best performing concept based on the rewards that were returned for the concepts. How many times specific components are selected for every component type over an entire design tool cycle is also stored.

In the diagram of Figure 5.1, the lines between the boxes have the same definition as before. In this diagram, the concept evaluation part of the design tool is divided into the different subsystem analyses that are performed. The user entered mission requirements provide specific input for each subsystem/component type(s) analyses, where the performance of the components is determined. An individual reward is returned for each subsystem which are combined into the system reward. The environment, input to the design tool, state, actions, reward system and the state dimensions analysis will be further elaborated upon after this section.

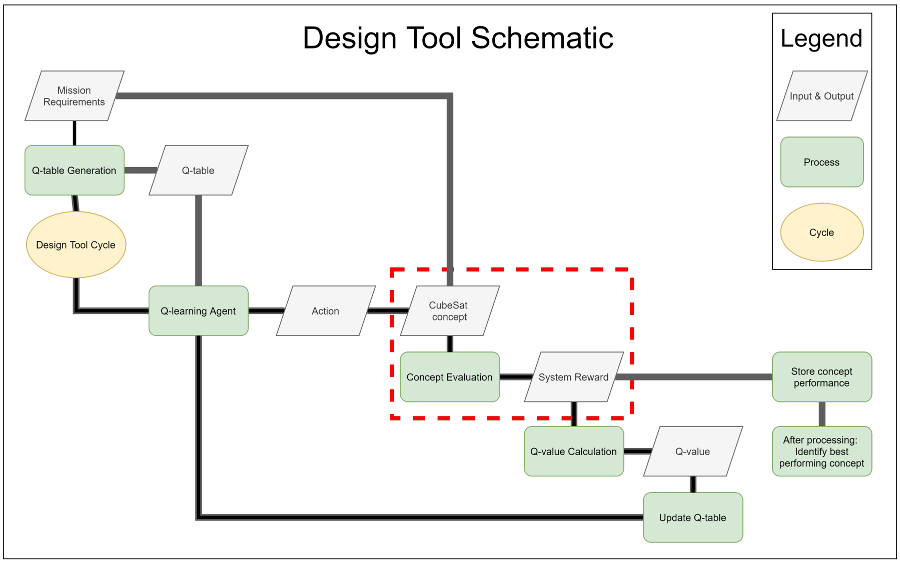


Figure 5.1: Schematic of the design tool

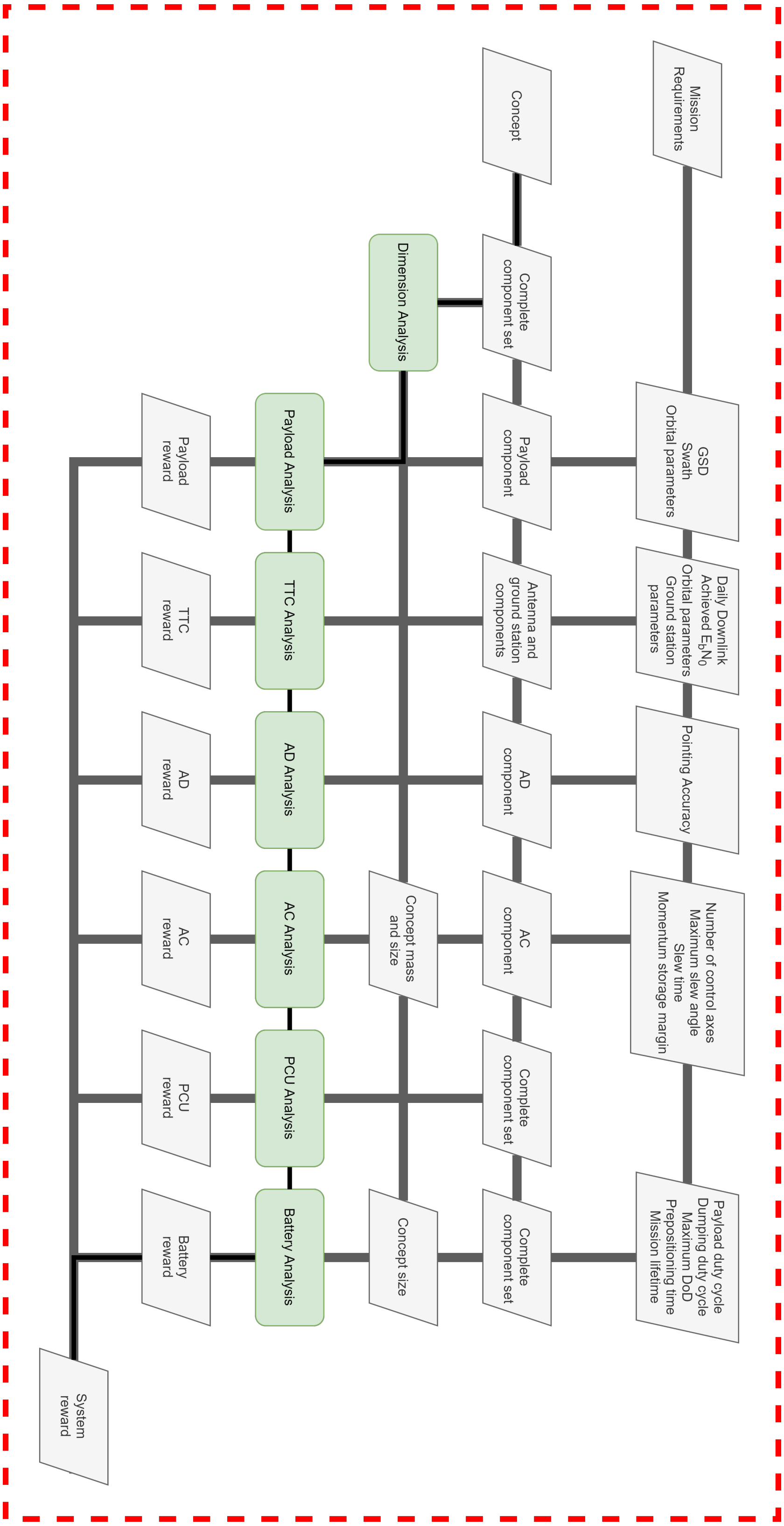


Figure 5.2: Schematic of the concept analysis section of the design tool

5.1.2. Environment

In chapter 4, it was concluded that for future versions of the tool, state space option 1 of the experiments should be used. This means that the components are used directly in the design tool. In the final setup of the tool, 8 different types of components are used. The component types are; camera, ground station, antenna, transceiver, attitude determination, attitude control, power control unit and battery. These component types make up the environment of the design tool, and the design tool will be able to select a component for each type to create a satellite configuration.

The design space of the design tool is therefore every possible combination of components that the agent can select. The components are entered into a component database by providing several specifications of each type of component. The specifications that are used will be indicated in the relevant section in this chapter. The components that are used for generating the results for this thesis have been selected as such that they provide a decent range in the relevant specifications. A set of common specifications exists for every component in the database, which can be seen in Table 5.2. The type of specifications that are specific for each component type will be given in the relevant sections. Furthermore, the exact specifications of the components in the database can be found in Appendix A.

Table 5.2: General component specifications

Component Specification	Unit
Power consumption	W
Voltage	V
Mass	kg
Size	m

For each component type, a distinct number of discrete components are used. The number varies between the different types, but is aimed to be at least 3. Note that with an increase in number of components per component type, the design space increases exponentially. A large number of components is not possible for the Q-learning method because of RAM memory limitations. The number of around 3 components per type was found after tests of the tool with varying numbers. This number per type allows for a similar number of components per type.

5.1.3. Input

The design tool should select a configuration of components for certain requirements. For these requirements, the performance of the configuration can be determined. There is a set of requirements that can be entered by the user of the design tool. This set of requirements determines the mission profile and the system and subsystem requirements. For each component type, specific requirements and parameters should be entered which are specified in the relevant sections section 5.2 to section 5.7. The mission profile is determined by entering the orbital parameters of the mission for which the design tool should identify the optimal configuration of components. The orbital parameters that should be specified by the user are listed in Table 5.3.

Table 5.3: User entered orbital parameters for the design tool

Parameter	Unit
Orbit Altitude	km
Inclination	degrees
RAAN	degrees

5.1.4. State

The state of the design tool in the final setup is the set of components that the agent has selected. The state is simply a list of length equal to the number of component types. Each index in this list indicates what component is selected for every component type, with every type being a specific index of the list. The values that can be encountered in this list are determined by the number of component for each component type. For example, the state list $[1, 0, 2]$ would indicate a configuration of the second component of the first type, first component of the second type, and third component of the third type. The actions that the agent can take will change the values in this list and thereby change the state and change the selected components.

5.1.5. Actions

The actions that the agent can take is similar to the actions that the agent could take during the initial experiments. The choice is again either going up, down, or staying at the same location in a list. In this list, the location determines which component is selected by the agent. For each component type, 3 actions exist. Because the agent now has to select a multitude of components it can perform an action for each component type as well. The action space of the agent is then created by every possible combination of actions for each component type. This means that the number of actions is governed by Equation 5.1.

$$n_{actions} = 3^{n_{components}} \tag{5.1}$$

5.1.6. Reward System

The analysis of the configuration should provide a reward to the Q-learning algorithm. The reward should take the performance of the different component types into account. A reward is therefore established for every component type or component type set. The reward of the complete satellite configuration is the sum of these individual rewards. An example of this system is shown in Equation 5.2.

$$R_{system} = R_{pay} + R_{ttc} + R_{ad} + R_{ac} \tag{5.2}$$

In chapter 4, the results showed that using a continuous reward system improved the performance of the design tool. The component rewards that are used for the final setup of the design tool are generally obtained by using a semi-continuous reward system. The reward system per dimension is based on the performance of the components with respect to the user entered requirements that are present on the component type or component type set. A reward of 1 is returned to the agent when the component's performance meets the requirements. If the performance is below the required performance for the requirements, the reward that is obtained follows is a continuous straight line between 1 and 0. The point where a reward of 0 is obtained is generally dictated by the outermost data from the components database. For example, for the attitude determination this point is determined by the component with the least accurate attitude determination angle. The reward system for the attitude determination subsystem can be seen in Figure 5.3. This semi-continuous reward system is selected because it is desired that the design tool can construct the reward functions without having to perform an extensive analysis of the complete component set. If such an analysis would be required, it might be required to analyse every possible configuration of components because of the reward could depend on the performance, which in turn could depend on the complete concept. This would demand extensive computations in order to construct the reward functions, which is not desired. The minimum reward point of 0 is determined by a parameter that can be obtained easily. This point determines the slope of the continuous section of the reward function.

The only component type reward that does not follow this semi-continuous reward system is the reward that is given for the PCU performance. Because this system can either support the required voltage range or cannot, a reward of 1 or 0 is obtained. In this chapter, the reward system figures will be provided for every component type or component type set in the relevant sections.

5.1.7. Determining Performance

As explained in the previous section, the reward that is returned to the agent for the state which is currently selected is determined by estimating the performance of the state. The reward as used by the Q-learning algorithm is composed of the individual component type(s) rewards, which are based on their individual performance. The performance is estimated for the user entered requirements relevant to the component type. In section 5.2 to section 5.7, the implementation of the different component types are elaborated upon. The required input for every type will also be stated.

5.1.8. Concept Dimensions Analysis

The agent of the design tool selects a component for every component type and thereby selects a complete configuration of the satellite. For the analysis of some component types the size and mass

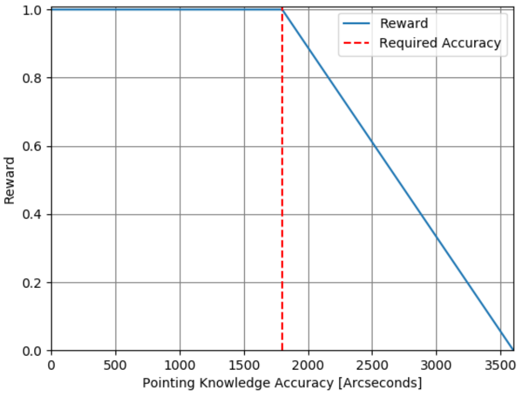


Figure 5.3: Reward system for the attitude determination component type

are inputs, which should therefore be estimated. In this section, the analysis that is performed to estimate the satellite size and mass is given.

In Table 5.2, it is stated that the size is one of the general component specifications that is known for each component in the database. The individual sizes of the components can then be used to estimate the complete size of the satellite. The size of a component that is obtained from the specifications is given in three-dimensional space, which is the length of each side. Most components for CubeSats are placed on PC104 boards, which allows the stacking of components inside the satellite. A PC104 board has a form factor of $90x96mm$ and therefore fits inside a single unit. For the size analysis, it is assumed that the direction of stacking the component is along z-axis. The x and y dimensions of the components are rounded up to complete units since it is assumed that all components are placed on PC104 boards which take up at least a complete unit. Because the components are stacked in the z-direction, the z-dimension of the components is used as an exact number. With the rounded x and y dimensions and the exact z-dimension the required volume in units is determined for each component. The required total volume of the satellite is then determined by summing up the individual component volumes.

Based on the required volume of the satellite, a satellite configuration is determined. If the volume of the satellite is below 3U, a 1x1x3U configuration is used. If the volume is between 3 and 6U, a 2x1x3U configuration is selected. If the volume is bigger than 6U, the configuration is determined through an automated method. The base on the xy-plane of the configuration is determined by rounding down the cube root of the required volume. This means that for for example a required volume of 12U, the configuration on the xy-plane would be 2x2U. The configuration in the z direction is then determined by adding a unit until the required volume can be satisfied by the configuration.

The final part of determining the size of the configuration is checking whether the z-dimension of the determined size complies to the maximum z-dimension of the components. If this is not the case, the z-dimension of the size is increased by a unit until the required dimensions can be accommodated.

The satellite's dimensions that result from this approach are used by other parts of the design tool that perform analyses for the different component types. The expected mass of the satellite is also estimated based on the determined size. The expected mass is determined using Equation 5.3.

$$m_{sat} = 2 \cdot size_{sat}(inU) \tag{5.3}$$

5.2. Payload Component

The payload that is used in the design tool is similar to the design tool of the experiment, an EO payload, and more specifically a camera. The analysis that is performed is similar to the analysis that was performed in the basic scenario in chapter 4. The analysis methodology is shortly revisited in subsection 5.2.1, and the implementation of the analysis into the tool is elaborated upon in subsection 5.2.2. Here, the reward system is also shown.

5.2.1. Payload Analysis

The component type for the payload subsystem is a camera component. The requirements that are provided by the user which determine the performance of the camera are listed in Table 5.4. The specifications of the components are provided in Table 5.5.

Table 5.4: Payload user requirements

Identifier	User Requirement	Unit
REQ-PAY-01	Ground Sample Distance	m
REQ-PAY-02	Swath	m

Table 5.5: Payload component specifications

Component Type	Specification	Unit
Camera	Focal Length	m
	Number of Pixels	-

The performance of the selected camera is evaluated by calculating the achieved GSD and swath with the camera that is selected by the agent. The calculations that are performed are described in section 4.1. The equations that are used are Equation 4.1 to Equation 4.2.

5.2.2. Implementation

To complete the implementation of the payload into the design tool, the performance that is returned from the analysis that was described in subsection 5.2.1 should be coupled to a reward. The reward is then returned to the agent which will influence the future actions that the agent takes. The pseudocode of the implementation of this part of the design tool is provided in Algorithm 3.

Algorithm 3 Payload Analysis

```
Determine required focal length and number of pixels
if Component focal length > Required focal length then:
    $R_f = 1$
else
    $R_f = \frac{1}{f_{req}} f_{comp}$
end if
if Component number of pixels > Required number of pixels then:
    $R_{nop} = 1$
else
    $R_{nop} = \frac{1}{nop_{req}} nop_{comp}$
end if
$R_{PAY} = \frac{R_f + R_{nop}}{2}$
```

In Figure 5.4 and Figure 5.5, the sub-rewards that are given for the focal length and number of pixels of the selected camera against the required values are shown. These sub-rewards are then combined into the payload reward, with both sub-rewards having equal weight in the final payload reward. The required focal length and number of pixels are determined for the entered orbit parameters and payload requirements in terms of GSD and swath. The payload components that can be selected by the agent can be found in section A.1.

5.3. Telemetry, Tracking & Command Components

The first component types that are included after the payload are of the Telemetry, Tracking and Command (TTC) subsystem. For this subsystem, to limit the complexity of the model, only the downlink

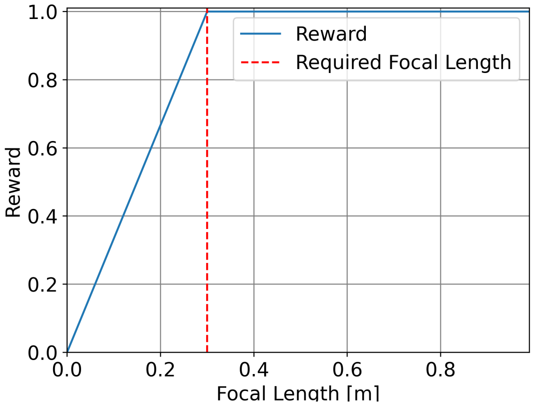


Figure 5.4: Reward system for the payload subsystem

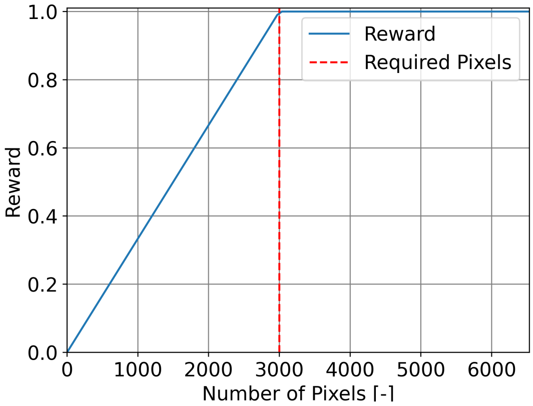


Figure 5.5: Reward system for the payload subsystem

is analysed and the components are also selected solely for downlink compatibility. The three components that are selected in this part of the tool are the antenna, transceiver and ground station. In subsection 5.3.1, the methodology behind the analysis of this part of the tool is explained, after which the implementation of this analysis into the tool is described in subsection 5.3.2.

- Number of components.
- List of components in appendix.

5.3.1. Telemetry, Tracking & Command Analysis

The three components that are selected by the tool for the TTC subsystem are the antenna, transceiver and ground station. These three options are included all at once because of the analysis depends on all three, and can not be performed without one of the three. The requirements on the TTC subsystem that are entered by the user of the design tool are listed in Table 5.6. Next, the parameters of the ground station that should be provided by the user are listed in Table 5.7. The specifications that are provided for each component type are shown in Table 5.8.

Table 5.6: Telemetry, Tracking & Control user requirements

Identifier	User Requirement	Unit
REQ-TTC-01	Daily Data Downlink	MB
REQ-TTC-02	Achieved $E_B N_0$	dB

Table 5.7: User entered ground station parameters

Parameter	Unit
Longitude	degrees
Latitude	degrees
Minimum Elevation	degrees
Simulated Orbits	-

Table 5.8: Telemetry, Tracking & Control component specifications

Component Type	Specification	Unit
Antenna	Frequency	MHz
	Antenna Gain	dB
	Beamwidth	degrees
Transceiver	Frequency	MHz
	RF Power	W
Ground Station	Frequency	MHz
	Ground Station Gain	dB
	Beamwidth	degrees
	Noise Figure	-

As stated previously, only the downlink is used as capability measure for the TTC components. To analyse the downlink, a link budget is established using the $E_B N_0$ method. The equations that are used within this method are obtained from [22]. The basic form of the link budget calculation can be seen in Equation 5.4 and in decibels in Equation 5.5.

$$\frac{E_b}{N_o} = \frac{P L_l G_t L_s L_a G_r}{k T_s R} \tag{5.4}$$

$$\begin{gathered}\frac{E_b}{N_o} = P + L_l + G_t + L_{pr} + L_s + L_a + G_r + 228.6 - 10logT_s - 10logR \\ P + L_l + G_t = EIRP \\ \frac{E_b}{N_o} = EIRP + L_{pr} + L_s + L_a + G_r + 228.6 - 10logT_s - 10logR\end{gathered} \tag{5.5}$$

In this equation, $\frac{E_b}{N_o}$ is the ratio of received energy per bit to noise density. P is the transmitter power, L_l is the transmitter to antenna line loss, G_t is the gain of the transmitter antenna, L_s is the free space loss, L_a is the transmission path loss and G_r is the gain of the receiver antenna. Below the denominator, parameter k is the Boltzmann's constant, which is $1.38064852 \cdot 10^{23} \frac{m^2 kg}{s^2 K^1}$, T_s is the system noise temperature and R is the data rate. As will become clear further on in this section, the only parameter that the link budget depends on after selecting components is the data rate. The required data rate is known from the user entered requirements and the capability of the selected components can therefore be measured by determining the achievable data rate and comparing this to the required data rate.

In the link budget equation, G_r and G_t are component inputs, k is a constant and the line loss L_l and transmission path loss L_a will be assumed. This leaves the free space loss L_s, pointing loss L_p, and the system noise temperature T_s as parameter which remain to be determined before the link budget can be calculated.

The free space loss L_s can be calculated by using Equation 5.6. λ is the wavelength in m and is determined using Equation 5.7. In this equation, c is the speed of light and f is the transmitter frequency. The distance between the satellite and the ground station is the slant range r_{slant}, which is determined using Equation 5.8. The slant range is dependent on orbit altitude r and minimum elevation angle ε, which is a user enter assumed value.

$$L_s = 10log\left(\frac{\lambda}{4\pi r_{slant}}\right)^2 \tag{5.6}$$

$$\lambda = \frac{c}{f} \tag{5.7}$$

$$r_{slant} = r_e\left(\sqrt{\frac{(r+r_e)^2}{r_e^2} - cos^2\varepsilon} - sin\varepsilon\right) \tag{5.8}$$

The pointing loss L_p is calculated using Equation 5.9. The pointing loss depends on the pointing error ϵ and the antenna half-power beamwidth θ. Because the pointing errors on both the ground station and in general on the satellite are close to zero, the pointing loss is not expected to contribute significantly to the losses in the link budget.

$$L_p = -12\left(\frac{\epsilon}{\theta}\right)^2 \tag{5.9}$$

The system noise temperature T_s is determined using Equation 5.10. The system noise temperature depends on the antenna noise temperature T_{ant}. Next to that, the reference temperature T_0, the line loss between the antenna and receiver L_r which are both assumed values and can be changed by the user of the tool. The noise figure F is calculated using Equation 5.11. In this equation, T_r is the noise temperature of the receiver.

$$T_s = T_{ant} + \left(\frac{T_0\,(1-L_r)}{L_r}\right) + \left(\frac{T_0\,(F-1)}{L_r}\right) \tag{5.10}$$

$$F = 1 + \frac{T_r}{T_0} \tag{5.11}$$

The assumed parameters in Equation 5.5 were stated to be the line loss L_l and transmission path loss L_a. The transmission path loss consists of several terms, namely the atmospheric loss L_{atmos}, the polarisation loss L_{pol}, the ionospheric loss L_{ion} and losses caused by rain in the transmission path L_{rain}. The assumed values for these losses can be seen in Table 5.9.

Table 5.9: Link budget assumed parameters' values

Loss	Assumed Value	Unit
L_l	0.5	dB
L_a	3	dB

Required Data Rate Estimation

To estimate the required data rate from the user entered daily data rate, the amount of time that the satellite is in view of a ground station should be determined. The ground station location is specified by the user as well. This set of parameters, together with the entered orbital parameters of the satellite, enables the calculation of the time in view of the satellite for the ground station.

The time in view is calculated by determining the elevation angle of the satellite with respect to the ground station. If the elevation angle is larger than the minimum elevation angle, the satellite is in view of the ground station and communications can be established. To determine the elevation angle of the satellite its position on the globe should be known. The equation with which the elevation angle of the satellite is determined is Equation 5.12 [31]. The Earth-satellite geometry on which these calculations are based can be seen in Figure 5.6.

$$sinE = \left[cos\phi - \frac{R_E}{r}\right]/(R/r) \tag{5.12}$$

$$(R/r) = \sqrt{1 + (R_E/r)^2 - 2(R_E/r)cos\phi} \tag{5.13}$$

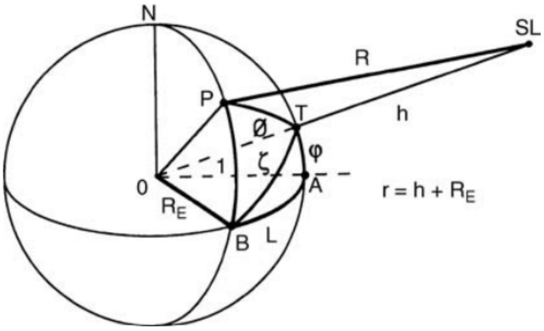


Figure 5.6: Visualisation of Earth-satellite geometry, from [31]

The unknown in Equation 5.12 is $cos\phi$, which is determined using Equation 5.14. This equation is a function of longitude and attitude positions of both the satellite and the ground station.

$$cos\phi = cosLcos\varphi cosl + sin\varphi sinl \tag{5.14}$$

For a selected number of orbits, the longitudinal and latitudinal position of the satellite on the Earth is established for every full degree in its orbit using Equation 5.15 and Equation 5.16. The parameters in these equations are visualised in Figure 5.7. A good estimate for the orbit period of a general CubeSat is 90 minutes, this would mean that the time between measurements is 15 seconds, which is seen as able to provide an accurate enough assessment of the time in view.

$$\lambda = \lambda_{SL} - \Delta\lambda \tag{5.15}$$

$$\varphi = arcsin\left[sinisin(\omega + v)\right] \tag{5.16}$$

Using the calculated elevation angles for the satellite with respect to the ground station, the average daily time in view can be calculated. When the satellite is in view is determined by Equation 5.17. From this, an average daily time in view is determined, which in turn is used to calculate the required average daily data rate in Bps with Equation 5.18. Note that this data rate is in Bps, to convert it to bps for the link budget calculations, we simply multiply by 8.

$$E \leq 90 - E_{min} \tag{5.17}$$

$$R = \frac{DATA_{daily_{req}}}{Time - in - view_{daily}} \tag{5.18}$$

5.3.2. Implementation

In the previous section, subsection 5.3.1, the analysis of the TTC subsystem was explained. This section will treat the implementation of this analysis into the Q-learning algorithm. To enable the algorithm to take the TTC performance into account, the performance influences the reward that is returned to the algorithm. The pseudocode for the implementation of the TTC into the design tool is shown in Algorithm 4.

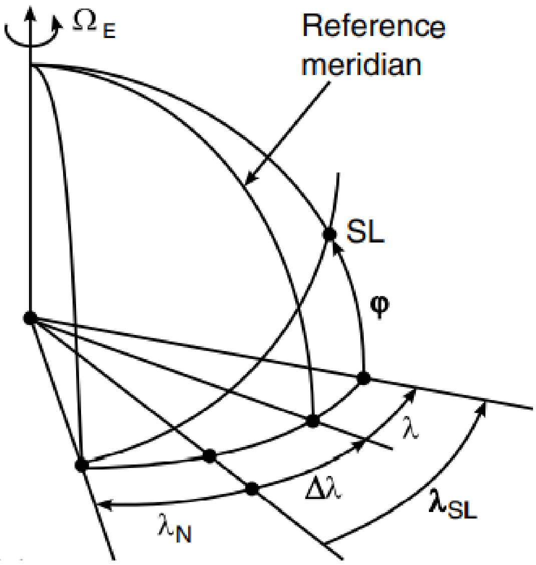


Figure 5.7: Visualisation of the latitude and longitudinal satellite parameters, from [31]

Algorithm 4 TTC Analysis

```
Check frequency compliance
if frequency complies then:
    Check previously calculated performance
    loop Calculate achievable data rate:
        Calculate achieved link margin with required data rate
        Vary data rate until a link margin of 3 is achieved
    end loop
    if Achieved data rate > Required data rate then:
        $R_{TTC} = 1$
    else
        $R_{TTC} = \frac{1}{DR_{req}} DR_{ach}$
    end if
else frequency does not comply:
    $R_{TTC} = 0$
end if
```

The components that have to be selected by the tool are the antenna, transceiver and ground station. To limit the number of components in the component database but still enable different configurations, three distinct frequency configurations are created. The three frequency bands that are used are VHF, UHF and S-band. Each component type has an option for each frequency band. On top of that, two different ground stations are entered for the S-band frequency, to see whether this influences the training of the agent. This results in three component options for the antenna, three for the transceiver, and four options for the ground station. Their specifications can be seen in section A.1.

The required data rate of the downlink is known from the user entered required daily data rate and the time in view of the satellite with respect to the ground station. This required data rate is independent of the selected components and it should therefore be evaluated what data rate can be achieved with a certain configuration of components. First, there is an elemental parameter which determines if the components can communicate at all which is the frequency at which they operate. For this version of the tool, it is allowed to select the antenna of the satellite and the ground station. The transceiver of the satellite is chosen automatically based on the frequency of the antenna. This is possible because of the low number of components. For future versions, this should be implemented as a variable option as well. If the antenna and ground station operate at a different frequency, a reward of 0 is immediately given. If they operate at the same frequency, the performance of the selected configuration is evaluated.

It is desired to implement a continuous reward system for this subsystem to be able to compare the performance of different configurations. It should therefore be investigated what the capability of the current configuration which the agent has chosen is. In stead of using the required data rate to determine the link budget of this configuration, which would only indicate whether the system is capable of achieving the data rate or not, the analysis is performed as such to determine the maximal data rate that is possible with the configuration. In this way, a continuous reward can be given. For a given configuration, the tool will iterate over the used data rate in the link budget calculations until a desired link margin is achieved. If the achieved data rate is larger than the required data rate, a reward of 1 is given. If the achieved data rate is lower than the required data rate, the reward is calculated with Equation 5.19. An example of this reward system for certain parameters can be seen in Figure 5.8. Note that this iteration is computationally expensive, the results of the different configurations are therefore saved and re-used when the agent selects those configurations again, saving computation time.

$$R_{TTC} = \frac{1}{R_{req}} \cdot R_{ach} \tag{5.19}$$

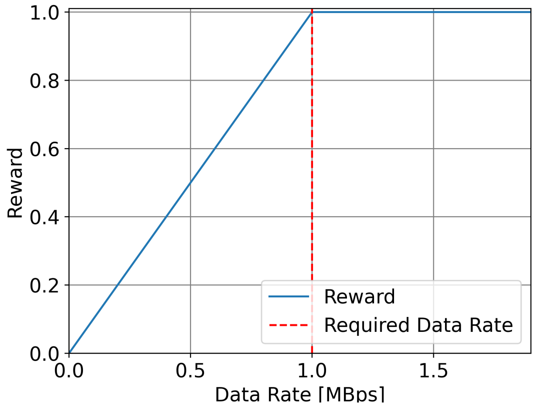


Figure 5.8: Reward system for the TTC subsystem

5.4. Attitude Determination Component

The next component type that is treated, is the attitude determination (AD) component type. Different options exist for this component type, ranging from star trackers to horizon sensors. In this section, the analysis methodology for this component type is explained in subsection 5.4.1 and the method of implementation into the tool is explained in subsection 5.4.2.

5.4.1. Attitude Determination Analysis

The only component type that is added during this dimension increase is the AD type. A multitude of different options exist for this type, but they can be expressed in similar specifications.The requirement for the AD dimension is shown in Table 5.10. The specifications that are used for this component type are shown in Table 5.11. No extensive analysis is required for the implementation of the AD dimension.

Table 5.10: Attitude determination user requirements

Identifier	User Requirement	Unit
REQ-AD-01	Pointing Accuracy	degrees

Table 5.11: Attitude determination component specifications

Component Type	Specification	Unit
Attitude Determination	AD Accuracy	Arcsec

5.4.2. Implementation

From the user entered requirements, it is known what pointing knowledge accuracy should be achieved. Using this requirement, it can be directly assessed whether the component complies to this requirement from the component specifications. The pseudocode for the implementation of the AD dimension can be seen in Algorithm 5. The reward system for the AD dimension can be seen in Figure 5.9. This figure is created for an example required attitude determination knowledge accuracy of 0.2 degrees, which is indicated by the dotted line in the figure. A reward of 1 is given when the component complies to the required accuracy and follows a continuous slope towards 0 if it does not. The point where a reward of 0 is given is determined by the outermost component. The components that are entered into the database are selected as such that the database provides a broad range in the performance of AD components. This ranges from star trackers to horizon sensors. The database can be found in section A.1.

Algorithm 5 AD Analysis

```
Obtain component pointing accuracy
if Achieved pointing accuracy > Required pointing accuracy then:
    $R_{AD} = 1$
else
    $R_{AD} = -(\frac{1}{acc_{lim}} acc_{ach}) + 1$
end if
```

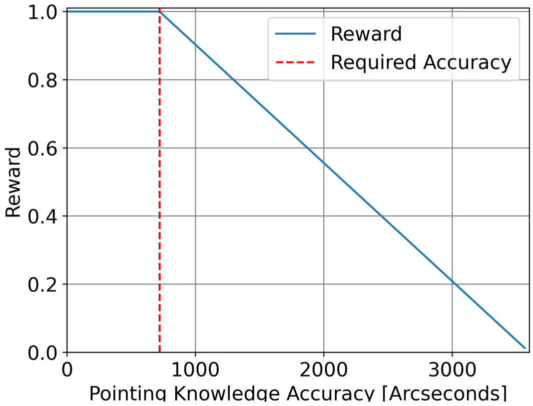


Figure 5.9: Reward system for the attitude determination component type

5.5. Attitude Control Component

The attitude control (AC) component type is also included in the satellite configuration. AC can be achieved by using a variety of components, such as solely magnetorquers, reaction wheels or a combination of different components. Since these different types of AC components achieve attitude control through different means, the tool should be able to differentiate between the different types. In this section, the analysis methodology of the AC component type is explained in subsection 5.5.1 and the implementation of this analysis with the reward system is explained in subsection 5.5.2.

5.5.1. Attitude Control Analysis

To analyse the performance of an AC component, performance parameters which defines whether the component is able to provide the required level of control should be established. The two performance parameters that are defined for the component are the torque and momentum storage capabilities. To determine these parameters, an analysis of the torques that are required should be performed. Next to that, the user of the tool will input several user requirements that serve to determine the required performance parameters. The user requirements are listed in Table 5.12. The parameters in which the components are specified can be seen in Table 5.13.

Table 5.12: Attitude control user requirements

Identifier	User Requirement	Unit
REQ-AC-01	Number Control Axes	-
REQ-AC-02	Maximum Slew Angle	degrees
REQ-AC-03	Slew Time	seconds
REQ-AC-04	Momentum Storage Margin Factor	-

Table 5.13: Attitude control component specifications

Component Type	Subtype	Specification	Unit
Attitude Control	Magnetorquer	Axis Control	-
		Dipole Moment	degrees
	Reaction Wheel	Axis Control	-
		Torque	Nm
		Momentum	Nms

The torques that the AC component(s) should be able to provide are to counteract external torques that are acting on the satellite throughout the mission and to perform manoeuvres. The external torques are called disturbance torques. The disturbance torques that are assumed to relevant for the analysis are the effects caused by solar radiation pressure (SRP), atmospheric drag, the magnetic field and the gravity gradient. Next to these disturbance torques, the satellite should be able to perform slew manoeuvres for which the requirements are determined by the user.

Torque Estimation

The disturbance torques depend on the geometry and other specifications of the satellite, such as mass. Because the agent selects a complete configuration at every step, these specifications can be determined before initiating the AC analysis. At this dimension however, the satellite configuration is not yet complete since there are still components missing which will be included in further dimensions. This incompleteness will affect the selection process of the agent, since the AC component will be selected for an incomplete satellite. The equations to estimate the disturbance torques are obtained from [50].

Solar Radiation Pressure

The first of the disturbance torques is the torque generated by the solar radiation pressure. This pressure is caused by the the transfer of momentum of the sunlight to the satellite. Because this momentum is transferred differently on different parts of the satellite, an external torque can be generated. This torque can be estimated using Equation 5.20. Here, Φ is the solar constant, set at 1366 $\frac{W}{m^2}$. c is the speed of light and A_s is the sunlit surface area of the satellite. For the sunlit surface area, the maximal phase area of the satellite is used. q is the reflectance factor, which is 0 at perfect absorption and 1 at perfect reflection. The reflectance factor is assumed to be uniform across the satellite [50] and is assumed at 0.6. cp_s and cm are the centers of SRP and mass and $cp_s - cm$ is the distance between those centers. The worst case distance for this term is assumed to be half of the largest phase area of the satellite. φ is the angle of incidence of the incoming sunlight and for a worst case analysis is assumed to be 0.

$$T_{SRP} = \frac{\Phi}{c} A_s \, (1 + q) \, (cp_s - cm) \, cos\theta \tag{5.20}$$

Atmospheric Drag

An external torque can also be generated by the atmosphere through which the satellite is moving. A satellite in LEO, which is mostly the operating region for CubeSats, encounters the atmosphere at a density which can not be neglected. The particles of the atmosphere impact the satellite and similar to the SRP can cause an external torque. This torque can be estimated using Equation 5.21. Here, ρ is the atmospheric pressure, C_d is the drag coefficient which is assumed at 2.2 [33], [50]. A_r is the area facing the ram or flight direction and is again assumed to be the maximal phase area of the satellite. V is the velocity of the satellite, which depends on the orbit altitude and can be determined for a circular orbit using Equation 5.22, where G is the gravitational constant, M is in this case the mass of the Earth and r is the orbit radius. $cp_a - cm$ is the distance between the center of atmospheric pressure and the center of mass, respectively. A worst case distance for this term is used as half of the maximal distance of the largest phase of the satellite.

$$T_a = \frac{1}{2}\rho C_d A_r V^2 \left(cp_a - cm\right) \tag{5.21}$$

$$V = \sqrt{\frac{GM}{r}} \tag{5.22}$$

Magnetic Field

Due too the electronic devices on a satellite, a satellite can generate its own magnetic field. When this field is not aligned with the Earth's magnetic field, an external torque acts on the satellite that attempts to align both fields. This external torque can be estimated using Equation 5.23. Here, D is the satellite's residual dipole moment, a measure of the strength of the satellite's magnetic field. B is the magnetic field strength of the Earth, which can be calculated by the term in brackets in Equation 5.23. Between the brackets, M is the magnetic moment of the Earth times the magnetic constant which results in $M = 7.8 \cdot 10^{15} Tm^3$. R is the radius of the satellite's orbit and finally, λ is a function of the magnetic latitude, which is 1 at the magnetic equator and 2 at the magnetic poles. λ is therefore directly dependent on the orbit's inclination and can also be calculated using Equation 5.24.

$$T_m = DB = D\left(\frac{M}{R^3}\lambda\right) \tag{5.23}$$

$$\lambda = 2 - cosi \tag{5.24}$$

Gravity Gradient

The final disturbance torque that is considered is caused by the gravity gradient. This torque is caused by a misalignment between the center of gravity and center of mass of the satellite. The torque can be estimated using Equation 5.25. Here, μ is the Earth's gravitational constant, R is the radius of the orbit. I_z and I_y are the moments of inertia about the Y and Z axis, respectively. θ is the angle between the local vertical and the Z principle axis, which is can be determined by the user of the tool.

$$T_g = \frac{2\mu}{2R}\left|I_z - I_y\right| sin2\theta \tag{5.25}$$

Slew Torque

The AC component should be able to counteract the disturbance torques that were explained previously. Next to that, it should be capable of performing manoeuvres that adjust the attitude of the satellite. These manoeuvres could be required during communication windows where the satellite should point towards the ground station, or for pointing towards an area of interest for the payload. Such a manoeuvre is called a slew manoeuvre. The torque required for a certain slew manoeuvre can be estimated using Equation 5.26. Here, θ is the angle around a specific axis for the slew manoeuvre and I is the moment of inertia of the satellite corresponding to this manoeuvre direction. t is the time in which the manoeuvre should be performed. Both the angle θ and tme t are user entered requirements.

$$T_{slew} = \frac{4\theta I}{t^2} \tag{5.26}$$

After the estimation of all disturbance torques and slew torques, the maximum required torque is defined by the maximum torque that acts on the satellite or should be exerted.

Momentum Storage Estimation

Torques acting on, or exerted by, the satellite causes momentum to build up. This momentum needs to be stored and periodically dumped in order to maintain attitude control. Both the disturbance and slew torques require a momentum storage capability. The requirement to this capability is determined by the maximum of the momentum build ups. The momentum that builds up during a slew manoeuvre can be estimated using Equation 5.27, from [50]. The momentum is assumed to not build up after a full manoeuvre, since a reverse rotation is required to obtain the nominal attitude, thereby cancelling out the momentum that was build up.

$$M = T_{slew}\frac{t}{2} \tag{5.27}$$

The momentum buildup causes by the disturbance torques depends on the type of disturbance torque and the attitude of the satellite. It is assumed that a disturbance torque accumulates its maximum momentum in $1/4$ of an orbit. The equation with which this momentum is estimated is Equation 5.28, from [50]. This momentum build-up caused by the disturbance torques is calculated for every disturbance torque individually.

$$h = T_D P \frac{0.707}{4} \tag{5.28}$$

After all the momentum build-ups caused by the various torques have been estimated, the largest number is used as the requirement for the attitude control components. Both the requirements on the amount of torque and momentum storage are used to assess the components in the database.

5.5.2. Implementation

In subsection 5.5.1, the analysis method to estimate the required torque and momentum storage capability of the AC components are explained. The tool should then assess the performance of the component that is currently selected by the agent against the requirements, and provide a reward based on this performance. The pseudocode for the implementation of the AC dimension can be seen below in Algorithm 6. In Table 5.13, it can be seen that the component type is subdivided into magnetorquers and reaction wheels. This means that the components present in the database can be both magnetorquers or reaction wheels. The components that can be selected by the agent are provided in section A.1. Depending on the type of component, determining the performance of the component with respect to the user requirements changes. If the component is of the reaction wheel type, the performance can be evaluated simply by obtaining the torque and momentum capability from the specifications. For the magnetorquer type however, the torque and momentum capability has to be estimated. This is done using Equation 5.29, from [22]. The inclination i is included to account for the inclination of the orbit.

$$T = D_{mtq}\frac{M_E}{R^3}(2 - |cosi|) \tag{5.29}$$

The reward is divided into two parts, one part of the reward is given for the torque capability and the other part for the momentum storage capability. Both rewards are given through the same semi-continuous reward system as encountered before, where a reward of 1 is given when the component satisfies the requirement. If the component does not satisfy the requirement, the reward is given based on a continuous slope of 1 to 0 with the point where this line crosses 0 is the outermost point of the components database. An example of reward graphs are shown in Figure 5.10 and Figure 5.11. For the magnetorquer subtype, the reward for the torque and momentum capabilities is the same, since the system will prevent a momentum build up if the system can cope with the torque requirement.

Algorithm 6 AC Analysis

```
Determine the maximum required torque
Determine the maximum required momentum storage
if Component axis control >= Required axis control then:
    Determine/obtain torque and momentum capability of the component
    if Achieved torque > Required torque then:
        $R_{torque} = 1$
    else
        $R_{torque} = -(\frac{1}{T_{lim}} T_{ach}) + 1$
    end if
    if Achieved momentum > Required momentum then:
        $R_{momentum} = 1$
    else
        $R_{momentum} = -(\frac{1}{M_{lim}} M_{ach}) + 1$
    end if
    $R_{AC} = \frac{R_{torque} + R_{momentum}}{2}$
end if
```

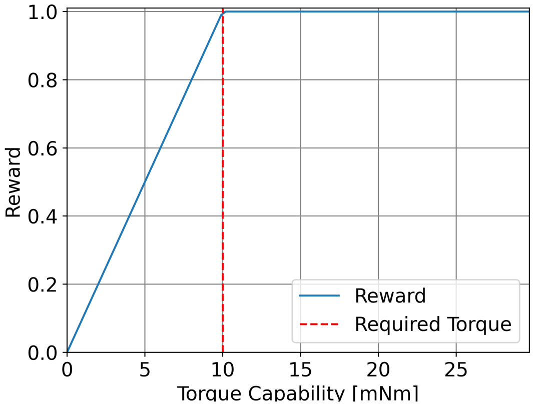


Figure 5.10: Reward system for the torque capability of the attitude control component

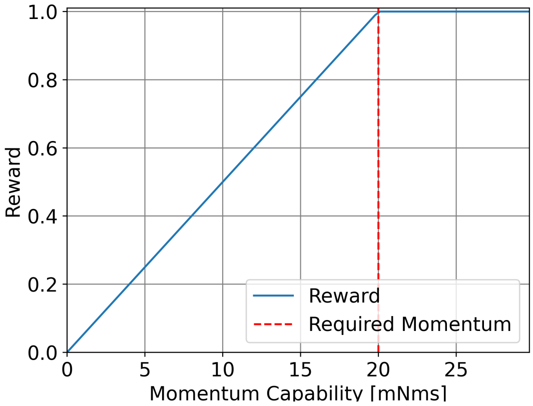


Figure 5.11: Reward system for the momentum capability of the attitude control component

5.6. Power Control Unit Component

The first component of the EPS that is included is the power control unit (PCU). The inclusion of the PCU does not require significant analyses, since the requirements for this component can be obtained directly from the other components' specifications. The analysis part of this dimension is explained in subsection 5.6.1 and the reward system is explained in subsection 5.6.2.

5.6.1. Power Control Unit Analysis

The PCU has to ensure the delivery of power to the entire satellite configuration. Each component has its own specific operating voltage and the PCU should accommodate for that. The analysis part is therefore to establish the operating voltage range of the satellite configuration. This is done by retrieving the operating voltage from every component in the satellite configuration. Using this, the selected PCU component can be checked for compatibility for the voltage range. The requirements for the PCU are determined by the configuration of components that is selected and therefore no user requirements are provided for this dimension. The specifications in which the components are entered into the database are shown in Table 5.14.

Table 5.14: Power Control Unit component specifications

Component Type	Specification	Unit
Power Control Unit	Output Voltage	V

5.6.2. Implementation

The reward for the PCU component is given based on its capability to supply the satellite configuration at the right operating voltages of the components. Because the PCU can either comply to this range, or is infeasible for supplying this voltage range, the rewards that are given are either 1 or 0. A reward of 1 is given by the agent if the PCU is able to supply the correct voltages and a reward of 0 is given if this is not the case. The PCU components that are available to the agent are listed in section A.1.

5.7. Battery Component

A battery component is included in the final setup of the design tool. To analyse the performance of the EPS, the parameter that indicates the performance is taken to be the depth-of-discharge (DoD). The desired DoD can be specified by the user of the tool. The DoD can be estimated by analysing the power consumption and generation of the satellite. Establishing these aspects is the main goal of the analysis for this dimension. This will be explained in subsection 5.7.1. After that, the implementation of this analysis in terms of rewarding the components is elaborated upon in subsection 5.7.2.

The incorporation of the battery component means that the configuration of components that the agent selects influences the DoD reward by changing the power consumption. Next to that, the size of the satellite for the selected configuration of components also has influence on the AC subsystem and its reward and the power generation of the satellite with again the EPS reward. In all, the configuration of components selected by the agent influences the different rewards that the agent will obtain. This has the result that the agent no longer only tries to maximise the subsystem rewards, but looks for a solution that is the global maximum. It is easier to explain this through an example. For the payload requirements, a certain camera might get the best reward, however this camera might also be very power consuming, which would influence the EPS reward. The goal is that the agent will select the camera that has the best overall reward, and not only the best payload reward.

5.7.1. Battery Analysis

The analysis of the battery component consists of two parts. The first part is establishing the power consumption of the satellite throughout the mission. The next part is establishing the amount of power the satellite can generate using solar panels. These will be explained separately and then combined in the section where the DoD is discussed. The requirements for the EPS that can be entered by the user are listed in Table 5.15. Several parameters have to specified by the user, which are listed in Table 5.16. The specifications of the components in the database are shown Table 5.17.

Power Consumption

The power consumption of the satellite depends on several factors. First, there is the configuration of the satellite itself, with the power consumption of the different components that make up the configuration. Next to that is the configuration of the mission orbit. A regular mission orbit, which will be called an

Table 5.15: Electrical Power System user requirements

Identifier	User Requirement	Unit
REQ-EPS-01	Payload Duty Cycle	%
REQ-EPS-02	Dumping Duty Cycle	%
REQ-EPS-03	Maximum Depth of Discharge	%
REQ-EPS-04	Prepositioning Time	s
REQ-EPS-05	Mission Lifetime	yr

Table 5.16: User entered solar light angles

Parameter	Unit
X positive - angle	degrees
X positive - time factor	-
X negative - angle	degrees
X negative- time factor	-
Y positive - angle	degrees
Y positive - time factor	-
Y negative - angle	degrees
Y negative- time factor	-
Z negative - angle	degrees
Z negative- time factor	-

Table 5.17: Electrical Power System component specifications

Component Type	Specification	Unit
Electrical Power System	Capacity	Wh

operational orbit, contains several segments during which a combination of components can be active. To establish the power consumption of the satellite, an operational orbit has to be established.

Operational Modes
The operational orbit is divided into segments during which specific operational modes are active. The length and type of operation mode for each segment depends on the mission requirements. The user entered requirements, which can be seen in Table 5.15, that are applicable here are the duty cycle of the payload and dumping, and the prepositioning time. The operational modes that are used to create an operational orbit are IDLE, DUMP, COMM and OPS. The IDLE operational mode is the mode where no specific activities or manoeuvres are performed, and can therefore also be seen as the mode where the batteries can be recharged. The DUMP mode is the mode during which the excess momentum that is stored in the satellite is dumped, using the attitude control components. The COMM mode is the mode during which communications with a ground station is established and data can be uploaded and/or downloaded. Finally, the OPS mode is the mode during which the payload is being used and therefore the mode during which the mission data is being obtained. The components that are active during each operational mode, can be seen in Table 5.18, with components that are active marked with a 1 and inactive with a 0. Using this table, the tool is able to specify the power consumption of each operational mode. This can be done because each component's specifications include the maximum power consumption. At the moment, only the maximum power consumption of the components are to analyse the power consumption. The components are therefore either fully on or off, meaning they use the maximum power consumption or do not consume any power at all. For future versions of the tool, the tool could be expanded to use different component power consumption modes.

Operational Orbit
With the operational modes known, the time these modes are active throughout an orbit define the operational orbit. The operational orbit is divided into segments in each of which an operational mode is active. An example of this division into segments is illustrated in Figure 5.12. The duration of each segment can be determined through the user entered parameters in Table 5.15. From the requirements

Table 5.18: Operational modes of the satellite with corresponding component mode

Operational Mode	PAY	TTC	OBC	ADC	EPS
OPS	1	0	1	1	1
COMM	0	1	1	1	1
DUMP	0	0	1	1	1
Idle	0	0	1	1	1

on the duty cycle of operational the time of this segment can be calculated using Equation 5.30, where the duty cycle should be provided as a percentage, and P is the orbit period. The time of the segments can be calculated for the operations mode, during which the payload is active, and the dumping mode, during which excess momentum is dumped. The operational modes that remain are the communications and idle modes. The communications mode time is determined by the average amount of time the satellite has access to a ground station during an orbit. This value was already determined for the TTC analysis and is therefore known. For all of these previous mode, a prepositioning time is included in the segments, which allows for the satellite to attain the attitude that is required for that mode. The time that is left of the orbit period after the previous three modes, OPS, DUMP, and COMM, is appointed to the IDLE mode. The power consumption for the entire orbit is then calculated using Equation 5.31.

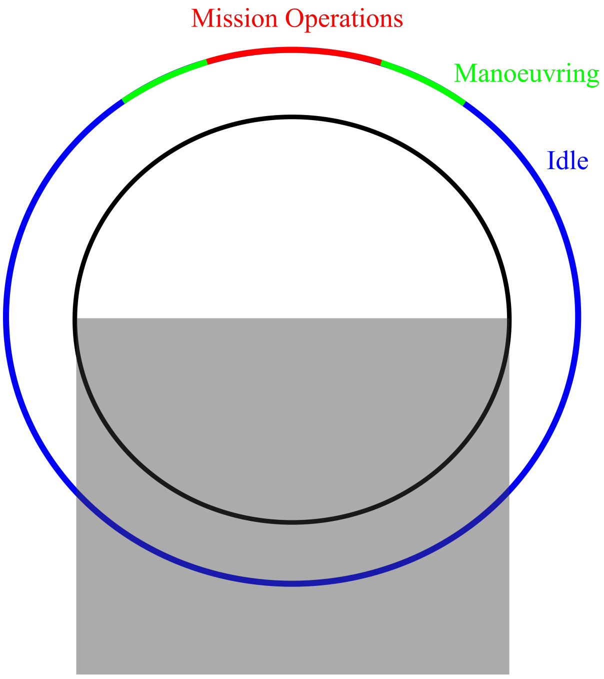


Figure 5.12: Example of nominal operational orbit

$$t_{seg} = \frac{DutyCycle}{100} P \tag{5.30}$$

$$P_{cons_{orbit}} = \sum P_{cons_{mode}} t_{mode} \tag{5.31}$$

Power Generation

The next part that is analysed is the power generation of the satellite. The power is generated through the use of solar panels. For this dimension, it is assumed that all solar panels are body mounted and that solar panels are located on every phase of the satellite. The satellite size and outer dimensions are generated in subsection 5.1.8. Using these dimensions, the amount of solar panel area per satellite phase is calculated. The average incidence angle is specified by the user entered parameters that are listed in Table 5.16. For this thesis, these angles have been obtained using the free version of

STK and processing these results in Python. In STK, a report is generated with the angle between the incoming sunlight and each positive body axis for a specific orbit. This report is read in Python and processed to output the average incidence angle for each satellite phase. Next to that, the fraction of time that the phase is in sunlight and the solar panels would be able to generate power is generated as output.

Next, the amount of power per unit area the solar cells are capable of generating is calculated with Equation 5.32. Here, η is the solar cell efficiency and P_i is the incoming solar illumination intensity, which is taken as $1367\frac{W}{m^2}$, as stated by [22]. Because the solar cells experience a level of degradation over time, the end of life (EOL) power generation per unit area should be calculated. This is taken into account by the parameters I_D and L_D, which are the inherent degradation and the lifetime degradation, respectively. The lifetime degradation is a function of the degradation factor D and the lifetime in years L. Finally, θ is the Sun incidence angle. The power generation per unit area at the EOL for each phase can then be calculated. The power output for each phase is calculated using Equation 5.33, where A_{phase} is the area of a satellite phase. It is assumed that the positive z phase does not contain any solar panels due to other components being present on that side.

$$\begin{aligned} P_{EOL} &= \eta P_i I_D cos\theta L_D \\ L_D &= (1-D)^L \end{aligned} \tag{5.32}$$

$$P_{phase} = P_{EOL_{phase}} A_{phase} \tag{5.33}$$

Finally, the amount of power every satellite phase generates during an orbit is estimated using Equation 5.34. In this equation, P_{phase} is calculated using Equation 5.33, P is the orbit period. E is the maximal eclipse fraction and F_{sunlit} is the fraction of time the phase is in direct sunlight. To estimate the maximal eclipse fraction, a separate analysis is performed, which is explained next.

$$P_{orbit_{phase}} = P_{phase} P (1-E) F_{sunlit} P_{gen_{orbit}} = \sum P_{orbit_{phase}} \tag{5.34}$$

Eclipse Fraction

To estimate the eclipse fraction, Equation 5.35 from [37], is used. The angle θ is visualised in Figure 5.13. Using the fraction of the orbit that is spent in eclipse, the time that is spent in eclipse can be estimated by multiplying the eclipse fraction by the orbit period, which is calculated using Equation 5.36. In Equation 5.35, the beta angle β is still unknown and therefore has to estimated as well.

$$sin(\theta) = \sqrt{\frac{1}{cos^2(\beta)}[(\frac{r_e}{r_e+H})^2 - sin^2(\beta)]} \tag{5.35}$$

$$P = 2\pi\sqrt{\frac{r^3}{GM}} \tag{5.36}$$

The beta angle β is visualised in Figure 5.14, and can be calculated using Equation 5.37 [37]. The angle can be described as the angle between the solar vector and the projection of this vector onto the orbital plane.

$$\beta = sin^{-1}(cos(\Gamma)sin(\Omega)sin(i) - sin(\Gamma)cos(\varepsilon)cos(\Omega)sin(i) + sin(\Gamma)sin(\varepsilon)cos(i)) \tag{5.37}$$

In Equation 5.37, Γ is the ecliptic true solar longitude and ranges from 0-360° in a year. Ω is the right ascension of the ascending node (RAAN) and is variable throughout a mission. Next i is the inclination of the orbit and finally, ε is the obliquity of the ecliptic of Earth, which is 23.45°. The RAAN precession due to the J2 effect is estimated using Equation 5.38. The precession depends on the mission's orbit

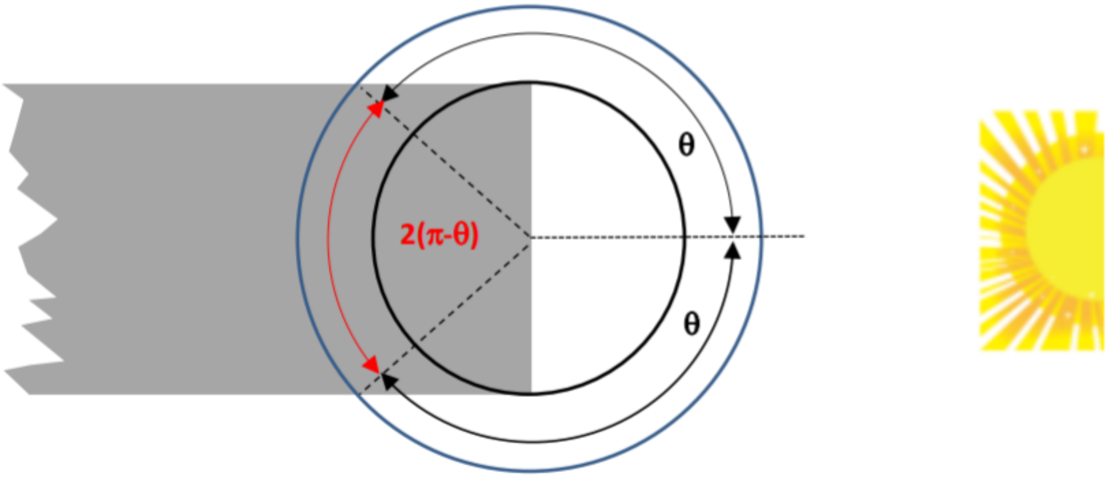


Figure 5.13: Visualisation of the eclipse with angle θ, from [37]

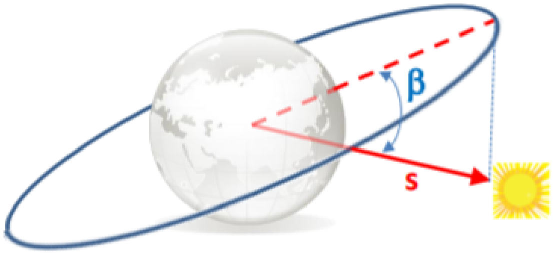


Figure 5.14: Visualisation of the beta angle β, from [37]

parameters. The parameter p is determined using Equation 5.39, which is equal to the orbit radius for a circular orbit.

$$\dot{\Omega} = -\frac{3}{2} J_2 (\frac{r_{eq}}{p})^2 \sqrt{\frac{\mu}{r^3}} cos(i) \tag{5.38}$$

$$p = a(1 - e^2) \tag{5.39}$$

Depth-of-Discharge
Now the power consumption and generation have been analysed, it is possible to analyse the DoD of the batteries that are selected by the agent. It is decided to estimate the DoD at 2 points in 5 consecutive orbits. The 2 points are the points between eclipse and sunlight. These points are selected because at the end of eclipse, the batteries have not been charged for the largest amount of time and the DoD should be the highest for a stable EPS. The other point, at the end of the sunlight period, is selected to check whether the solar panels are able to recharge the batteries after the largest period of sunlight. It is important to analyse the DoD in consecutive orbits in order to see whether the EPS is a stable system. It could be that after a single orbit the DoD remains under 20%, but after consecutive orbits there is an upward trend in the DoD. It is assumed that 5 consecutive orbits should suffice to check whether the EPS is able to sustain the satellite.

An average power consumption of the satellite is used for the DoD analysis, which is estimated by summing every operational orbit segment's power consumption and dividing this by the orbit period, Equation 5.40. An important assumption for this to hold is that the power generated by the solar panels outweigh the maximum power consumption of the satellite. This means that the net change of the charge of the batteries is positive during sunlight. If this would not be the case, it might be possible that during sunlight operations, the DoD can rise which would not be detectable with this method. The power consumption in Wh is then estimated for the duration of eclipse and sunlight using this average power consumption.

$$P_{cons_{avg}} = \frac{P_{cons_{orbit}}}{P} \tag{5.40}$$

The DoD is then estimated by iterating through 5 consecutive orbits and applying the power consumption and generation at each point in the orbit. From the batteries' specifications, the capacity is known and the DoD can be calculated using Equation 5.41. At each point the DoD is saved and the maximum DoD of the batteries can be obtained after the iteration.

$$DoD = 1 - \frac{Charge_{current}}{Charge_{max}} \tag{5.41}$$

5.7.2. Implementation

The reward system that is used to reward the agent's choice of battery is similar to the reward systems that were used in previous dimensions. A reward of 1 is given when the component achieves the user entered requirement on maximum DoD. If the component does not achieve this performance, a reward is given that follows a straight line from 1 to 0, with the 0 point being assigned to a DoD of 100, which means that the batteries are completely drained. An example of the reward system for an example required DoD can be seen in Figure 5.15. As explained in subsection 5.7.1, the DoD can be estimated by estimating the power consumption of the components selected by the agent. This power consumption is estimated by using the power consumption that is specified for every component that requires power. The power generation can be estimated through the estimated size of the satellite with the configuration of components. The pseudocode for the implementation of the EPS dimension is shown in Algorithm 7. The battery components that the agent is able to select are provided in section A.1

Algorithm 7 DoD Analysis

```
Calculate the orbit modes operation times
Calculate the orbit modes power consumption
Calculate the orbit power consumption
Calculate the orbit power generation
Determine the maximum eclipse fraction
Determine the maximum depth of discharge
if Achieved DoD < Required DoD then:
    $R_{battery} = 1$
else
    $R_{battery} = -(\frac{1}{DoD_{lim}} DoD_{ach}) + 1$
end if
```

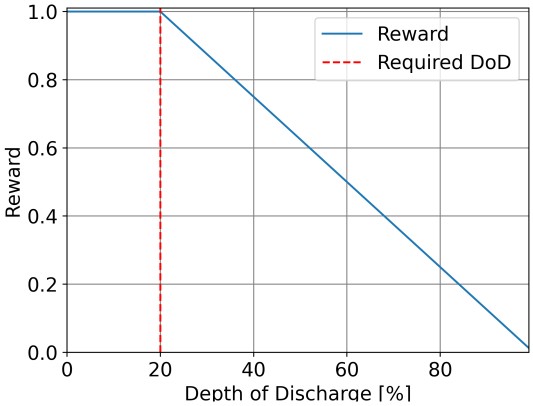


Figure 5.15: Reward system for the EPS dimension

II

Case Study & Results

6 Validation: AltiCube Case Study

The case study that is selected to perform a validation of the design tool with is the AltiCube mission study. The AltiCube project is a feasibility study of a Ka-band altimeter CubeSat constellation for ocean monitoring, carried out by the DUT, specifically by the Civil and Aerospace Engineering faculties, under the support of the European Space Agency. The author of this thesis has been involved in the system design of this mission. The validation of the design tool will be performed by comparing the system designs from both the mission study itself and the design that the design tool provides. In this way, it can be validated whether the design tool provides a feasible CubeSat concept for certain mission requirements.

In section 6.1, the system design that the feasibility study resulted in is presented. After that, the changes that were made to the design tool in order to present a more suitable comparison are shown in section 6.2. The results from the design tool are presented in section 7.3.

6.1. AltiCube Mission Study

The AltiCube mission study is a feasibility study of a Ka-band altimeter CubeSat constellation for ocean monitoring. Through altimetry measurements of the ocean, physical models of the ocean dynamic changes can be validated. Next to that, it could improve the accuracy of weather forecasting, which in turn can provide many benefits. Today's altimeters cannot observe submeso-scale oceanic features. The launch of a CubeSat constellation can solve this problem, while keeping costs low. Radar altimetry on CubeSats has been demonstrated in the NASA RainCube mission [34]. The feasibility study can be seen as an extension to the RainCube mission, the Ka-band altimeter is deployed in a similar manner while being larger. The use of a constellation of multiple CubeSats enables the mission to achieve the required accuracy in the scientific measurements.

The requirements for the AltiCube mission are provided in Table 6.1, and mission parameters that specify the orbital parameters, Table 6.2, ground station location, Table 6.3, and incoming solar light angles, Table 6.4. These requirements are also used in the design tool. The requirements vary slightly from the requirements that are specified in chapter 5 because some requirements are no longer relevant due to the changes to the design tool. These changes are elaborated upon in section 6.2. The dumping duty cycle requirement has been modified to the manoeuvring duty cycle requirement. The dumping duty cycle requirements is no longer relevant because the power mode is identical to the IDLE power mode. The key design specifications of the AltiCube system design can be seen in Table 6.5. Also, an external view of the system design is shown in Figure 6.1.

Next to that, the system design budgets are presented as well. First, the itinerary of the components can be seen in Table 6.6. The mass and power budgets can be seen in Table 6.7 and Table 6.8, respectively. Finally, the sizes of the components that are used are shown in Table 6.9.

Table 6.1: AltiCube case study design tool requirements values

Identifier	**User Requirement**	Value	**Unit**
REQ-TTC-01	Daily Data Downlink	2857	MB
REQ-TTC-02	Achieved $E_B N_0$	0	dB
REQ-AD-01	Pointing Accuracy	0.002	degrees
REQ-AC-01	Number Control Axes	3	-
REQ-AC-02	Maximum Slew Angle	30	degrees
REQ-AC-03	Slew Time	60	seconds
REQ-AC-04	Momentum Storage Margin Factor	1	-
REQ-EPS-01	Payload Duty Cycle	25	%
REQ-EPS-02	Manoeuvring Duty Cycle	2	%
REQ-EPS-03	Maximum Depth of Discharge	20	%
REQ-EPS-04	Prepositioning Time	60	s
REQ-EPS-05	Mission Lifetime	2	yr

Table 6.2: AltiCube case study design tool orbital parameters

Parameter	**Value**	**Unit**
Orbit Altitude	550	km
Inclination	98	degrees
RAAN	180	degrees

Table 6.3: User entered ground station parameters

Parameter	**Value**	**Unit**
Longitude	20.96	degrees
Latitude	67.86	degrees
Minimum Elevation	10	degrees
Simulated Orbits	300	-

Table 6.4: AltiCube case study design tool entered solar light angles

Parameter	**Value**	**Unit**
X positive - angle	45.17	degrees
X positive - time factor	0.50	-
X negative - angle	45.17	degrees
X negative- time factor	0.50	-
Y positive - angle	88.18	degrees
Y positive - time factor	0.66	-
Y negative - angle	86.68	degrees
Y negative- time factor	0.34	-
Z negative - angle	45.17	degrees
Z negative- time factor	0.50	-

6.2. AltiCube Design Tool Version

In Table 6.6, it can be observed that the number of distinct components for the AltiCube system design is far greater than the 6 component types that can be selected by the design tool constructed in chapter 5. To provide an adequate comparison between the results of the design tool and the AltiCube system design, several modifications will have to be made to the design tool. These modifications are aimed to incorporate the components that cannot be selected by the design tool.

6.2.1. Design Tool Modifications

Part of the components that are present in the system design of AltiCube cannot be selected by the design tool. However, to provide a suitable comparison between the two design approaches, the design

Table 6.5: AltiCube design specifications

Specification	Value
Dimensions	200x200x400 mm^3 (16U)
Mass	25 kg
Payload	0.8m Ka-band parabolic antenna & electronics
PROP	Cool-gas 6DOF propulsion, 20m/s Delta-V capability
ADCS	3-axis stabilization with STR-RW, pointing <0.05 deg, attitude knowledge <0.01 deg, positioning <5cm with dual-frequency GPS
CDH	CAN bus, ARM-9 based OBC, 4Gb storage
EPS	90 Wh battery, 2x8U deployed solar panel
TTC	Uplink UHF @ 600bps; Downlink X @ 8 Mbps; ISL S @ 16 kbps

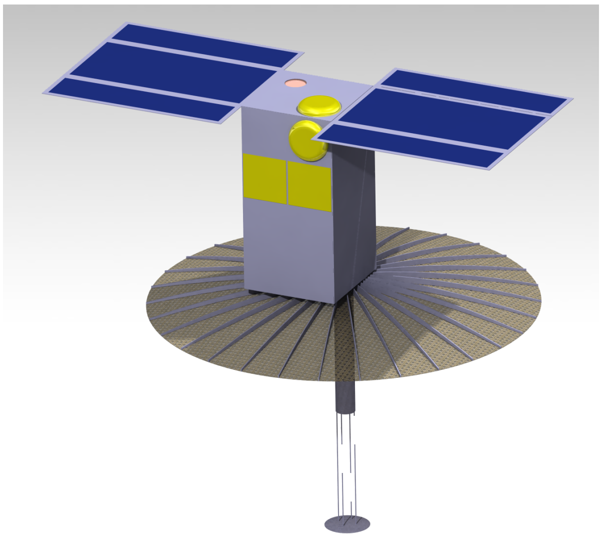
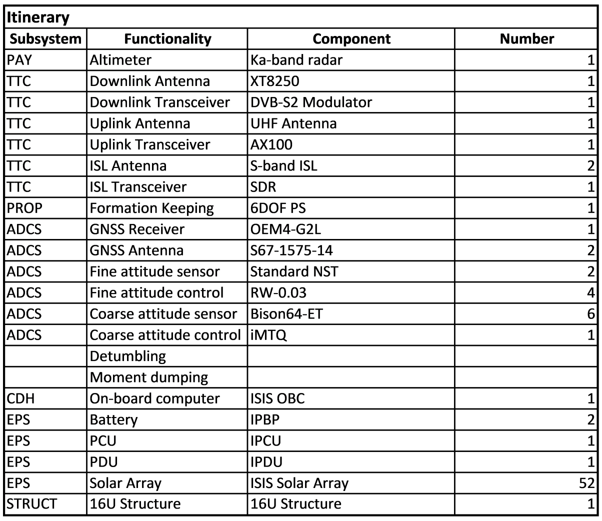


Figure 6.1: External view of the AltiCube CubeSat

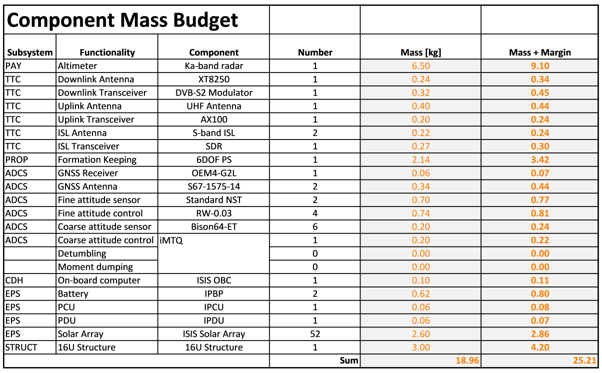


Itinerary			
Subsystem	**Functionality**	**Component**	**Number**
PAY	Altimeter	Ka-band radar	1
TTC	Downlink Antenna	XT8250	1
TTC	Downlink Transceiver	DVB-S2 Modulator	1
TTC	Uplink Antenna	UHF Antenna	1
TTC	Uplink Transceiver	AX100	1
TTC	ISL Antenna	S-band ISL	2
TTC	ISL Transceiver	SDR	1
PROP	Formation Keeping	6DOF PS	1
ADCS	GNSS Receiver	OEM4-G2L	1
ADCS	GNSS Antenna	S67-1575-14	2
ADCS	Fine attitude sensor	Standard NST	2
ADCS	Fine attitude control	RW-0.03	4
ADCS	Coarse attitude sensor	Bison64-ET	6
ADCS	Coarse attitude control	iMTQ	1
	Detumbling		
	Moment dumping		
CDH	On-board computer	ISIS OBC	1
EPS	Battery	IPBP	2
EPS	PCU	IPCU	1
EPS	PDU	IPDU	1
EPS	Solar Array	ISIS Solar Array	52
STRUCT	16U Structure	16U Structure	1

Table 6.6: AltiCube system design itinerary

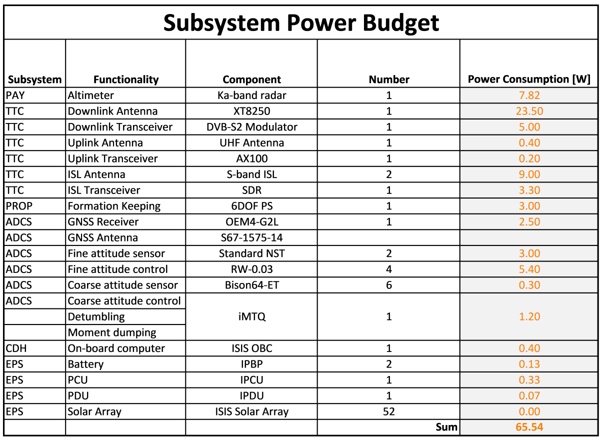
tool should incorporate the components that cannot be selected. If this is not done, the system will most likely be under designed because of a reduced power consumption and mass due to less components. The components that cannot be selected are treated next.

The components that cannot be selected by the design tool in its current version are the following; the



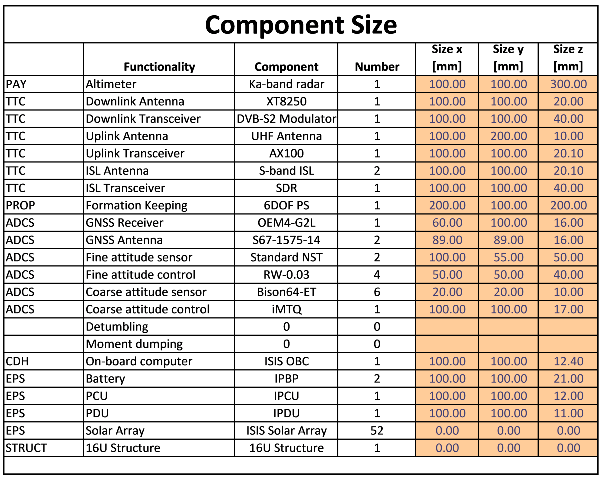
Component Mass Budget					
Subsystem	Functionality	Component	Number	Mass [kg]	Mass + Margin
PAY	Altimeter	Ka-band radar	1	6.50	9.10
TTC	Downlink Antenna	XT8250	1	0.24	0.34
TTC	Downlink Transceiver	DVB-S2 Modulator	1	0.32	0.45
TTC	Uplink Antenna	UHF Antenna	1	0.40	0.44
TTC	Uplink Transceiver	AX100	1	0.20	0.24
TTC	ISL Antenna	S-band ISL	2	0.22	0.24
TTC	ISL Transceiver	SDR	1	0.27	0.30
PROP	Formation Keeping	6DOF PS	1	2.14	3.42
ADCS	GNSS Receiver	OEM4-G2L	1	0.06	0.07
ADCS	GNSS Antenna	S67-1575-14	2	0.34	0.44
ADCS	Fine attitude sensor	Standard NST	2	0.70	0.77
ADCS	Fine attitude control	RW-0.03	4	0.74	0.81
ADCS	Coarse attitude sensor	Bison64-ET	6	0.20	0.24
ADCS	Coarse attitude control	iMTQ	1	0.20	0.22
	Detumbling		0	0.00	0.00
	Moment dumping		0	0.00	0.00
CDH	On-board computer	ISIS OBC	1	0.10	0.11
EPS	Battery	IPBP	2	0.62	0.80
EPS	PCU	IPCU	1	0.06	0.08
EPS	PDU	IPDU	1	0.06	0.07
EPS	Solar Array	ISIS Solar Array	52	2.60	2.86
STRUCT	16U Structure	16U Structure	1	3.00	4.20
			Sum	18.96	25.21

Table 6.7: AltiCube system design mass budget



Subsystem Power Budget				
Subsystem	Functionality	Component	Number	Power Consumption [W]
PAY	Altimeter	Ka-band radar	1	7.82
TTC	Downlink Antenna	XT8250	1	23.50
TTC	Downlink Transceiver	DVB-S2 Modulator	1	5.00
TTC	Uplink Antenna	UHF Antenna	1	0.40
TTC	Uplink Transceiver	AX100	1	0.20
TTC	ISL Antenna	S-band ISL	2	9.00
TTC	ISL Transceiver	SDR	1	3.30
PROP	Formation Keeping	6DOF PS	1	3.00
ADCS	GNSS Receiver	OEM4-G2L	1	2.50
ADCS	GNSS Antenna	S67-1575-14		
ADCS	Fine attitude sensor	Standard NST	2	3.00
ADCS	Fine attitude control	RW-0.03	4	5.40
ADCS	Coarse attitude sensor	Bison64-ET	6	0.30
ADCS	Coarse attitude control	iMTQ	1	1.20
	Detumbling			
	Moment dumping			
CDH	On-board computer	ISIS OBC	1	0.40
EPS	Battery	IPBP	2	0.13
EPS	PCU	IPCU	1	0.33
EPS	PDU	IPDU	1	0.07
EPS	Solar Array	ISIS Solar Array	52	0.00
			Sum	65.54

Table 6.8: AltiCube system design power budget



Component Size						
	Functionality	Component	Number	Size x [mm]	Size y [mm]	Size z [mm]
PAY	Altimeter	Ka-band radar	1	100.00	100.00	300.00
TTC	Downlink Antenna	XT8250	1	100.00	100.00	20.00
TTC	Downlink Transceiver	DVB-S2 Modulator	1	100.00	100.00	40.00
TTC	Uplink Antenna	UHF Antenna	1	100.00	200.00	10.00
TTC	Uplink Transceiver	AX100	1	100.00	100.00	20.10
TTC	ISL Antenna	S-band ISL	2	100.00	100.00	20.10
TTC	ISL Transceiver	SDR	1	100.00	100.00	40.00
PROP	Formation Keeping	6DOF PS	1	200.00	100.00	200.00
ADCS	GNSS Receiver	OEM4-G2L	1	60.00	100.00	16.00
ADCS	GNSS Antenna	S67-1575-14	2	89.00	89.00	16.00
ADCS	Fine attitude sensor	Standard NST	2	100.00	55.00	50.00
ADCS	Fine attitude control	RW-0.03	4	50.00	50.00	40.00
ADCS	Coarse attitude sensor	Bison64-ET	6	20.00	20.00	10.00
ADCS	Coarse attitude control	iMTQ	1	100.00	100.00	17.00
	Detumbling	0	0			
	Moment dumping	0	0			
CDH	On-board computer	ISIS OBC	1	100.00	100.00	12.40
EPS	Battery	IPBP	2	100.00	100.00	21.00
EPS	PCU	IPCU	1	100.00	100.00	12.00
EPS	PDU	IPDU	1	100.00	100.00	11.00
EPS	Solar Array	ISIS Solar Array	52	0.00	0.00	0.00
STRUCT	16U Structure	16U Structure	1	0.00	0.00	0.00

Table 6.9: AltiCube system design component sizes

payload, propulsion system, for the ADCS no coarse ADCS components are selected, the GPS system, the CDH system, the deployable solar panels and finally the uplink and intersatellite-link. In the AltiCube system design, distinct components are present for these systems. The design tool is modified in such a way that the configuration of a concept always incorporates the components that cannot be selected, except for the coarse AD components and the deployable solar panels. For several subsystems, new components are entered in the components database from which the agent can select components. This database can be found in section A.2.

The payload selection for the current version of the tool is the selection of a camera. Because the payload of the AltiCube mission is not a camera, this selection is removed and the payload is fixed as the Ka-band altimeter of the AltiCube system design. Instead, the selection of deployable solar panels is included in the design tool. This is done to keep the complexity of the design tool at the same level by not reducing the amount of component types the design tool is able to select. In subsection 6.2.2, the way in which the selection of deployable solar panels is implemented is elaborated upon.

6.2.2. Deployable Solar Panels Selection

The deployable solar panel component type is specified solely in deployable solar panel factor, Table 6.10. This factor determines the amount of times the solar panels are folded. Full integers are used for this factor, which can range between 0 up to and including 3. The analysis that is performed for the selection of the deployable solar panels is solely in changing the solar panel area per satellite phase. Several extra assumptions are made to calculate the solar panel area per satellite phase when deployable solar panels are included. These assumptions are listed below. The deployable solar panels are not rewarded as an individual system because the influence of the solar panels presents itself in the DoD of the batteries. The deployable solar panels will therefore influence the battery reward.

- The satellite's nominal attitude is nadir pointing.
- The deployable solar panels deploy from the y phases of the satellite and are pointing in the negative z direction. See Figure 6.2 for the used axis system.
- Due too the solar panels being deployed from the y phases of the satellite, these phases do not contain any body mounted solar panels.

Table 6.10: Deployable Solar Panel component type specifications

Component Type	Specification	Unit
Deployable Solar Panel	Deployable Factor	-

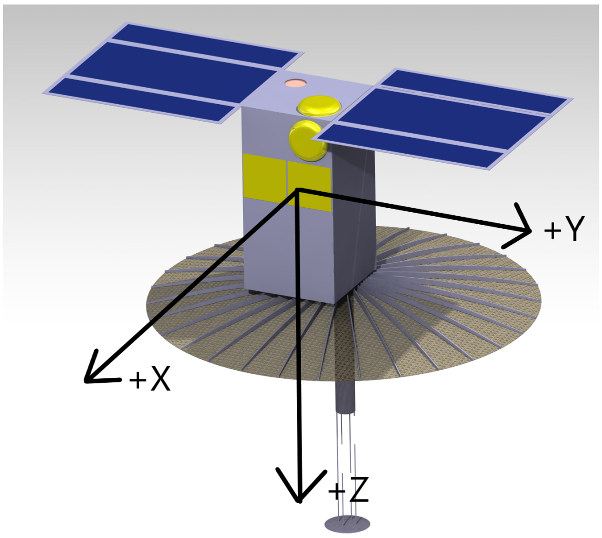


Figure 6.2: External view of the AltiCube CubeSat, with axis system

7 Results & Discussion

The final setup of the design tool was constructed in chapter 5 and the approach with which the design tool is able to generate CubeSat concepts is explained. The results of this work are presented and discussed in three ways in this chapter. There exists a significant difference in complexity of the design tool between the first experiments of chapter 4 to the final setup described in chapter 5. The influence of increasing complexity on the performance of the design tool is treated in section 7.1. After that, it is shown what kind of results the design tool in the final setup is able to generate in section 7.2. These results are generated for randomly configured mission requirements. As there is no reference design for these mission requirements, these results do not indicate the feasibility of the design that the design tool provides. These results do however, provide a measure of the learning performance of the design tool. A case study is used to have a measure of the feasibility of the concept generation capability of the design tool. The design specifications of this case study, the AltiCube mission study, are presented in chapter 6. The results of the design tool applied to the case studies mission requirements are shown and discussed in section 7.3. Finally, a general discussion on the functionality of the design tool is done in section 7.4.

7.1. Design Tool Increase In Complexity

It is aimed to show the impact of complexity on the performance of the design tool. These results could also be used to change the settings of the agent in order to provide an increased performance for the final setup of the design tool. The design tool is increased in complexity iteratively by adding component types in 6 steps. These steps, called dimensions, are shown in Table 7.1. The input for which these results are generated will first be outlined in subsection 7.1.1. After that, the results itself are presented in subsection 7.1.2.

Table 7.1: Component types added during each increase in dimensionality

Dimension	Component Types Added
D1	Camera
D2	Ground Station, Antenna, Transceiver
D3	Attitude Determination
D4	Attitude Control
D5	Power Control Unit
D6	Battery

7.1.1. Design Tool Settings

The results are created for a certain set of settings. The settings are divided into the user input, the component database from which the design tool can select components, and finally the algorithm settings. These settings will be treated in succession.

User Input

The user of the design tool has the ability to enter values for the user requirements, Table 7.2, and mission parameters that specify the orbital parameters, Table 7.3, ground station location, Table 7.4, and incoming solar light angles, Table 7.5. Note that the solar light angles and time factors are obtained through the STK and processing in Python as explained in subsection 5.7.1.

Table 7.2: User requirements values

Identifier	User Requirement	Value	Unit
REQ-PAY-01	Ground Sample Distance	5	m
REQ-PAY-02	Swath	15000	m
REQ-TTC-01	Daily Data Downlink	1000	MB
REQ-TTC-02	Achieved $E_B N_0$	10	dB
REQ-AD-01	Pointing Accuracy	0.2	degrees
REQ-AC-01	Number Control Axes	3	-
REQ-AC-02	Maximum Slew Angle	45	degrees
REQ-AC-03	Slew Time	45	seconds
REQ-AC-04	Momentum Storage Margin Factor	4	-
REQ-EPS-01	Payload Duty Cycle	20	%
REQ-EPS-02	Dumping Duty Cycle	5	%
REQ-EPS-03	Maximum Depth of Discharge	20	%
REQ-EPS-04	Prepositioning Time	60	s
REQ-EPS-05	Mission Lifetime	2	yr

Table 7.3: User entered orbital parameters

Parameter	Value	Unit
Orbit Altitude	500	km
Inclination	97.2	degrees
RAAN	180	degrees

Table 7.4: User entered ground station parameters

Parameter	Value	Unit
Longitude	15	degrees
Latitude	78	degrees
Minimum Elevation	10	degrees
Simulated Orbits	300	-

Table 7.5: User entered solar light angles

Parameter	Value	Unit
X positive - angle	58.27	degrees
X positive - time factor	0.5	-
X negative - angle	58.27	degrees
X negative- time factor	0.5	-
Y positive - angle	54.93	degrees
Y positive - time factor	0.49	-
Y negative - angle	54.57	degrees
Y negative- time factor	0.51	-
Z negative - angle	58.27	degrees
Z negative- time factor	0.5	-

Component Database

The component database used for the generation of the results is shown in section A.1. The agent is able to select components from this database for the generation of concepts.

Algorithm Settings

The algorithm settings that were issued for the generation of concepts for every dimension are shown in Table 7.6.

Table 7.6: Increase in complexity Q-learning algorithm settings

Setting	Value
Number of Episodes	25000
Number of Iterations	100
Epsilon Initial	0.99
Epsilon Decay	0.9997
Epsilon Minimal	0.1
Learning Rate	0.3
Discount Factor	0.95

7.1.2. Increasing Complexity Results & Discussion

The design tool is trained 5 times consecutively for each dimension, where at every dimension an extra component type(s) and a reward is added to the design tool. This makes the design tool more complex by increasing the amount of concepts that are possible and the number of actions that the agent can take increases as well. The results that are shown in this section aim to indicate the effect of this increase in complexity on the performance of the design tool. The design tool is trained 5 times in order to average the results over multiple training cycles. This is done because a factor of randomness is present in the design tool.

The increase in complexity of the design tool with the increasing dimension can be visualised by showing the number of concepts and state-action combinations that is possible for every dimension. The increase in these numbers as a function of the design tool's dimension can be seen in Figure 7.1. The number of concepts and state-action combinations increases exponentially, which was already identified in section 2.1. Next, the amount of time it takes to train the design tool's agent as a function of dimension is shown in Figure 7.2. The time that it takes to train the agent does not follow a strict exponential trend. The time that it takes to train the agent is not influenced by the Q-learning method, since the agent's settings do not change. What does influence the time are the extra analyses that have to be performed at every dimension increase. The greater the computational expense of a added dimension, the bigger the increase in time it takes to train the agent can be observed. This is especially evidenced at the fourth dimension, where the AC component type is added, for which a relative computationally expensive analysis is added to the design tool.

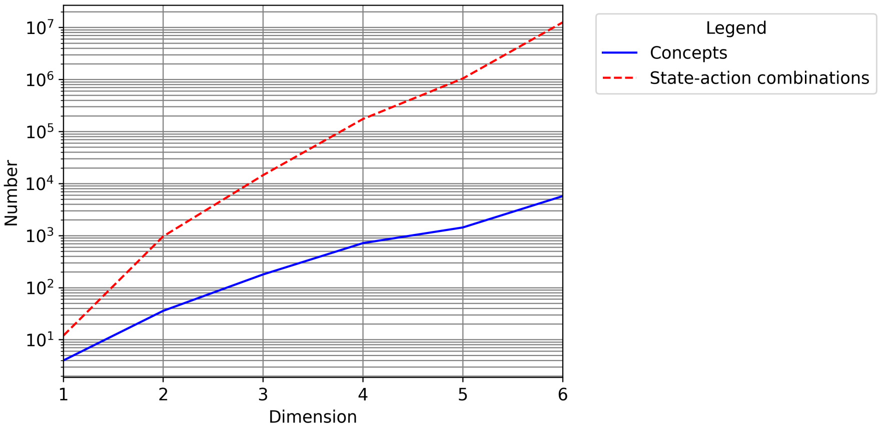


Figure 7.1: Number of concepts and state-action combinations as a function of design tool dimension

The next results that are shown are the rewards that are obtained in the training cycle of the design tool. As mentioned previously, the design tool is trained 5 times for every dimension. The graphs, as were encountered in section 4.2, show the pattern of the obtained reward throughout a training cycle.

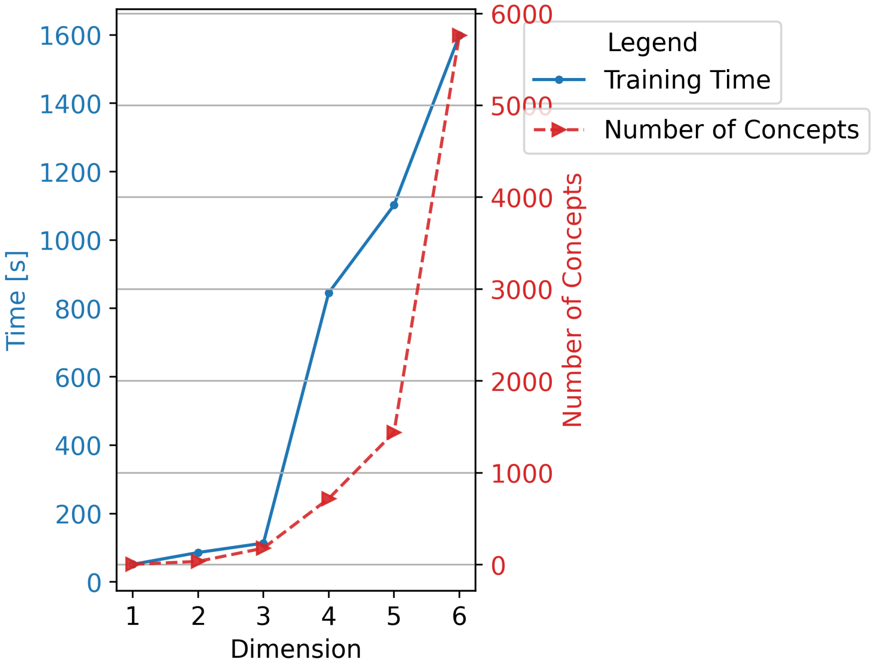


Figure 7.2: Training time and number of concepts and as a function of design tool dimension

The rewards in the graphs are the averaged rewards of a certain number of consecutive episodes of all 5 training cycles. The expected pattern of this training cycle is that the achieved reward increases as the design tool finds itself further in the training cycle. This is expected because the actions that the agent takes move from more randomly to more Q-table based actions, which instructs the agent to exploit more in stead of exploring. In this more exploiting oriented mode, the agent will try to maximise the reward that it can obtain. The epsilon decay over a training cycle can be seen in Figure 7.3.

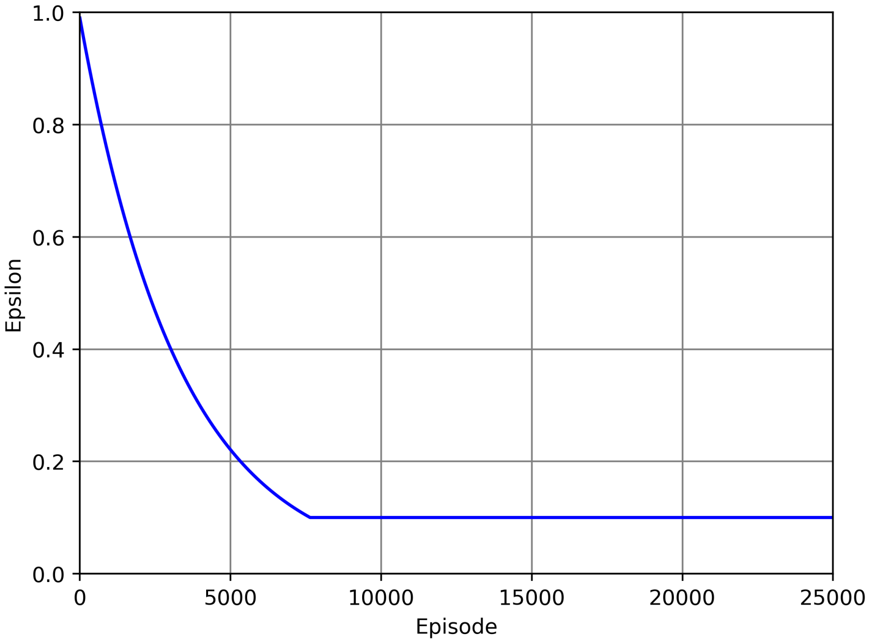
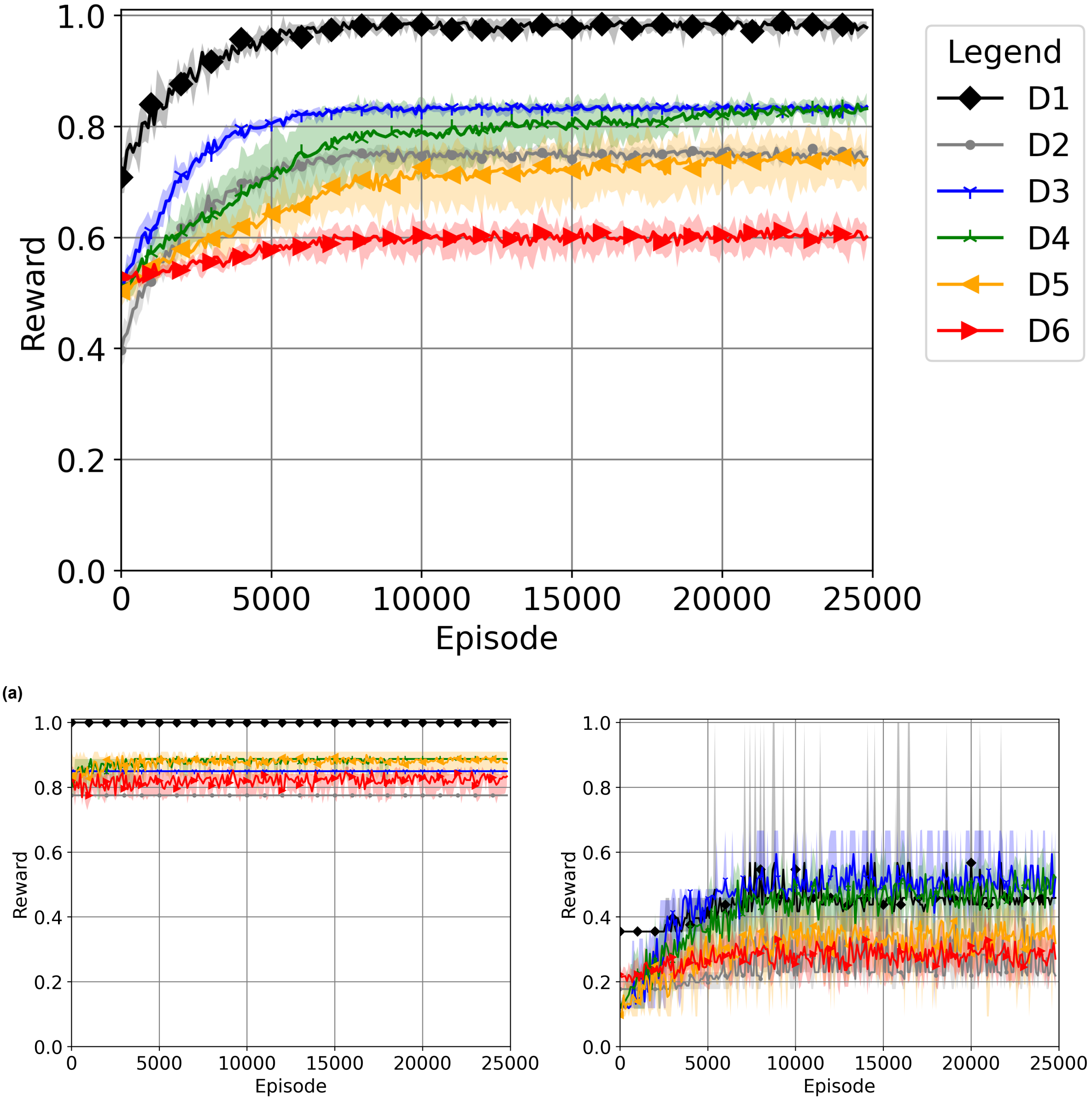


Figure 7.3: Epsilon decay over a training cycle

The first rewards graphs that are shown are the overall rewards that the agent obtains throughout a training cycle. The overall reward is the reward that is used by the agent for determining the Q-values and updating the Q-table accordingly. These rewards therefore also determine the actions that the agent takes. The overall reward is determined by summing all the individual subsystem rewards and dividing this by the number of individual rewards. At the moment, the individual rewards are not weighted and therefore influence the overall reward equally. The overall rewards are shown in Figure 7.4a to Figure 7.4c.



(a)

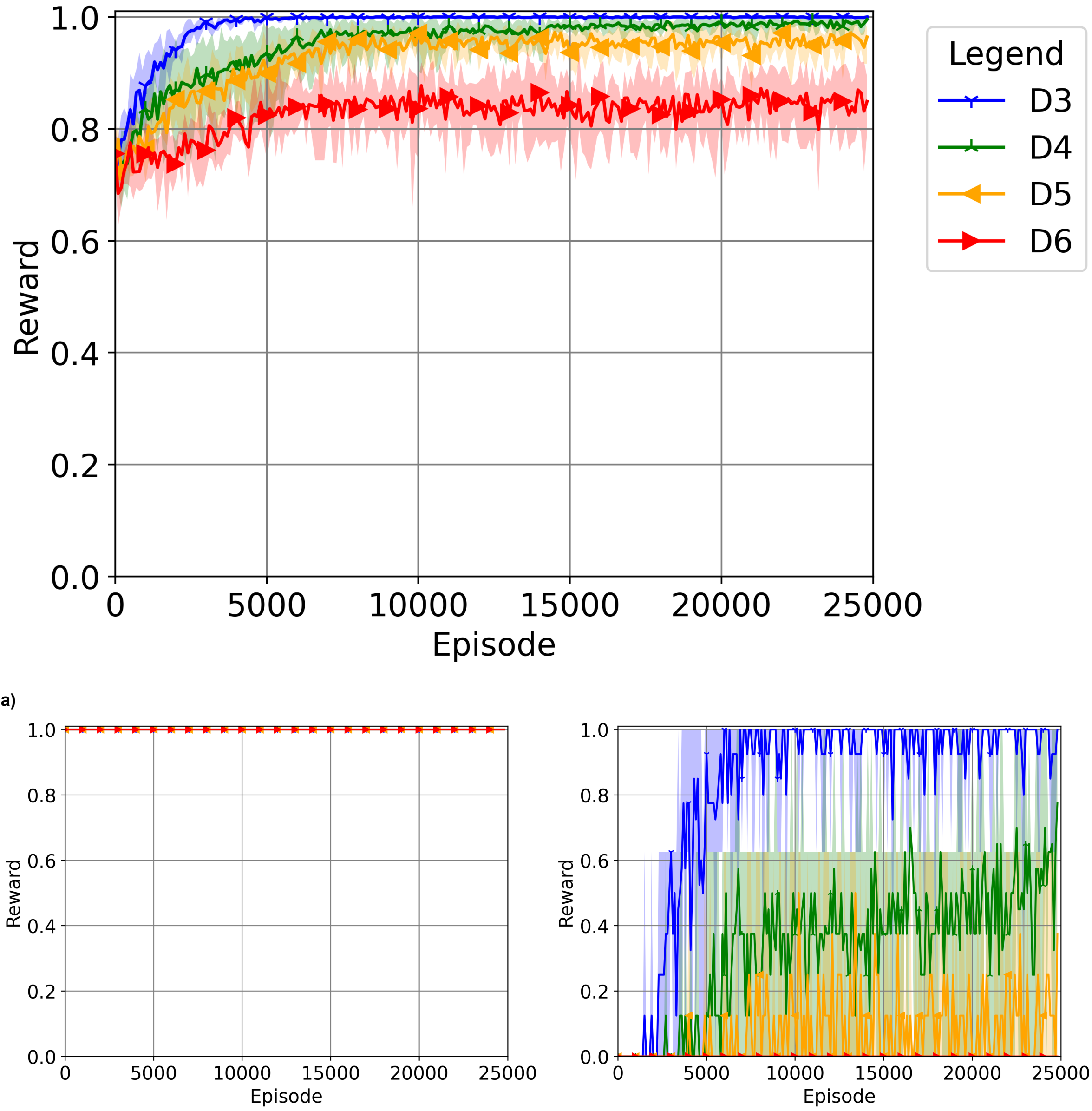
(b)

(c)

Figure 7.4: Overall rewards as a function of episode number for increasing design tool dimension with (a) Average reward with variation. (b) Maximal reward with variation. (c) Minimal reward with variation.

Next to the overall rewards, the individual subsystem rewards can also be shown. Since these combine into overall rewards, a similar behaviour is expected in these graphs. The subsystem rewards for the AD reward can be seen in Figure 7.5a to Figure 7.5c. The subsystem rewards for the other subsystems show similar patterns and are therefore not shown.

In Figure 7.4a, it can be seen that the overall reward that is obtained at the end of a training cycle decreases as the dimension of the design tool increases. The increase in reward at the end of training



(a)

(b)

(c)

Figure 7.5: Attitude Determination subsystem rewards as a function of episode number for increasing design tool dimension with (a) Average reward with variation. (b) Maximal reward with variation. (c) Minimal reward with variation.

cycle as compared to the reward at the beginning diminishes for increasing dimensions. This shows that the agent learning capability is decreasing for increasing complexity, since the agent is still able to theoretically achieve a higher reward, as can be seen in the maximal possible rewards in Figure 7.4b. For some reason, the agent is not able to end up with configurations that provide such higher rewards. This can be explained by looking at Figure 7.6. This figure shows the number of times a AD component is selected during a training cycle. While the agent has a high tendency to select a specific component for the low dimensions, this tendency decreases as the design tool's dimension increases. This means that the agent is more likely to select lower rewarding components for higher dimensions, which in turn reduces the overall reward. As this behaviour can be observed for every component type, this explains the decrease in performance that is observed in Figure 7.4a.

It is expected that this behaviour is caused by the increase in complexity, which increases the number of state-action combinations. This means that the number of actions that would be required to traverse the design space also increases. Since the number steps that the agent can take during an episode remains the same, this means that the agent is less 'mobile' in the design space. In other words, when the agent is initiated randomly, it might be the case that due to the randomness factor present in the training cycle, the agent is out of bounds of certain components.

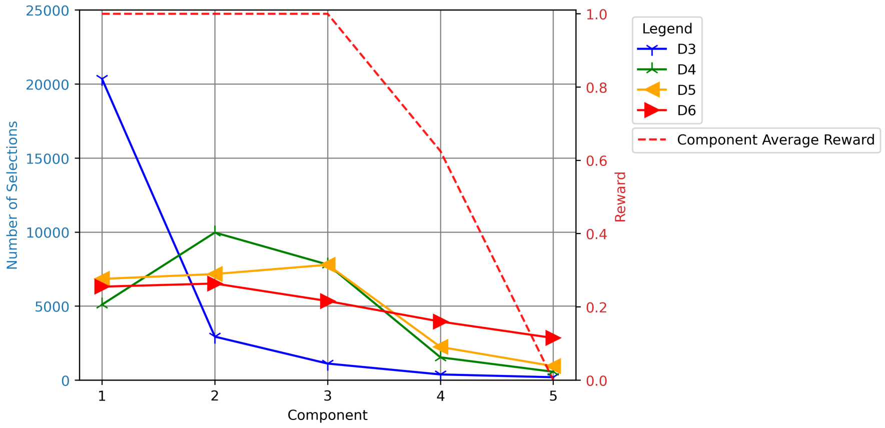


Figure 7.6: Number of selections of the attitude determination components for increasing design tool dimension

7.2. Design Tool Results

The results of section 7.1 showed that as the complexity of the design tool's environment increased, the design tool is less likely to select high rewarding components during the training cycle. The performance of the design tool in its final setup constructed in chapter 5 is now investigated in more detail. In this section, results are shown with which it is aimed to investigate whether the design tool is performing as desired. Because it is desired to use the design tool with a trained agent to enable the re-use of knowledge, the results that are obtained in a training cycle and in a cycle with a trained agent are compared. To use a trained agent, the Q-table from a training cycle is saved and re-used in the same environment and allowed to fully exploit the design space. This means that there is no longer any randomness present in the design tool, the epsilon is immediately set to 0. Next to that, the cycle is shortened to 5000 episodes because it is expected to be sufficient to cover every possible path that the Q-table dictates the agent to take. The settings of the design tool are similar to the settings shown in subsection 7.1.1.

In Figure 7.7, the number of times a certain payload component is selected during a training cycle is indicated by the blue line, the payload reward for the payload components is indicated by the red dotted line, and the variation in overall reward obtained in configurations using the payload component is indicated with the boxplots. The same is plotted for the use of a trained agent in Figure 7.8. What can be seen in these figures is the correlation between the number of times the agent selects a component and the overall reward that can be obtained with that component. It is expected that when the design tool is used with a trained agent, the agent will select the component that is able to achieve the largest overall reward. In Figure 7.7, it can be seen that during a training cycle, the agent does not necessarily select the component with the highest possible overall reward, which in this figure would be component 3. This is accredited to the fact that in a training cycle, a certain factor of randomness is present. However, Figure 7.8 shows that the design tool that uses this trained agent actually tends to select component 1, which is not according to expectations. The reason for this discrepancy is thought to be due to the size of the state-action combination space. The size of this space makes it difficult for an agent to traverse it to an optimal region since the agent is initialised at a random location in the design space. This might cause the agent to get stuck in local optima, something which was also encountered in chapter 4. The solution to this problem in that scenario was to increase the discount in the agent's settings. This approach is attempted again, the discount factor is changed from 0.95 to 0.99 in order to further increase the importance of the final reward at an episode for the agent.

It should also be noted that the boxplots indicate a greater variation in overall rewards that are obtained during a training cycle as opposed to the rewards that the trained agent obtains, Figure 7.7 versus

Figure 7.8. This is explained by the fact that during a training cycle the agent is allowed to explore the design space which will result in a greater variation in the components that are selected at the end of an episode. When a trained agent is used, this exploration is removed and the agent will therefore move straight towards the state that can provide the optimal reward. The results of a training cycle can tell the user of the design tool more on the sensitivity or the flexibility of components relative to the results that are obtained when a trained agent is used.

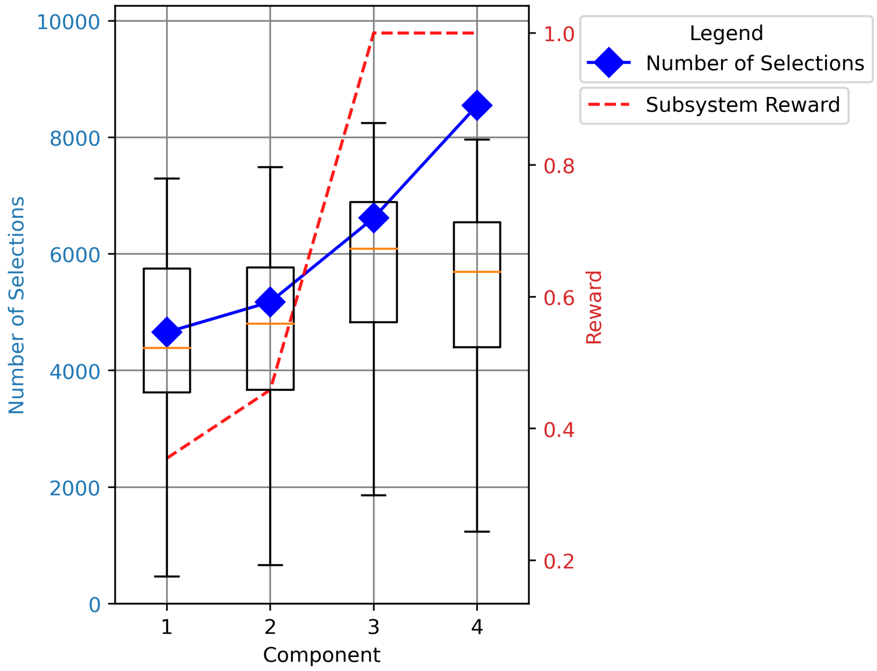


Figure 7.7: Payload subsystem training cycle: Number of times a component is selected and overall and subsystem reward.

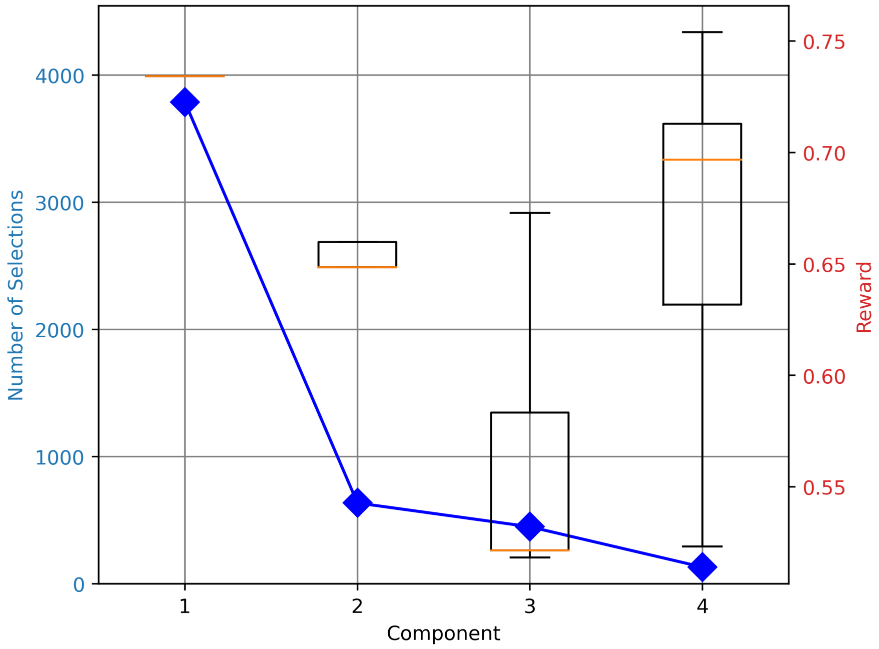


Figure 7.8: Payload subsystem trained agent: Number of times a component is selected.

The same figures as were previously shown for the design tool with a discount factor of 0.95, Figure 7.7

and Figure 7.8, are also presented for the tool with a discount factor of 0.99. These results are shown in Figure 7.9 and Figure 7.10. In Figure 7.11a to Figure 7.11c, the difference between the obtained overall reward for the design tool with a discount factor of 0.95 and 0.99 during a training cycle can be seen. The design tool with a discount factor is able to achieve a higher overall reward at the end of the training cycle, which is shown in Figure 7.11a.

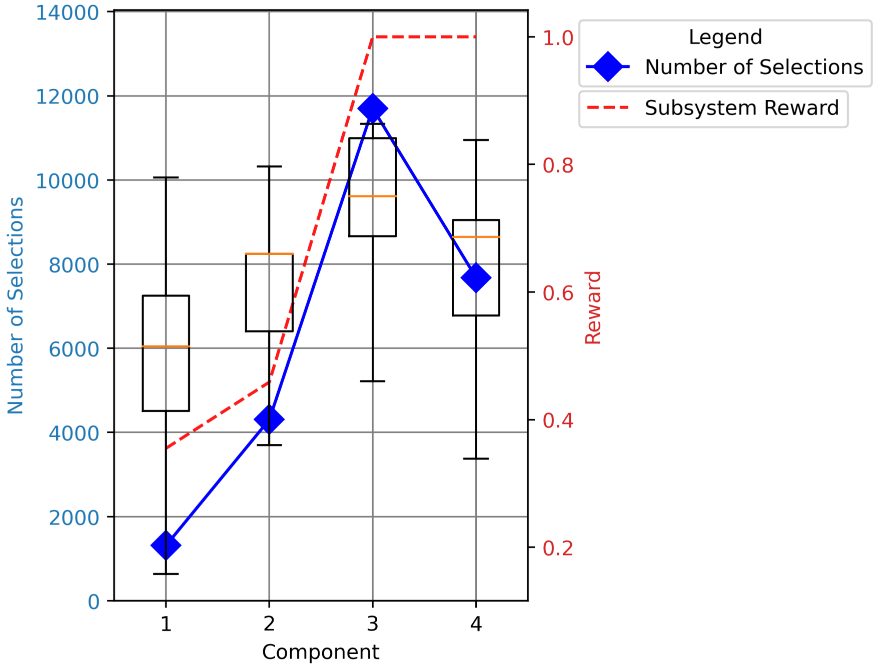


Figure 7.9: Payload subsystem training cycle with a discount factor of 0.99: Number of times a component is selected and overall and subsystem reward.

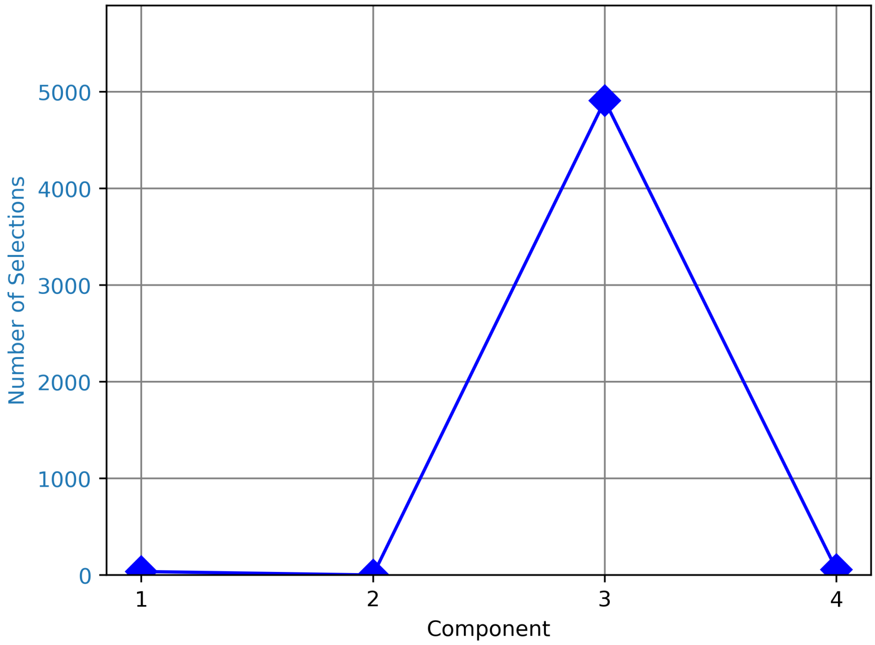
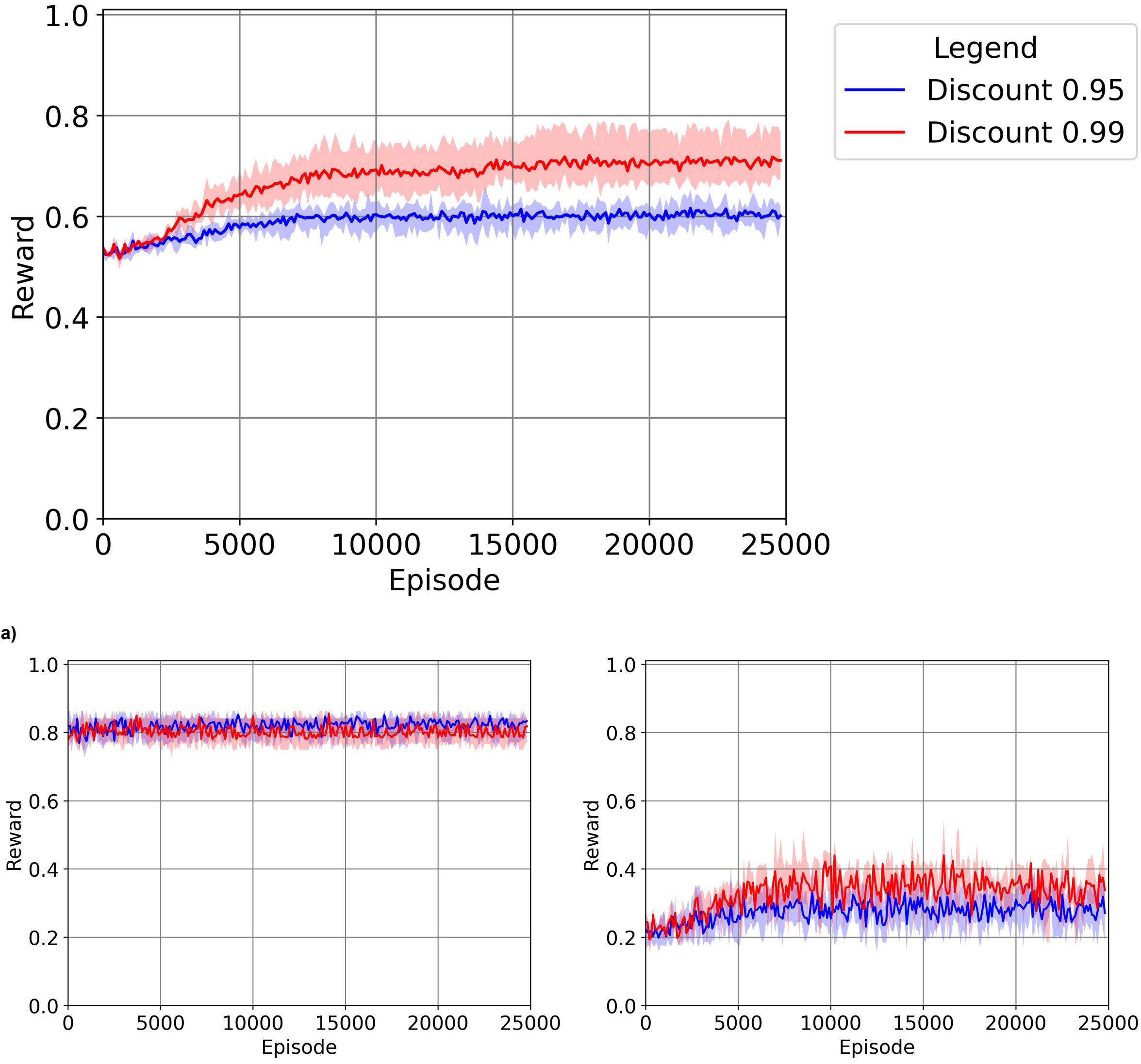


Figure 7.10: Payload subsystem trained agent with a discount factor of 0.99: Number of times a component is selected.

In previous figures, the performance of the design tool has been shown in terms of obtained reward throughout a training cycle and the selected component for component types. To show whether the de-



(a)

(b)

(c)

Figure 7.11: Overall rewards comparison between training the design tool with a discount factor of 0.95 and 0.99 (a) Average reward with variation. (b) Maximal reward with variation. (c) Minimal reward with variation.

sign tool's capability of assessing concepts works as intended, the sensitivity of the generated concepts for varying components can be investigated. Because of the complexity of the problem, it is difficult to visualise these results of the design tool in a single figure. Using different figures that present overlapping results, the reading of the results can be improved. These results could also show something about the sensitivity of certain design options. The CubeSat concept parameters that are dependent on the selected components and will be investigated are listed below. The investigations should show that variations in the parameters indeed influence the outcome of the concept generation as desired.

- Size: The size of the system directly depends on the components that are selected by the agent, as explained in subsection 5.1.8.
  - Mass: The mass of the satellite is estimated based on the size, as explained in subsection 5.1.8.
  - Attitude Control Capability: The required torque and momentum capabilities of the AC component depends on the size and mass of the satellite.
  - Power Generation: The power that can be generated by the satellite depends on the amount

of surface area that is covered by solar panels and therefore depends on the size of the satellite.

- Power Consumption: The power consumption of the satellite directly depends on the components that are selected by the agent, as each component has its specific power consumption.
- Depth of Discharge: The DoD depends on the power generation and power consumption, and thereby depends on the selected components.

In Figure 7.12, the variation of the overall reward as a function of the concept size is shown using boxplots of the overall reward. This figure indicates the overall reward that can be obtained with various satellite sizes. It also shows the distinct satellite sizes that the design space results in. The variation in size of a concept using for the payload components is shown in Figure 7.13. In this figure, it can be seen that the concept size is dependent on the payload component that is selected. The payload components become bigger as the component number increases, meaning component number 1 is the smaller camera and number 4 the largest. The influence of the payload size is visible in Figure 7.13, as the concepts that use the larger payload components result in larger satellite sizes. The variation for component number 3 is due to the variation in the other components of the concepts.

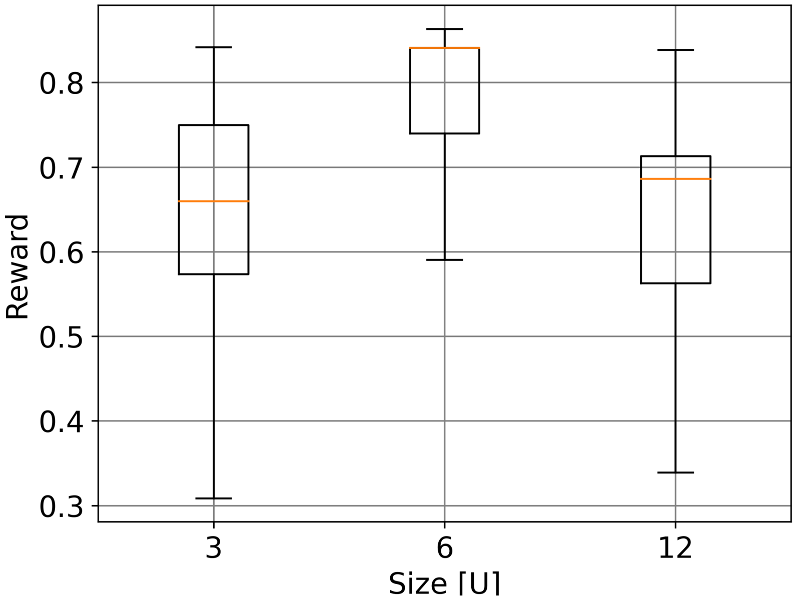


Figure 7.12: Overall reward variation as a function of concept size.

The variation in mass as a function of concept size is illustrated in Figure 7.14. It can be seen that there is no variation in the concept mass for a specific concept size. This is as expected, due to the way in which the mass of the satellite is estimated, as explained in subsection 5.1.8. Similar patterns are encountered in the variation in the required torque and momentum capabilities for varying concepts. The expected behaviour that the required torque and momentum increase as the concept size increases are shown in Figure 7.15 and Figure 7.16, respectively.

The variation in generated power as a function of satellite size is shown in Figure 7.17. It can be seen that there is no variation in generated power for specific satellite sizes. This is expected, because the amount of generated power depends on the total area that is covered by solar panels. As this area depends on the surface area of the satellite, no variation in generated power exists for a specific satellite size.

The other concept parameters that depend on the selected component set are the power consumption and the DoD, as this is in term dependent on the power consumption. The variation in these parameters as a function of satellite size is shown in Figure 7.18 and Figure 7.19, respectively. The power con-

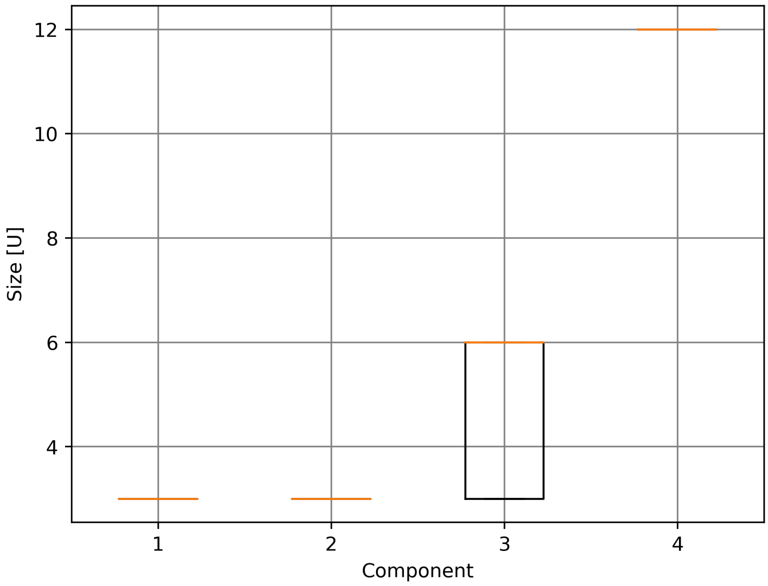


Figure 7.13: Variation in concept size for payload components.

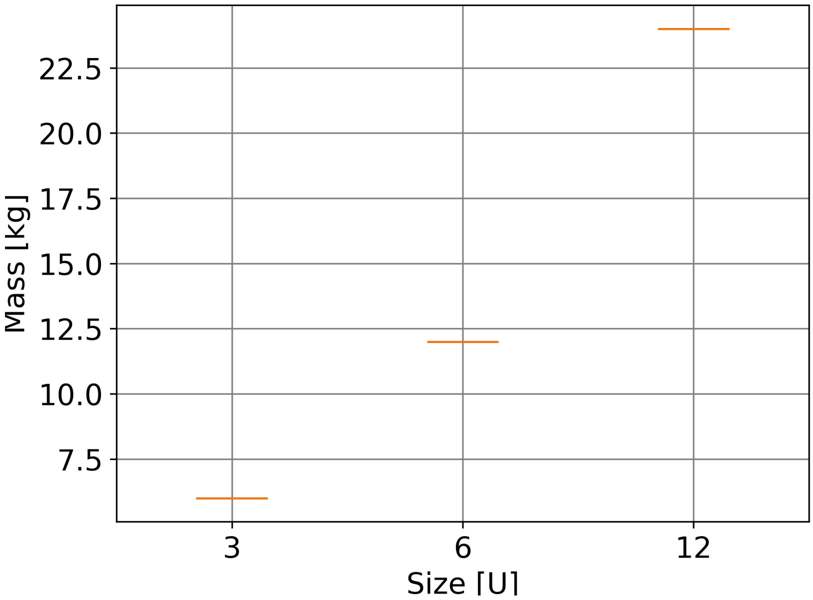


Figure 7.14: Variation in concept mass for concept size.

sumption varies significantly for each satellite size, with a generally higher possible power consumption for larger satellite sizes. As the DoD depends on the power generation and consumption, a significant variation can be seen as well. This shows that the power consumption and generation of a size is dependent on the concept, and that the design tool is taking this into account.

It is also interesting to show the dependency of such parameters in the subsystem level. An example of this is shown in Figure 7.20, where the variation of overall reward for the payload components is plotted. The rewards are separated by size and the variation is shown using boxplots for every possible size. This figure shows that using payload components 1 and 2 will always result in a 3U size, while it is

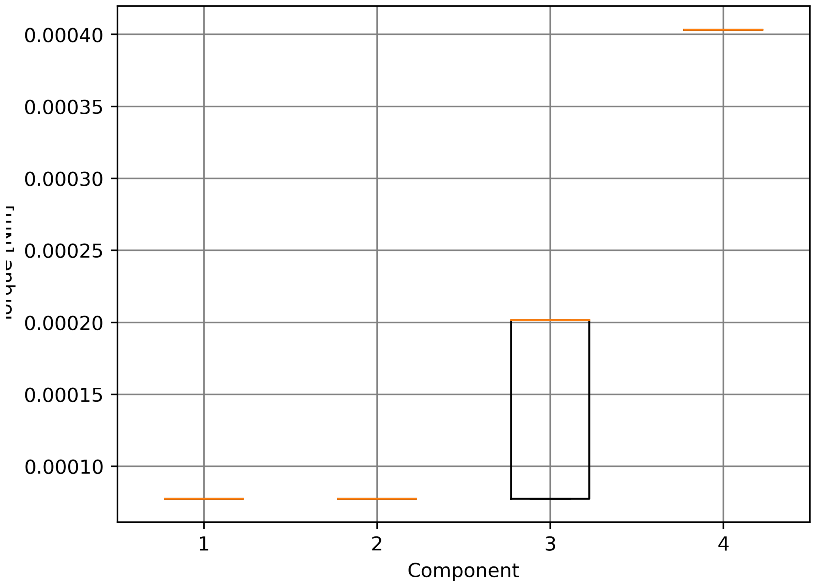


Figure 7.15: Variation in required torque for concept size.

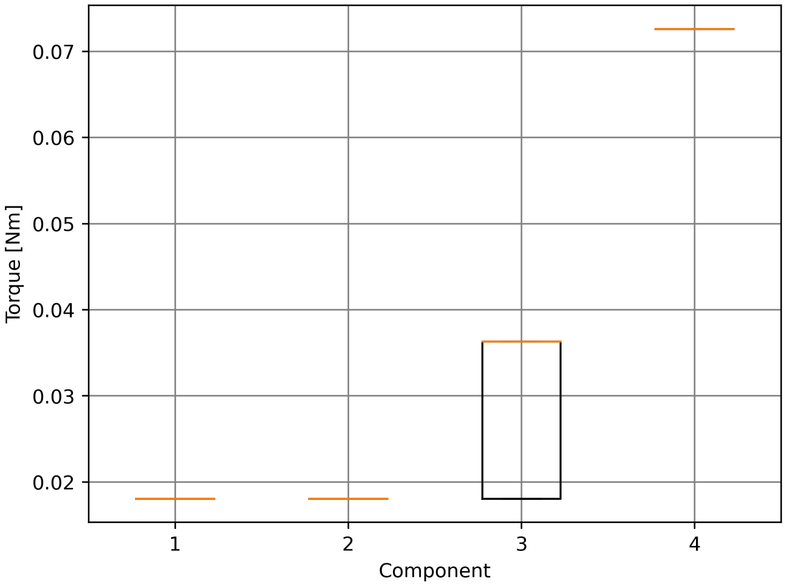


Figure 7.16: Variation in required momentum for concept size.

possible that component 3 is used in both a 3U and 6U satellite. For component 3 however, the 6U satellite is able to achieve better overall rewards, as indicated by the boxplot. Meanwhile, component 4 always results in a 12U configuration. Overall, the 6U configuration using component 3 can result in the best overall reward. This is confirmed by Figure 7.12, which also shows that the 6U configuration can results in the best overall reward. These results have shown that the design tool is able to analyse the selected concepts correctly and that the analyses have provided expected behaviours. In chapter 6, the results will be expanded in order to explain in more detail how the optimal configuration can be identified using the design tool.

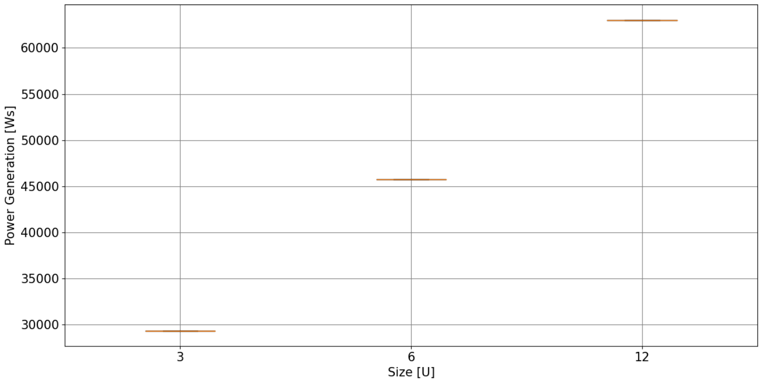


Figure 7.17: Variation in generated power for concept size.

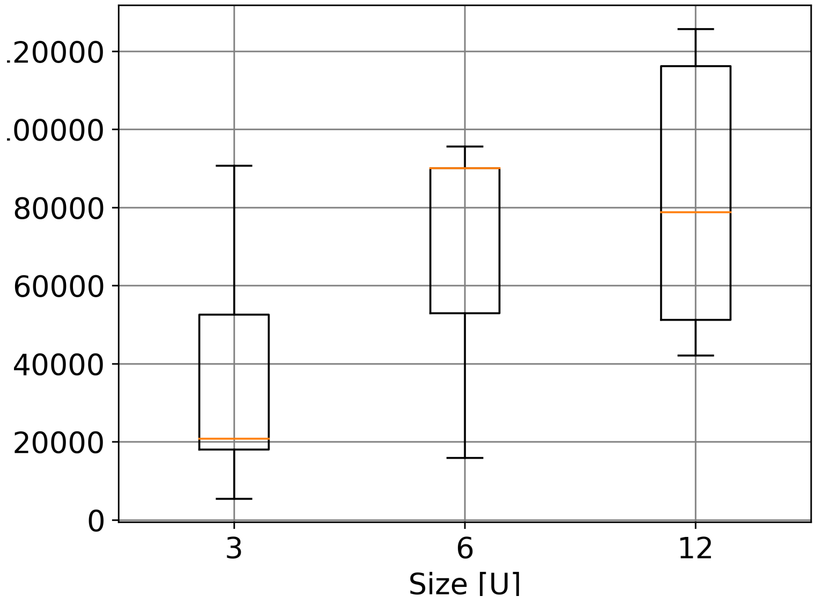


Figure 7.18: Power consumption variation as a function of satellite size.

7.3. AltiCube Case Study

In section 7.2, it is shown that the design tool functions as intended, by taking the influence of system parameters into account correctly. These results however, do not indicate whether the design tool can identify a CubeSat concept that is actually feasible for the mission requirements. To provide this analysis, a validation case study was set up in chapter 6. In this section, the results of the AltiCube case study will be presented and discussed. First, in subsection 7.3.1 the settings of the design tool will be given, after which the results are presented and discussed in subsection 7.3.2. The results are presented in terms of the components that a trained agent selects for the entered mission requirements for each component type.

7.3.1. AltiCube Design Tool Settings

The same settings for the design tool have to be provided as were listed in subsection 7.1.1. The settings have to be reformulated for the AltiCube case study and are therefore listed again in this section.

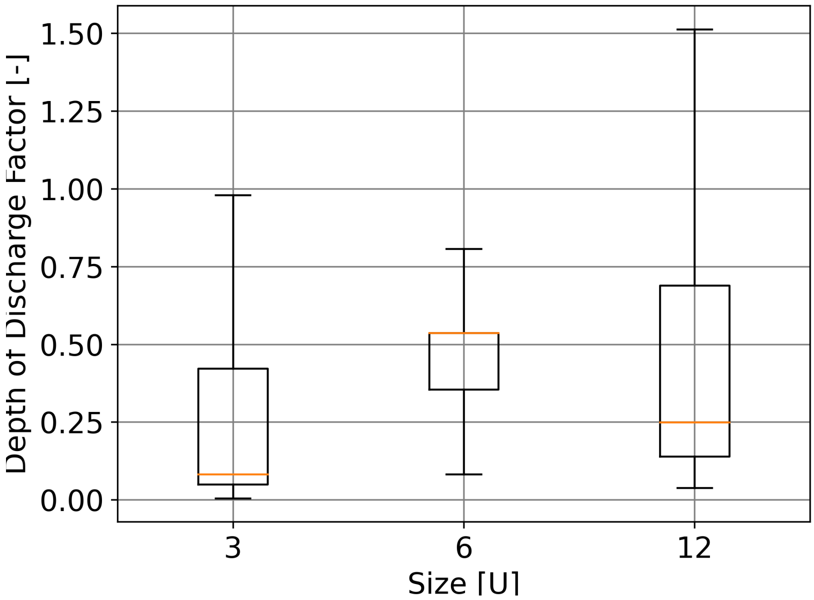


Figure 7.19: Depth of Discharge variation as a function of satellite size.

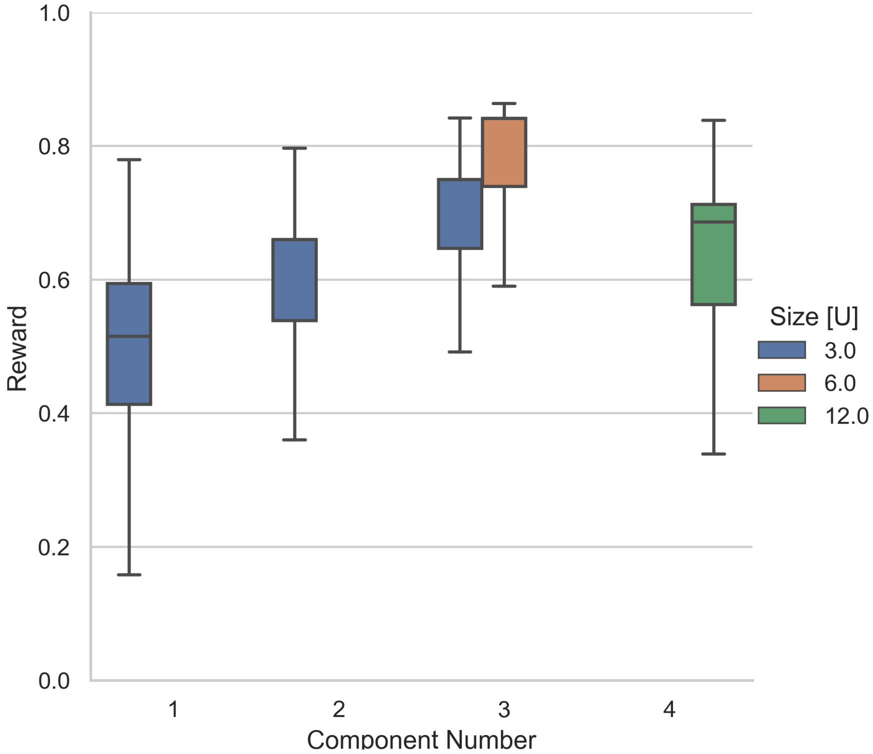


Figure 7.20: Overall reward variation for the payload components, separated by size

Component Database

The component database that is used for the AltiCube case study differs from the component database of chapter 5. The database is shown in section A.2. The agent is able to select components from this database for the generation of concepts.

Algorithm Settings

The agent setting that are used during the generation of concepts for the case study are shown in Table 7.7. These settings are based on the results of section 7.2.

Table 7.7: Case Study Q-learning algorithm settings

Setting	Value
Number of Episodes	25000
Number of Iterations	100
Epsilon Initial	0.99
Epsilon Decay	0.9997
Epsilon Minimal	0.1
Learning Rate	0.3
Discount Factor	0.99

7.3.2. Case Study Results

The modified design tool is used to generate results for the settings that were described in subsection 7.3.1. The design tool is able to select components for 7 different component types. The component database is provided in Appendix A. The goal of running the design tool for these settings and in this environment is to be able to select optimal components for every component type. Components will be selected for every component type in this section. After that the selected configuration is compared to the AltiCube system design in section 6.1. For the identification of the optimal configuration of components, it is not required to look at the graphs that contain the rewards over a training cycle. These graphs indicates the training performance of the design tool's agent, but do not indicate what components the agent tends to select and what kind of rewards the components offer. This section will first present some overall system results, after which the component types are treated consecutively.

AltiCube Design Tool System Results

In Figure 7.21, the overall reward as a function of the possible satellite size is shown. A satellite size of 20U shows to be the configuration that can achieve the highest overall reward. Next, Figure 7.22 presents the DoD as a function of satellite size. The configurations of components that result in a 24U satellite generally lead to a DoD over 100%, meaning the batteries are completely drained and making the configurations infeasible. The torque and momentum requirements on the system are shown in Figure 7.23 and Figure 7.24, respectively. These figures show that the torque and momentum requirements increase identically for increasing satellite size, which makes sense due to the used expected mass estimation, which uses the assumption of 2 kg per unit.

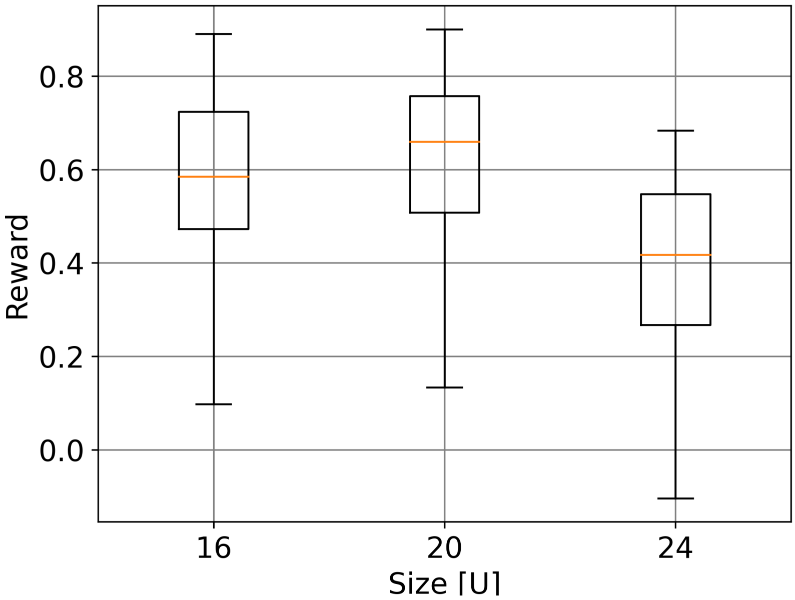


Figure 7.21: Overall reward variation as a function of satellite size.

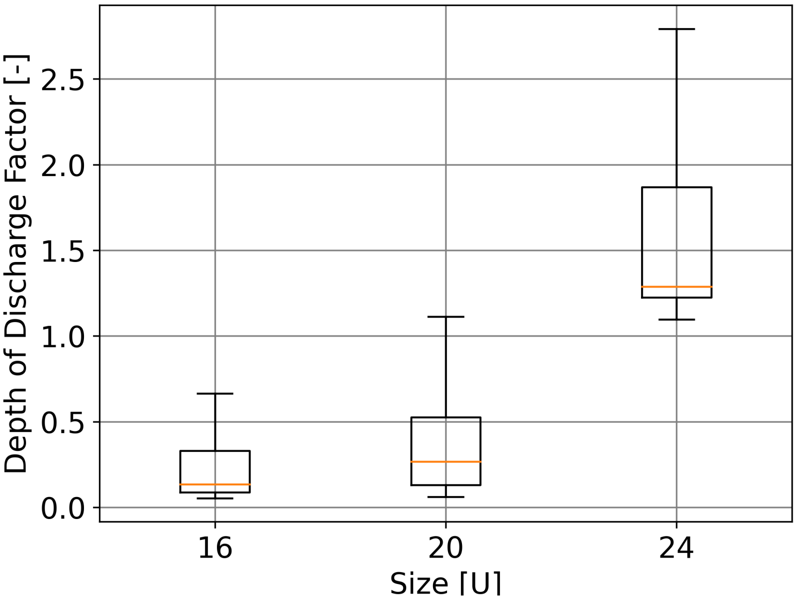


Figure 7.22: Depth of Discharge variation as a function of satellite size.

AltiCube Component Type Results

In Table 7.8, the maximal component type and system rewards that the agent obtained for each component of every component type are shown. For every component type, the best performing components are highlighted. It can be seen that multiple components for the same component type provide similar type and system rewards. These numbers are obtained during a training cycle during which the design space is explored. These highlighted components therefore do not necessarily indicate the components that a trained agent will select. It is envisioned that a trained agent will have the tendency to prefer a single component for each component type. It will be interesting to see which components a

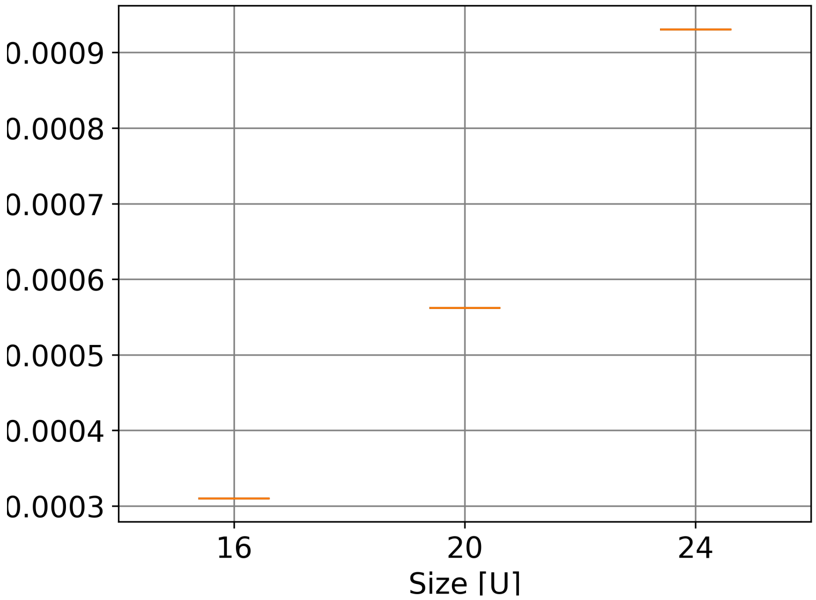


Figure 7.23: Torque variation as a function of satellite size.

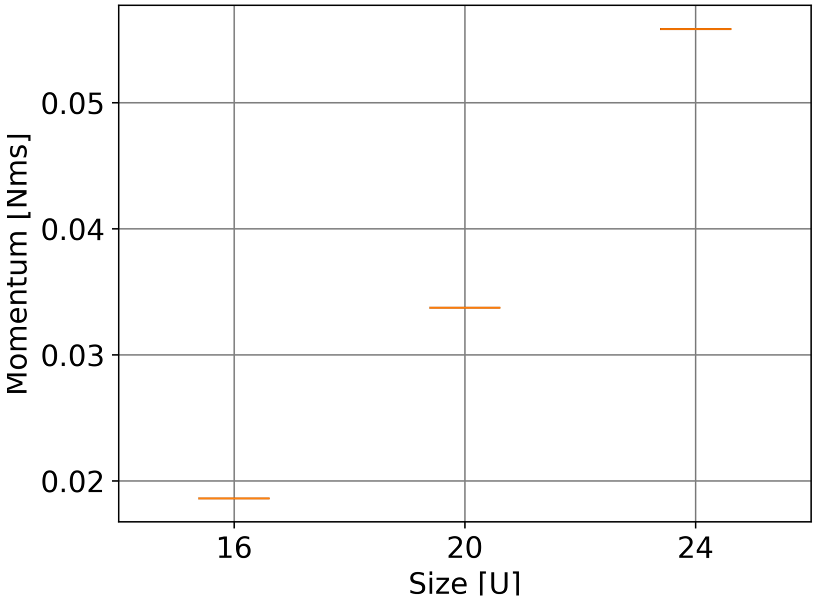


Figure 7.24: Momentum variation as a function of satellite size.

trained agent is likely to select. These results will be presented hereafter.

Telemetry, Tracking & Command Results

The first subsystem that is treated is the TTC subsystem. This subsystem consists of two component types that can be selected by the agent, namely the antenna and ground station. Note that the location of the ground station is fixed. The number of times a component is selected by a trained agent for both components are shown in Figure 7.25 and Figure 7.26. The components that are selected for the antenna and ground station component types are for both types component 4. This coincides with the

Table 7.8: AltiCube case study maximum component type and system rewards

Component Type	Component Number	1	2	3	4	5
Antenna	Type Reward	0.35	0.00	0.50	0.55	0.10
	System Reward	0.86	0.79	0.89	0.90	0.81
Ground Station	Type Reward	0.35	0.00	0.50	0.55	
	System Reward	0.86	0.79	0.89	0.90	
Attitude Determination	Type Reward	1.00	1.00	0.90	0.50	0.00
	System Reward	0.90	0.90	0.89	0.81	0.71
Attitude Control	Type Reward	0.02	0.36	0.77	1.00	1.00
	System Reward	0.71	0.76	0.84	0.90	0.68
Power Control Unit	Type Reward	0.00	1.00			
	System Reward	0.70	0.90			
Battery	Type Reward	0.91	1.00	1.00	1.00	
	System Reward	0.79	0.85	0.90	0.90	
DSP	Type Reward	1.00	1.00	1.00	1.00	
	System Reward	0.80	0.90	0.90	0.90	

results that are shown in Table 7.8.

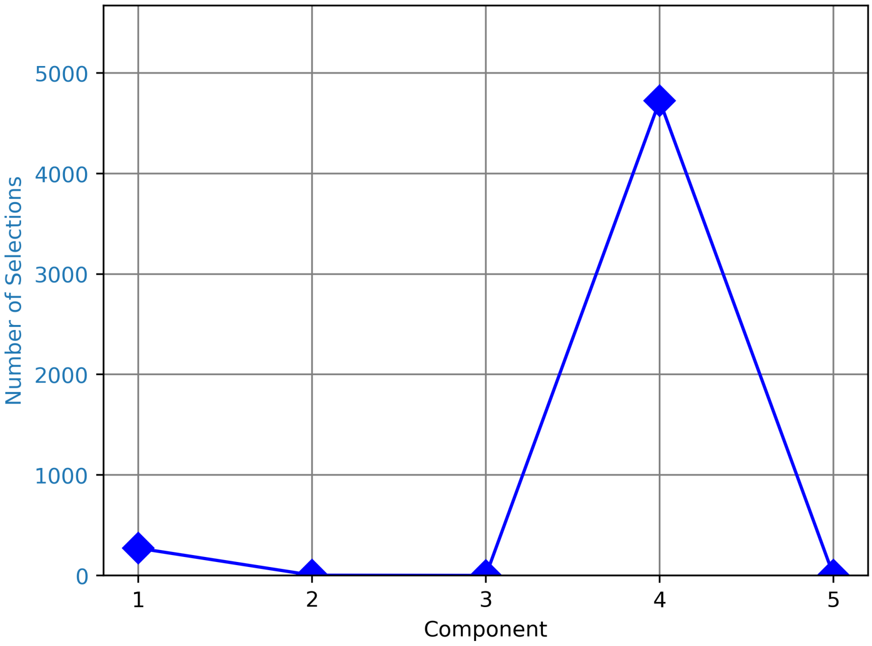


Figure 7.25: Number of times an antenna component is selected by a trained agent.

Attitude Determination Results

The second subsystem, which is more accurately a subsystem of the ADCS, is the AD system. This subsystem consists of a single component type that has to provide the fine AD capabilities for the AltiCube mission. The number of times a trained agent selects a component is shown in Figure 7.27. The trained agent almost exclusively selects component number 1, which is one of the two best performing components in Table 7.8.

Attitude Control Results

The AC system consists of a single component type for which the agent can select a component. The results are shown in Figure 7.28 and show that component 4 is selected as the AC system component, which confirms the results of Table 7.8.

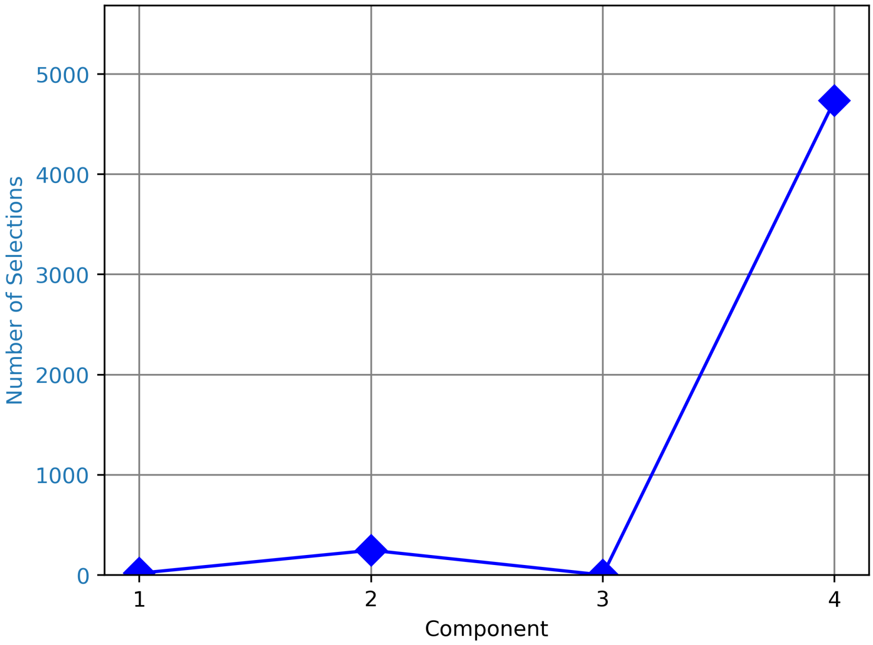


Figure 7.26: Number of times a ground station component is selected by a trained agent.

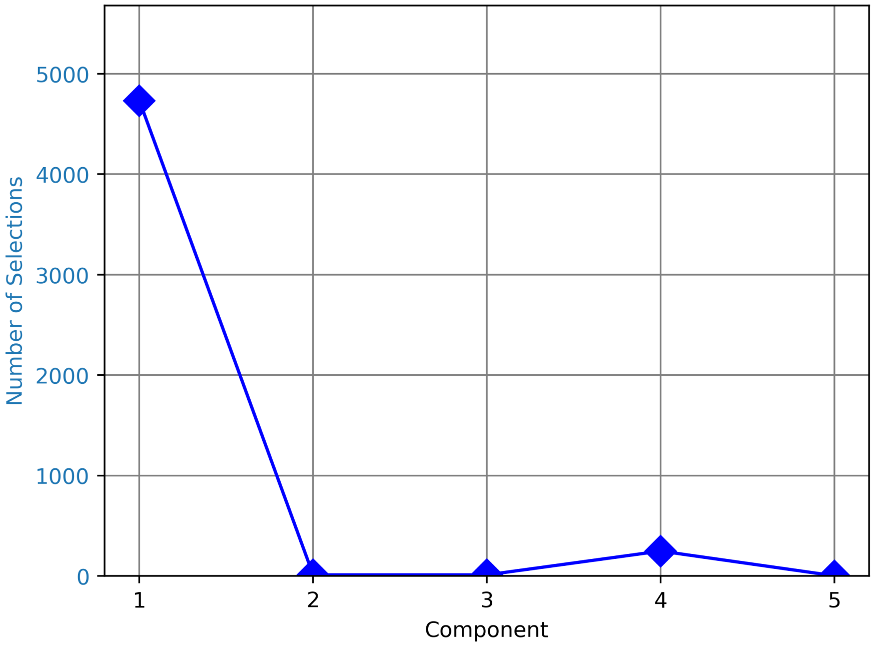


Figure 7.27: Number of times an attitude determination component is selected by a trained agent.

Power Control Unit Results

The results of the PCU component type are presented in Figure 7.29. The component option does not have significant influence on the size of the satellite, as both options are used in every satellite size configuration. The clear optimal PCU component is component 2, again confirming the results of Table 7.8.

Battery Results

In Figure 7.30, the number of times a battery is selected is shown. This figure illustrates that component 4 is the most selected component, which is one of the two top performing components of Table 7.8. In

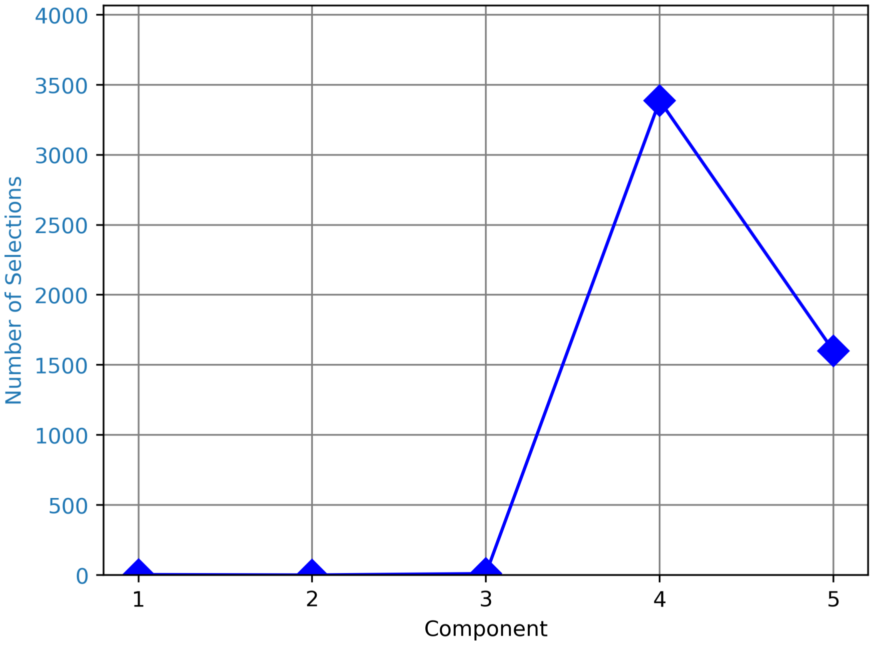


Figure 7.28: Number of times an attitude control component is selected by a trained agent.

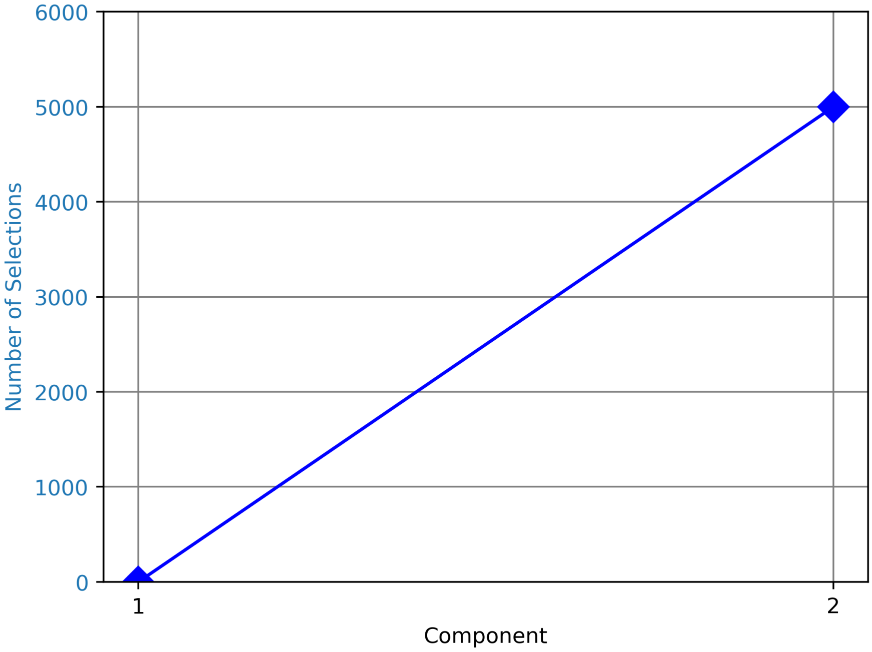


Figure 7.29: Number of times a power control unit component is selected by a trained agent.

Figure 7.31, the variation in DoD for the battery components is given. Since the batteries have a higher capacity as the component number increases, the results that this figure shows can be expected. The expected outcome is that the higher the capacity of the battery, the lower the DoD will be. This is confirmed by Figure 7.31, where it can be seen that battery component 4 has the lowest possible DoD. This does however not influence the reward that the agent obtains for the battery, since the reward is capped at 1 if the component meets the required DoD of 20%. This explains why in Table 7.8 the maximal rewards for battery components 3 and 4 are identical.

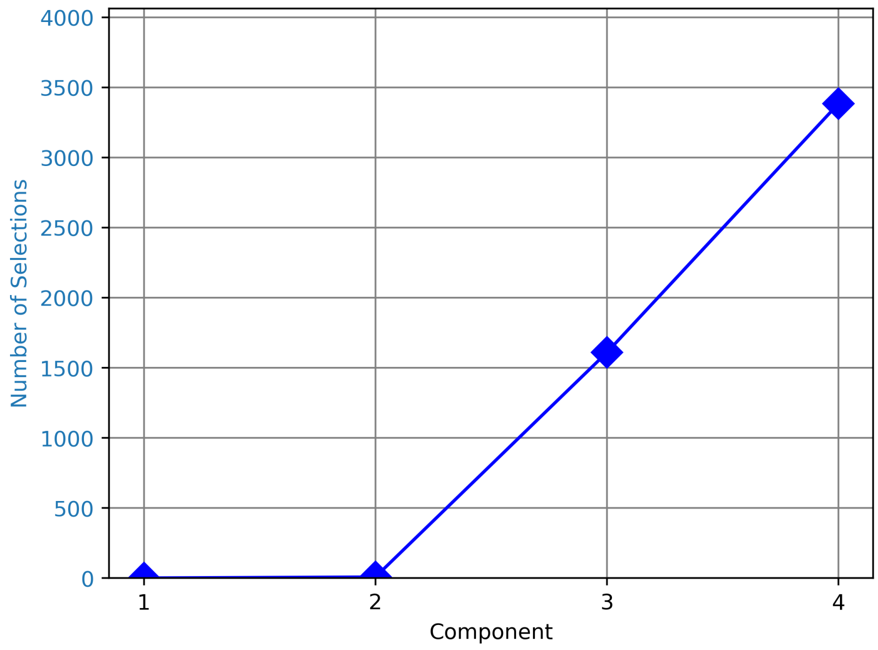


Figure 7.30: Number of times a battery component is selected by a trained agent.

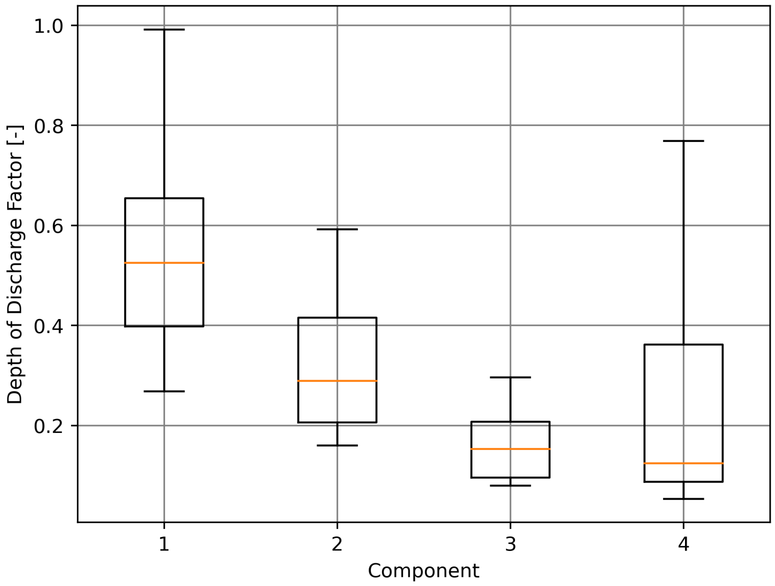


Figure 7.31: Variation in depth of discharge for battery components.

Deployable Solar Panel Results

In Figure 7.32, again the number of times that a trained agent selects a configuration option is shown. This figure shows that the DSP configuration 4, which is the 3x DSP factor, is most often selected. The variation in DoD for the different DSP configurations can be seen in Figure 7.33. In Table 7.8, it can be seen that the DSP components 2-4 have identical maximal rewards. To be more precise for the system rewards, DSP component 2 has a maximal system reward of 0.9000, while components 3 and 4 have an identical maximal system reward of 0.9002.

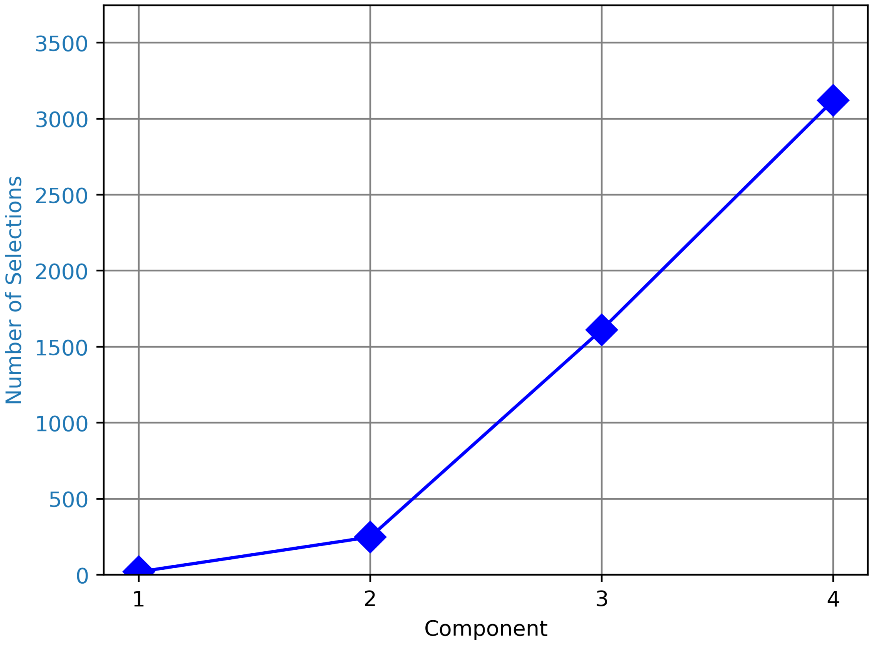


Figure 7.32: Number of times a deployable solar panel configuration is selected by a trained agent.

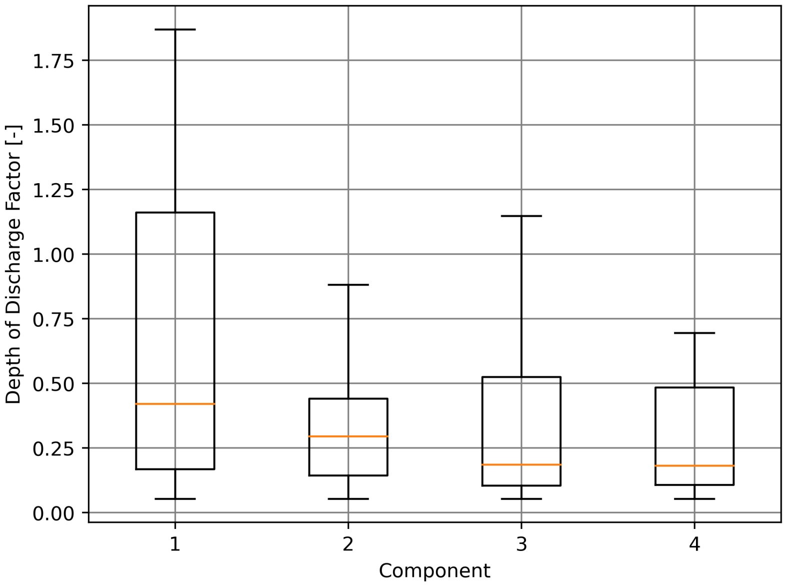


Figure 7.33: Variation in depth of discharge for deployable solar panel components.

7.3.3. System Design Comparison

The configuration that returns the maximal system reward obtains a reward of 0.9002. This reward is obtained by two configurations that differ only in the DSP component type selection. The configurations in terms of component numbers looks as follows: antenna 4, ground station 4, attitude determination 1, attitude control 4, PCU 2, battery 4 and either DSP 3 or 4. This configuration confirms the components that were selected by the trained agent.

If this configuration is compared to Table 7.8, it can be seen that the agent tends to select the ’best’ component option for each component type. For example, the agent selects AD component 1, with which a similar type and system reward is obtained as with component 2, while AD component 1 has a higher power consumption than component 2. Both components meet the requirements on the AD subsystem, so instinctively, it would be better to select component 2, as this decreases the power consumption of the overall concept. The reason why the trained agent selects component 1 can be seen when the obtained system reward is examined with an increased number of significant figures than is shown in Table 7.8. The concepts that use AD component 2 in combination with similar other components obtain a system reward of 0.9000, compared to 0.9002 for concepts with AD component 1. Because the agent aims to maximise the reward, the components that provide this maximal reward are selected, which in this case is AD component 1. In that sense, it is expected that the agent selects AD component 1.

Intuitively however, this does not make sense as AD component 2 has a lower power consumption which would improve the overall power consumption of the satellite. This showcases a limitation of the current version of the design tool. Because the reward function for the power consumption is semi-continuous, the reward that is returned is 1 when the concept meets the DoD requirement. When a configuration of components is able to meet the required DoD with varying components for a single component type, it can be deduced that the design tool has no clear capability to make a distinction between these varying components. This is to say that there is no clear advantage for selecting a specific component as all those components meet the requirements. This is demonstrated by the agent's selection for AD component 1 in stead of 2.

At the moment, the design tool makes the distinction between components based on marginally different system rewards. This aspect demonstrates that the design tool in its current version is not able to sufficiently take into account the system performance of the concept. There is no system in place that prevents the design tool from selecting concepts that are over-performing. Even though these concepts are feasible, they provide solutions that might not be optimal for the mission requirements, as these can be met by components that have lower specifications. In the current version of the design tool there is no penalty for such over-performing concepts. This observation will be further discussed in section 7.4.

The system design that results from the selection of components using the design tool is compared to the equivalent components of the AltiCube system design in Table 7.9. The table shows that the satellite that is designed by the design tool is a significantly heavier satellite, 25kg vs 40kg, even though the size does not show the same kind of difference, 16U vs 20U. The increase in mass can be explained by the assumption which the design tool uses that the mass of a CubeSat can be estimated by accounting 2kg per unit volume. The components that have been selected by the design tool are very similar to the components of the AltiCube system design, as can be seen by the key specifications of those components.

The exact similarity in the downlink and antenna could have been expected, as the X-band components that were entered into the component database are the components as present in the AltiCube system design. The design tool is still able to identify that an X-band communication system is required, which is the important aspect in the selection of the TTC components.

The attitude determination component that has been selected by the design tool has the same performance when compared to its AltiCube equivalent. AC is provided by reaction wheels in both system designs. The specifications of the attitude control components is similar for both system designs. It can therefore be concluded that the analysis that is done by the design tool provides an accurate assessment for the attitude determination and control component types.

The PCU component type is a relatively simple analysis in the design tool. The PCU should be able to accommodate the voltage range required by the configuration of components. As the component that is selected by the design tool provides a wider range than the AltiCube PCU component, and the component is the only component in the database that is able to provide such a wide range, this analysis is also performed well by the design tool. The design tool is able to identify the voltage range required by the configuration of components and select a PCU component accordingly.

Finally, the battery and DSP component types are combined into the analysis that formulates the DoD reward. It can be seen that the battery selection of the design tool has a somewhat larger capacity when compared to the AltiCube system design. Due to the difference in size, the solar panels are also

Table 7.9: AltiCube case study system design comparison

Specification	AltiCube System Design	Design Tool Selection
Dimensions	200x200x400 mm^3 (16U)	200x200x500 mm^3 (20U)
Mass	25 kg	40 kg
Downlink Antenna	X-band 13 dB 20 HPBW	X-band 13 dB 20 HPBW
Ground Station	X-band 30 dB 5.1 HPBW	X-band 30 dB 5.1 HPBW
Attitude Determination	Star Tracker 6 arcsec	Star Tracker 6 arcsec
Attitude Control	Reaction Wheels 2 mNm 20 mNms	Reaction Wheels 2.3 mNm 30.61 mNms
Power Control Unit	3-16V	3-18V
Battery	90 Wh	115 Wh
Deployable Solar Panels	2x 8U DSP	2x/3x 10U DSP

larger. The tendency of the design tool to select larger deployable solar panels and batteries is due to the fact that the design tool has no limitations on this. In other words, no penalties are given to the design tool for selecting over-performing batteries and deployable solar panels when these provide similar rewards.

The design tool is able to automatically trade-off between the individual subsystem rewards by looking for the best possible overall reward. In this way, it aims to find the optimal configuration. In the case study, it is observed that the selected components by the design tool partially match the components that are present in the AltiCube system design. The component database is still fairly limited, a total of 29 components are spread over 7 component types. The component database creates a total number of 16000 possible configurations of components. In the system design there is still a large discrepancy in terms of system mass, which is caused by the assumption on mass per unit for a CubeSat.

7.4. General Discussion

In previous sections, the results that can be obtained with the design tool are shown and are explained. In this section, a general discussion is performed to analyse the working of the design tool using the results of previous sections.

Link between in- and output

The results of section 7.2 and section 7.3 show that the design tool is able to identify the optimal components for each component type using a trained agent. The components are found to be optimal in terms of the total reward that can be obtained with configurations using those components. The subsystem rewards that are obtained with these 'global' optimal components is also shown. What these results do not directly show is the performance of the components for the input. The main output of the design tool can be seen as the rewards that are obtained by the agent and the Q-table which is essentially the agent. The Q-table can then be re-used for other runs of the design tool and will influence the components that are selected. Other output, such as the size of the satellite are also stored for subsequent analysis. At the moment, the direct performance, such as achieved GSD, is not stored. Therefore, no exact result is given for the performance of the optimal components for the component's requirements. In other words, the output of the tool does not tell directly what the best performing configuration can actually achieve in terms of real-life parameters such as GSD. Instead, the performance is shown in terms of rewards.

In figures such as Figure 7.26, the subsystem reward is indicated by the red dotted line. With this line, the performance of the component for the component specific requirement is illustrated. Using this data with the reward system figures, it can be deduced what the performance of the component is in real-life parameters. Not all component types have a fixed subsystem reward per component however, because the rewards can depend on other factors as well. For example, for the AC subsystem reward, the size influences the performance of the selected component. Because the size is dependent on the complete configuration of components, the subsystem reward will vary as well.

It can be seen that it becomes quite complex to show the performance in varying parameters of every option in the design space. For this work's purposes, it is desired to indicate the performance of the design tool, for which the reward figures provide a suitable analysis. For usability purposes, it is deemed possible to adapt the design tool to show the desired performance parameters of any configuration of components.

System performance

The design tool is set up as such that it aims to obtain the best reward possible. The reward that is optimised is made up of the subsystem rewards. The reward that is optimised will thereby optimise the subsystem performance. A question is where the system performance is taken into account and whether this is sufficient.

At the moment, the system as a whole is taken into account most directly by the reward that is returned to the battery, which is based on the DoD. The DoD is based on the power consumption, which is unique for every component, and the power generation, which is based on the size (and DSP, if applicable). An issue of this method for this version of the design tool is that a larger satellite will mean more power consumption which will return a better DoD reward, if the power consumption remains similar. The only aspect of the design tool that checks this behaviour is the AC reward, because a larger satellite will result in more strict AC requirements. Determining whether this is a balanced system can only be determined by trial and error. For this work it is assumed adequate for demonstration purposes, since the complete system does influence the reward that is obtained. Additional rewards for for example the mass of the system could be implemented to take the complete configuration further into account. Another reward that could be included is a reward for the cost of the system, for which the user can enter a desired system cost. This can be included if the cost for every component is known, which for this work was not possible to determine because the component database is hypothetical and it was deemed impossible to assign hypothetical costs.

A helpful addition to the design tool that can aid in setting priorities on certain aspects of the design can be to implement a weighting system on the rewards. In this way, if more importance should be given to the performance of a certain component type reward, this type can be given a higher reward. This type will then have a larger influence on the system reward, which will give it more importance for the agent.

Modularity of the design tool

From the discussion on the system performance the question can be raised whether the design tool would be able to accommodate an increased number of rewards. This question is extended to a more general discussion on the modularity of the design tool. Modularity of the design tool is thought of as the capacity of extending the different aspect of the design tool, such as the number of components, component types and the analyses. An important consideration in this discussion is that the design tool uses an object-oriented approach, object-oriented programming (OOP).

The main limitation of extending these aspects is due to the RL method that is implemented, the Q-learning method. The method stores a value for every state-action combination in the Q-table, which requires RAM memory to save. In principle, if no limitations on the required RAM size exist any number of components and component types, which determine the amount of state-action combinations, can be implemented. Realistically, this number is limited due too RAM limitations. For this work, the number of components that were used for each component type was limited due to this aspect. For the generation of this work's results, Google Colab is used that allows for the use of 12GB of RAM. The number of components and component types can therefore be extended as long as the RAM limitations are not exceeded.

Extra component types will require few adjustments to the code of the design tool in order to implement the new component type into the analysis of existing component types, such as the battery. In theory, it is therefore possible to implement any new component type into the design tool. If desired, an accompanying component type can be established as well. To establish this, most likely new analyses are required to determine the performance of the component type. The extra analyses will mean the design tool overall will become more computationally expensive, and the time it will take to run the design tool will also increase. An example of such an extension of the design tool was shown in chapter 6, where the DSP component type was implemented in the design tool.

The design tool uses an OOP approach. This makes for a more modular method. This can be illustrated by an example where the user of the design tool wants to implement a different performance analysis for a component. Because the design tool obtains the individual reward by calling the reward class for the component type, the user could swap the component type reward class for their own reward class, eliminating any need to change other parts of the code.

Re-usability of the trained agent

A prospect of using machine learning for generating CubeSat concepts is the fact that the design tool is able to re-use knowledge from previous runs. The design tool uses the Q-learning method that stores the knowledge it is gaining during a training cycle in the Q-table. This table can be saved to be re-used. The results that are shown in section 7.3 are generated using such a trained agent. The design tool that makes use of a trained agent will exploit the design space in stead of exploring it. This is done by removing the randomness from the actions that the agent takes. This guides the agent more directly to the components that provide the highest overall reward. The agent is thereby more likely to select a specific component per component type.

The values that are stored in the Q-table are the Q-values, which are determined by the rewards that are obtained at each state-action combination and are unique for each state-action combination. This means that the values that are stored are mission specific, since the performance is determined for the user entered mission requirements. When a trained agent is used for different mission requirements, the Q-values are no longer completely valid, since the performance of the components will change. How large the change in requirements is will most likely determine the ability of the trained agent to be re-used.

This highlights a main limitation of the design tool with its current reinforcement learning method. The Q-learning method has a limited capability of being re-used for varying mission requirements. It is expected that trained agents can be re-used for a limited change in the mission requirements, but this has not been investigated in this work and should therefore be investigated in future research to quantify this capability.

Limitations of the tool

It is shown that the performance of the design tool in terms of obtained rewards during a training cycle decreases as the complexity increases, in section 7.1. It was shown that this is due to the agent being less likely to select a specific component, leading to lower reward components being more often selected during training. This would suggest an upper limit to the complexity of the design tool for which it is still able to find feasible solutions. The reason for this decrease in performance is recognised to be due to the increase in number of state-action combinations for increasing complexity. In this work, the agent has the same number of iterations for every complexity, which means that as the number of state-action combinations increases it becomes more difficult to traverse the full design space. However, the results for the case study in section 7.3 show that a trained agent does not have this problem and is able to only select the optimal component during a run. This means that the training cycle provides sufficient exploration of the design space to identify the optimal components and paths to these components. For added complexity the design tool might run into problems, which could be solved by changing the agent's settings such as the number of iterations of an episode.

The increasing number of state-action combinations requires more and more RAM memory to store the Q-table of the design tool's agent. Specifications of the system on which the design tool is used means there is an upper limit to the amount of RAM available, which results in an upper limit to the complexity of the design tool. This upper limit determines the number of components and/or the number of component types that can be used in the tool.

An inherent limitation of the Q-learning method is the inability of the method to generalise over unseen data. Q-values are stored for the state-action combinations in the Q-table. However, if the agent finds itself in a state-action combination that has not been encountered before, the method is not able to generalise from surrounding data points to that state-action combination. Because the Q-table is initialised randomly, this results in the agent taking a random action. The sampling requirement for this method is therefore high because most, if not all state-action combinations should be visited in order to generate a fully trained agent. The ability to generalise between mission requirements is also an important aspect for the design tool. This ability can promote the re-use of a trained agent for varying mission requirements.

III

Conclusions & Recommendations

8 Conclusions

The aim of this work was to create a design tool for automated CubeSat concept generation. The design tool would have to be able to take into account the complete design space offered by COTS components, and use these components directly in the design method. The work set out to use a RL method for this goal. The RL method that is implemented in the design tool is the Q-learning method. The prospect of using a machine learning method for the problem at hand is the ability to be able to re-use knowledge from previous runs, accelerating future runs.

The design tool is first experimented with in a low complexity version in order to assess different sorts of implementations of the RL method. After that, the design tool is expanded gradually until the final setup is achieved. The influence of the increase in complexity on the performance of the design tool is documented. The final setup of the design tool consists of the following component types: camera, antenna, ground station, transceiver, attitude determination, attitude control, PCU and battery. Finally, the design tool was aimed to be validated with the AltiCube case study, in order to assess the capability of the design tool to generate a feasible concept. For this validation, the design tool was extended with a DSP component type, and the camera was removed.

The design tool in its current version is able to identify a feasible configuration for the validation case, which is shown in section 7.3. The AltiCube system design and the trained design tool component selection result in a mostly similar system design. Differences are observed in the size and mass of the system and the selected battery by the trained agent. The difference in mass is directly due to the difference in size and therefore does not require further explanation. The difference in size is expected to be due to the way that the design tool analyses the size of the configuration of components. It is observed that the design tool selects components based on a marginal difference in the system rewards of those concepts. The components that are selected by the design tool using a trained agent can be components that are over-performing for the mission requirements. To improve the quality of a trained agent, it is concluded that additional system performance rewards should be implemented that provide a penalty to concepts that are over-performing. Such system performance rewards are currently not implemented, causing the trained agent to select over-performing components. The case study does validate the capability of the design tool to identify feasible concepts for a certain set of mission requirements.

The usability of the design tool in its current version is however limited. The limitations on the usability of the design tool are due to the inherent limitations of the RL method that is implemented, the Q-learning method. The first limitation is because of the way that the method stores knowledge. Storing the knowledge that determines the path of the agent through the design space is done in a Q-table. In this table, a value is stored for every state-action combination. The size of the table that can be stored is limited by the RAM of the system on which the design tool is used. The number of state-action combinations is set by the combination of number of components and component types. The number of components and component types is thereby limited by the system on which the design tool is used. For this work, Google Colab with a RAM size of 12GB was used for the generation of the results. The system managed to run with an average of 3 components per component type, with a total of 7 component types. This resulted in a total of 12.6 million state-action combinations. Within the RAM limit for this work, the number of components and component types cannot be extended significantly over the currently used number. The other limitation of the Q-learning method is the fact that the method is not able to generalise over unseen states. The method saves discrete values for state-action combinations, which are initialised randomly at the beginning of a training cycle. These values determine the action that the agent is going to take. This means that when the agent encounters an unseen state, the action that it takes is based on the randomly initialised Q-values and will therefore be a random action. The agent is not able to use data from surrounding states to predict the most likely optimal action to take. This results in a high sampling requirement for the method because most, if not all state-action combinations should be visited in order to generate a fully trained agent. Although the design tool is able to identify the optimal configuration of components for a set of mission requirements, the method is inherently limited because of the previously specified limitations. For this work, this did not cause issues as the main goal of this work is to provide a proof of concept. However, for future versions of the design tool which might increase the complexity these limitations could prove to be a constricting factor.

The research questions that were stated in section 1.3 are now addressed.

RQ1: What Reinforcement Learning method can be used for automated CubeSat concept generation? The RL method that is used in this work is the Q-learning method. In this work, the design tool that uses this method is proven to be able to automate CubeSat concept generation. The method is able to select components from a component database for specific component types and thereby generate a CubeSat concept. The database contains the component's specifications. Every possible configuration of components makes up the design space of the method. The method's agent is able to travel this design space by taking actions that can change the selected components. These actions are dictated by the agent's Q-table that stores values for each state-action combination. Through a training cycle, the agent explores the design space and updates the Q-values of the Q-table according to the encountered states' rewards. The reward of a state consists of several sub-rewards, which are rewards given for individual component types or component type groups. The configurations of components are analysed for subsystem and system performance based on user entered requirements. By exploring the design space in the training cycle, the design tool is able to assess the performance of the different configurations. Using the trained agent, this gained knowledge can be used to identify the components that will provide the optimal rewards for the user entered requirements. In this process, the CubeSat concept generation is automated by the design tool, where the user only needs to enter the mission requirements, and can assess the output to identify the optimal configuration of components, or CubeSat concept.

RQ2: How can COTS components be implemented directly in the design tool? The components that are used in the design tool are saved in a database. This database contains the specifications of every component that are entered into it. There are specific specifications for every component type that should be specified for every component that is entered into the database. If this is not done correctly, the design tool will not function. By manually entering the specifications, the COTS can be implemented directly in the design tool. At the start of every episode, the agent is initialised randomly in the design space. This means that the agent starts out with a random configuration of components. A list is created for every component type, where the index indicates the component number. For example, if the agent is initialised randomly at index 0, the first component of that component type is selected. A state in the design tool looks simply as for example [1,0,3,4]. The actions that the agent can take is able to change this index for every component type and thereby changes the state and thus the selected COTS components.

RQ3: Is the design tool able to identify feasible CubeSat concepts for a set of mission requirements? The case study has shown that the design tool is able to identify feasible concepts for a set of mission requirements. The case study is performed using the AltiCube system design, which the author of this work assisted in. The results show that the design tool selects components that are generally similar to the system design that is created manually. The case study has also shown that it is relatively easy to modify the design tool by implementing a different component type and adding a component type reward for this new component type. An increase in complexity makes it more difficult for the design tool to select the optimal components during the training cycle. However, for the final setup and the case study, a trained agent is still able to select the optimal components. This indicates that the exploration of the design space that is done during the training cycle is sufficient to enable the agent to identify the optimal configuration of components. For more complex versions of the design tool, the current settings of the agent might prove to be inadequate for providing sufficient exploration of the design tool.

RQ4: Is the design tool capable of re-using data from previous runs? The design tool in its current form has a limited capability of re-using data from previous runs. The Q-learning method that is implemented is, as explained previously, not able to generalise over unseen states. This not only limits the sample efficiency of the method, it also limits the capability of re-using data from previous runs. The re-use of data is about re-using a trained agent for different mission requirements, thereby shortening the time it takes to provide an optimal design for the new mission requirements. The new mission requirements will mean that the states can obtain different rewards. As these states can thereby be classed as unseen states, the Q-learning method is not able to generalise from previous data to provide an estimation for the unseen states. This makes the design tool in its current form limited for re-use. It Is expected that a trained agent can be re-used for small variations in the mission requirements. The trained agent should then most likely be retrained for these new mission requirements, but it is thought that this training cycle can be shorter than the original training cycle. The goal of complete re-usability of the design tool’s trained agent for a large variation in mission requirements can however not be accomplished by this version of the design tool.

9 Recommendations for future work

In chapter 7, a general discussion on the performance of the design tool and the use of the RL method is examined. In chapter 8, the answers to the research questions are stated. The limitations of the tool have been described in these chapters. Future work that will continue on the results of this work should aim to solve and/or quantify these limitations. In this chapter, the author's recommendations are given on aspects that should be addressed in future work that aims to increase the usability and applicability of the design tool that has been created through this work.

Recommendation 1: The components that the design tool can select to form a concept does not constitute an entire satellite as of yet. The usability of the design tool can be increased by implementing aspects so that the generated concepts get closer to a real-life CubeSat design. Aspects that are identified by the author which are currently not implemented are the uplink, propulsion system, on-board computer, various parts of the EPS, and the structure. At the moment, the ability of the design tool to select of the components is strictly defined. A component has to be selected for every component type, and it is envisioned that the design tool should be able to handle user requirements somewhat more flexibly. If the user would for example not want a propulsion system on the satellite, it should be possible to indicate this in the user requirements which would dictate the tool to not select a component for this type. The selection of components is also very defined in terms of numbers in that a single component is selected for every component type. It might however be necessary for a mission to implement both fine and coarse attitude determination components. This situation is already relevant for the case study. A simple solution for this would be to implement multiple component types for such a situation, and split the attitude determination component type into a fine and coarse component type. Implementing such a capability would improve the quality of the concepts that the design tool is able to provide. To handle a larger variety of missions, the payload component type can be implemented as such that it can be specified by the user. In other words, it is recommended to develop the design tool in such a way that it is able to cope with a large variety of mission requirements. This results in a design tool that is able to provide concepts for a large range of missions. These additions are just a part of what could be further implemented in order to increase the quality of the concepts that the design tool can generate.

At the moment, the system performance of the design tool is only represented by the DoD reward. It is recommended that future work is aimed at improving the representation of the system performance into the reward system. Ways in which this can be accomplished is by implementing a reward for the mass of the system and/or the cost. Such rewards could prevent the design tool from over-designing for the requirements.

Recommendation 2: The previous recommendation talked about implementing additional rewards for increasing the representation of the system performance. Next to that, increasing the number of components into the design tool could further increase the number of rewards. All of these rewards are currently summed with equal weight to form the single system reward that is provided to the agent. With the increase in sub-rewards, the effect of a single sub-reward on the system reward decreases with this equal weight system. For future work, it is recommended to implement a weighting system in the reward system that the agent is using. Such a reward system can be seen in Equation 9.1. In this way, the user can specify importance on the performance of a certain sub-reward. If for example the mass is given a higher weight relative to the rest of the components, the design tool will most likely prefer to select components that optimise the mass reward over the other rewards. The exact balance of such weights is up to the user but might significantly influence the results that the design tool generates. Future work should quantify this influence if such a weighted reward system is implemented.

$$r_{system} = \frac{W_1 \cdot r_1 + W_2 \cdot r_2 + \dots + W_n \cdot r_n}{n} \tag{9.1}$$

Recommendation 3: In recommendation 1, it is recommended to implement additional component types in order to increase the usability of the concepts that the design tool generates. However, a limitation of the current method is the amount of RAM the design tool requires in order to save the Q-table that is used. Adding component types will increase the number of state-action combinations and will thereby also increase the required amount of RAM for running the design tool. An increased number of components per component type will also have this impact. If the system that the design tool runs cannot accommodate the increase in required RAM, the tool will become unusable.

The computational expense is an inherent limitation of the Q-learning method. It is therefore recommended that future work should try to implement a different RL that does not have this limitation. An promising option, which was encountered repeatedly in literature as shown in section 2.2, is the use of a Deep Q-Network (DQN), which is a logical extension of the current Q-learning method. DQN implements a neural network to predict the performance of unseen data. The DQN method does not make use of the Q-table but instead trains the neural network on batches of samples for the design space. This means that only the neural network weights need to be saved. The number of components and component types that are used thereby have no influence on the computational expense of the method since these do not influence the number of weights. This would allow a more extensive list of components and component types to be implemented. A downside of training a neural network is the time it takes to train such a network.

Recommendation 4: The other limitation of the Q-learning method is the inability of the method to generalise over unseen states. To improve the usability of the method, the author would recommend future work to implement value function approximation (VFA) in future versions of the design tool. VFA is able to generalise over unseen data by using previously encountered state-action combinations. In this way, the reward for a state is estimated, based on the surrounding data points. DQN also has this capability, since the neural network is trained on batches of samples and the trained network can then be used for the entire design space. It is uncertain whether the current implementation of the component database can still be used when VFA is implemented. This is uncertain because the current implementation presents a discrete design space. It might be required to transform this to a continuous design space for VFA to be successful. In that case, the components indirectly approach of chapter 4 might be more suitable. In this approach, a fit was generated through the components' specifications, thereby creating a continuous design space.

A RL based design tool in an expanded configuration, with VFA, shows promise in improving the feasibility of the design tool. Using a method such as DQN would allow the design tool to use a more extensive list of components. Next to that, it would also allow the generalisation over unseen data, which will improve the sample efficiency of the method. With these functionalities, the design tool is also expected to have increased performance over varying mission requirements, thereby re-using the knowledge gained from previous runs. This in turn could save valuable resources for future project by being able to establish feasible CubeSat configurations more quickly.

Bibliography

[1] What is reinforcement learning? - mathworks. URL https://nl.mathworks.com/help/reinforcement-learning/ug/what-is-reinforcement-learning.html.

[2] Find products and services for your next space mission. URL https://satsearch.co/.

[3] John Abel. Quick-turn, low cost spacecraft development principles, 2016.

[4] Lars Alminde and Karl Kaas Laursen. A strategic approach to developing space capabilities using cubesat technology. In *2009 4th International Conference on Recent Advances in Space Technologies*, pages 43–47. IEEE, 2009.

[5] David A Barnhart, Tatiana Kichkaylo, and Lucy Hoag. Spidr: Integrated systems engineering design-to-simulation software for satellite build. In *Conference on Systems Engineering Research*, 2009.

[6] Audrey Berquand, Iain McDonald, Annalisa Riccardi, and Yashar Moshfeghi. The automatic categorisation of space mission requirements for the design engineering assistant. In *70th International Astronautical Congress*, 2019.

[7] Nesrin Cavus. Aircraft design optimization with artificial intelligence. In *54th AIAA Aerospace Sciences Meeting*, page 2001, 2016.

[8] Wei Chen, Janet Allen, Daniel Schrage, and Farrokh Mistree. Statistical experimentation methods for achieving affordable concurrent systems design. *AIAA journal*, 35(5):893–900, 1997.

[9] Giorgio Cifani. Low cost space mission trends and approaches in early design phases. *International Systems & Concurrent Engineering for Space Applications Conference*, 8 2018.

[10] Hao Cui, Osman Turan, and Philip Sayer. Learning-based ship design optimization approach. *Computer-Aided Design*, 44(3):186–195, 2012.

[11] Peter Dayan and Yael Niv. Reinforcement learning: the good, the bad and the ugly. *Current opinion in neurobiology*, 18(2):185–196, 2008.

[12] Gérald Garcia and Xavier Roser. Enhancing integrated design model–based process and engineering tool environment: Towards an integration of functional analysis, operational analysis and knowledge capitalisation into co-engineering practices. *Concurrent Engineering*, 26(1):43–54, 2018.

[13] Eberhard Gill. AE4S12 Space Systems Engineering Lecture Notes V1.5. Faculty of Aerospace Engineering, Delft University of Technology, 2016.

[14] Daniela Girimonte and Dario Izzo. Artificial intelligence for space applications. In *Intelligent Computing Everywhere*, pages 235–253. Springer, 2007.

[15] David Ha. Reinforcement learning for improving agent design. *Artificial life*, 25(4):352–365, 2019.

[16] Lucy Hoag, Tatiana Kichkaylo, and David Barnhart. A systems architecting approach to automation and optimization of satellite design in spidr. In *International Conference on Engineering and Meta-Engineering (ICEME), Orlando, FA*, 2010.

[17] Yaochu Jin and Bernhard Sendhoff. Pareto-based multiobjective machine learning: An overview and case studies. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(3):397–415, 2008.

[18] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[19] Tatiana Kichkaylo and Gordon Roesler. Automating trade space analysis in systems design. In *Conference on systems engineering research, Los Angeles*. Citeseer, 2011.

[20] Sang-Gook Kim, Sang Min Yoon, Maria Yang, Jungwoo Choi, Haluk Akay, and Edward Burnell. Ai for design: Virtual design assistant. *CIRP Annals*, 68(1):141–144, 2019.

[21] K Kumar, A Vaccarella, N.P. Nagendra, S Gerené, and L Lindblad. Integrated mission design using satsearch. In *SECESA 2018*, 2018.

[22] Wiley J Larson and James Richard Wertz. Space mission analysis and design. Technical report, Torrance, CA (United States); Microcosm, Inc., 1999.

[23] Jean-Luc Le Gal, Guillaume Chaffarod, and Yohan Grégoire. Idm applications: A new paradigm to design parametric models in a collaborative environment. In *SECESA 2018*, 2018.

[24] Ke Li and Jitendra Malik. Learning to optimize neural nets. *arXiv preprint arXiv:1703.00441*, 2017.

[25] Yulin Li, Li Liu, Teng Long, and Weili Dong. Metamodel-based global optimization using fuzzy clustering for design space reduction. *Chinese Journal of Mechanical Engineering*, 26(5):928–939, 2013.

[26] HL Liao, QH Wu, and L Jiang. Multi-objective optimization by reinforcement learning for power system dispatch and voltage stability. In *2010 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT Europe)*, pages 1–8. IEEE, 2010.

[27] Louise Lindblad, Marco Witzmann, and Simon Vanden Bussche. Data-driven systems engineering: Turning mbse into industrial reality. In *SECESA 2018*, 2018.

[28] Pierre Lison. An introduction to machine learning. *Language Technology Group: Edinburgh, UK*, 2015.

[29] Bo Liu, Hadi Aliakbarian, Zhongkun Ma, Guy AE Vandenbosch, Georges Gielen, and Peter Excell. An efficient method for antenna design optimization based on evolutionary computation and machine learning techniques. *IEEE transactions on antennas and propagation*, 62(1):7–18, 2013.

[30] Chunming Liu, Xin Xu, and Dewen Hu. Multiobjective reinforcement learning: A comprehensive overview. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(3):385–398, 2014.

[31] Gérard Maral, Michel Bousquet, and Zhili Sun. *Satellite communications systems: systems, techniques and technology*. John Wiley & Sons, 2020.

[32] F Murdaca, A Berquand, A Riccardi, T Soares, S Gerené, N Brauer, and K Kumar. Artificial intelligence for early design of space missions in support of concurrent engineering sessions. In *8th International Systems & Concurrent Engineering for Space Applications Conference*, 2018.

[33] Daniel Oltrogge and Kyle Leveque. An evaluation of cubesat orbital decay. 2011.

[34] Eva Peral, Shannon Statham, Eastwood Im, Simone Tanelli, Travis Imken, Douglas Price, Jonathan Sauder, Nacer Chahat, and Austin Williams. The radar-in-a-cubesat (raincube) and measurement results. In *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*, pages 6297–6300. IEEE, 2018.

[35] R Venkata Rao, Vimal J Savsani, and DP Vakharia. Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems. *Computer-Aided Design*, 43(3):303–315, 2011.

[36] Xianlin Ren and Yi Chen. How can artificial intelligence help with space missions-a case study: Computational intelligence-assisted design of space tether for payload orbital transfer under uncertainties. *IEEE Access*, 7:161449–161458, 2019.

[37] Steven L Rickman. Introduction to on-orbit thermal environments. 2014.

[38] Manuela Ruiz-Montiel, Javier Boned, Juan Gavilanes, Eduardo Jiménez, Lawrence Mandow, and José-Luis PéRez-De-La-Cruz. Design with shape grammars and reinforcement learning. *Advanced Engineering Informatics*, 27(2):230–245, 2013.

[39] Frederic Runge, Danny Stoll, Stefan Falkner, and Frank Hutter. Learning to design rna. *arXiv preprint arXiv:1812.11951*, 2018.

[40] Silvia Salas Solano, Henri Darnes, Christian Rossiquet, Laurene Gillot, Frederick Viaud, Nelly Rey, Thibery Cussac, Philippe Lariviere, Dominique Delacroix, Laurent Javanaud, et al. Angels smallsat: Demonstrator for new french product line. In *2018 SpaceOps Conference*, page 2731, 2018.

[41] Andrew Santangelo. Opensat (tm), a framework for satellite design automation for responsive space. In *AIAA Infotech@ Aerospace 2007 Conference and Exhibit*, page 2910, 2007.

[42] Charles Schaff, David Yunis, Ayan Chakrabarti, and Matthew R Walter. Jointly learning to construct and control agents using deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9798–9805. IEEE, 2019.

[43] Sara Spangelo, David Kaslow, Chris Delp, Bjorn Cole, Louise Anderson, Elyse Fosse, Brett Sam Gilbert, Leo Hartman, Theodore Kahn, and James Cutler. Applying model based systems engineering (mbse) to a standard cubesat. *IEEE Aerospace Conference Proceedings*, 03 2012. doi:10.1109/AERO.2012.6187339.

[44] Richard S Sutton. Introduction: The challenge of reinforcement learning. In *Reinforcement Learning*, pages 1–3. Springer, 1992.

[45] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[46] C Underwood, G Richardson, and J Savignol. Snap-1: A low cost modular cots-based nanosatellite—design, construction, launch and early operations phase, 15th aiaa. In *USU Conference on Small Satellites*, 2001.

[47] G Gary Wang and Timothy Simpson. Fuzzy clustering based hierarchical metamodeling for design space reduction and optimization. *Engineering Optimization*, 36(3):313–335, 2004.

[48] G Gary Wang, Zuomin Dong, and Peter Aitchison. Adaptive response surface method-a global optimization scheme for approximation-based design problems. *Engineering Optimization*, 33(6): 707–734, 2001.

[49] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[50] James R Wertz, David F Everett, and Jeffery J Puschell. *Space mission engineering: the new SMAD*. Microcosm Press, 2011.

[51] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[52] Pengcheng Ye and Guang Pan. Global optimization method using ensemble of metamodels based on fuzzy clustering for design space reduction. *Engineering with Computers*, 33(3):573–585, 2017.

[53] Kazuo Yonekura and Hitoshi Hattori. Framework for design optimization using deep reinforcement learning. *Structural and Multidisciplinary Optimization*, 60(4):1709–1713, 2019.

[54] Pezhman Zarifian. Team xc: Jpl's collaborative design team for exploring cubesat, nanosat, and smallsat-based mission concepts. 03 2015.

A
Component Specifications

A.1. Final Setup Components

Table A.1: Payload components used in the final setup of the design tool

Name	f [m]	n_{px}	PC [W]	V [V]	Mass [kg]	Size X [m]	Size Y [m]	Size Z [m]
Camera 1	0.0082	2048	0.8	3.3	0.167	0.086	0.0917	0.0551
Camera 2	0.0705	2048	0.8	3.3	0.277	0.086	0.0917	0.0972
Camera 3	0.580	4096	6.0	5.0	1.2	0.098	0.098	0.176
Camera 4	1.0	6600	30.0	5.0	6.0	0.200	0.200	0.250

Table A.2: Ground station components used in the final setup of the design tool

Name	f [MHz]	Gain [dB]	BW [deg]	F [-]
Ground Station 1	146	11.5	53	2.2
Ground Station 2	436	15	35	3.3
Ground Station 3	2290	31.4	5.1	1.0

Table A.3: Antenna components used in the final setup of the design tool

Name	f [MHz]	Gain [dB]	BW [deg]
Antenna 1	146	0	140
Antenna 2	436	0	140
Antenna 3	2290	7.75	70

Table A.4: Transceiver components used in the final setup of the design tool

Name	f [MHz]	RFP [W]	PC [W]	V [V]	Mass [kg]	Size X [m]	Size Y [m]	Size Z [m]
Transceiver 1	146	1.0	3.0	8.0	0.143	0.094	0.094	0.011
Transceiver 2	436	0.501	4.0	8.0	0.075	0.09	0.096	0.015
Transceiver 3	2290	1.995	13.0	8.0	0.217	0.099	0.094	0.025

Table A.5: Attitude determination components used in the final setup of the design tool

Name	Accuracy [arcsec]	PC [W]	V [V]	Mass [kg]	Size X [m]	Size Y [m]	Size Z [m]
AD 1	2	1.0	5.0	0.250	0.09	0.09	0.05
AD 2	2	0.7	5.0	0.350	0.1	0.05	0.05
AD 3	360	0.0375	5.0	0.035	0.0	0.0	0.0
AD 4	1800	0.00875	5.0	0.024	0.0	0.0	0.0
AD 5	3600	0.132	3.3	0.033	0.0	0.0	0.0

Table A.6: Attitude control components used in the final setup of the design tool

Name	Type	Control	T [mNm]	M [mNms]	PC [W]	V [V]	Mass [kg]
AC 1	MTQ	3	Max dipole moment: 0.340		0.427	3.3	0.156
AC 2	Wheel	3	0.23	1.77	1.95	8.0	0.180
AC 3	Wheel	3	1.0	10.82	6.9	8.0	0.450
AC 4	Wheel	3	2.3	30.61	13.5	8.0	0.675

Table A.7: Attitude control components' size used in the final setup of the design tool

Name	Size X [m]	Size Y [m]	Size Z [m]
AC 1	0.0905	0.0969	0.0172
AC 2	0.056	0.056	0.028
AC 3	0.092	0.092	0.046
AC 4	0.099	0.099	0.057

Table A.8: Power control unit components used in the final setup of the design tool

Name	Output Voltage [V]	PC [W]	V [V]	Mass [kg]	Size X [m]	Size Y [m]	Size Z [m]
PCU 1	3.3-5.0	0.160	3.3	0.1	0.0893	0.0929	0.0153
PCU 2	3.3-18.0	0.150	3.3	0.2	0.09589	0.09017	0.01904

Table A.9: Battery components used in the final setup of the design tool

Name	Capacity [Wh]	PC [W]	V [V]	Mass [kg]	Size X [m]	Size Y [m]	Size Z [m]
Battery 1	23	0	-	0.150	0.07	0.0418	0.02068
Battery 2	38.5	0	-	0.258	0.094	0.084	0.023
Battery 3	77	0.004	-	0.5	0.093	0.086	0.041
Battery 4	115	0	-	0.750	0.07	0.0836	0.06204

A.2. AltiCube Components

Table A.10: Ground station components used for the AltiCube case study system design of the design tool

Name	f [MHz]	Gain [dB]	BW [deg]	F [-]
Ground station 1	146	11.5	53	2.2
Ground station 2	436	15.0	35	3.3
Ground station 3	2290	25.0	5.1	0.8
Ground station 4	8000	30.0	5.1	0.8

Table A.11: Antenna components used for the AltiCube case study system design of the design tool

Name	f [MHz]	Gain [dB]	BW [deg]
Antenna 1	146	0	140
Antenna 2	436	0	140
Antenna 3	2290	7.75	70
Antenna 4	8000	13	20
Antenna 5	8000	6	74

Table A.12: Transceiver components used for the AltiCube case study system design of the design tool

Name	f [MHz]	RFP [W]	PC [W]	V [V]	Mass [kg]	Size X [m]	Size Y [m]	Size Z [m]
Transceiver 1	146	1.0	3.0	8.0	0.143	0.094	0.094	0.011
Transceiver 2	436	0.501	4.0	8.0	0.075	0.09	0.096	0.015
Transceiver 3	2290	1.995	13.0	8.0	0.217	0.099	0.094	0.025
Transceiver 4	8000	3.0	28.5	8.0	0.560	0.099	0.099	0.025
Transceiver 5	8000	2.0	12.0	8.0	0.560	0.099	0.099	0.025

Table A.13: Attitude determination components used for the AltiCube case study system design of the design tool

Name	Accuracy [arcsec]	PC [W]	V [V]	Mass [kg]	Size X [m]	Size Y [m]	Size Z [m]
AD 1	6	1.5	5.0	0.350	0.099	0.055	0.05
AD 2	10	0.7	5.0	0.280	0.0905	0.0538	0.0538
AD 3	360	0.0375	5.0	0.035	0.0	0.0	0.0
AD 4	1800	0.00875	5.0	0.024	0.0	0.0	0.0
AD 5	3600	0.132	3.3	0.033	0.0	0.0	0.0

Table A.14: Attitude control components used for the AltiCube case study system design of the design tool

Name	Type	Control	T [mNm]	M [mNms]	PC [W]	V [V]	Mass [kg]
AC 1	MTQ	3	Max dipole moment: 0.340		0.427	3.3	0.156
AC 2	Wheel	3	0.23	1.77	1.95	8.0	0.180
AC 3	Wheel	3	1.0	10.82	6.9	8.0	0.450
AC 4	Wheel	3	2.3	30.61	13.5	8.0	0.675
AC 5	Wheel	3	20.0	60.0	70.2	8.0	0.778

Table A.15: Attitude control components' size used for the AltiCube case study system design of the design tool

Name	Size X [m]	Size Y [m]	Size Z [m]
AC 1	0.0905	0.0969	0.0172
AC 2	0.056	0.056	0.028
AC 3	0.092	0.092	0.046
AC 4	0.099	0.099	0.057
AC 5	0.199	0.199	0.075

Table A.16: Power control unit components used for the AltiCube case study system design of the design tool

Name	Output Voltage [V]	PC [W]	V [V]	Mass [kg]	Size X [m]	Size Y [m]	Size Z [m]
PCU 1	3.3-5.0	0.160	3.3	0.1	0.0893	0.0929	0.0153
PCU 2	3.3-18.0	0.150	3.3	0.2	0.09589	0.09017	0.01904

Table A.17: Battery components used for the AltiCube case study system design of the design tool

Name	Capacity [Wh]	PC [W]	V [V]	Mass [kg]	Size X [m]	Size Y [m]	Size Z [m]
Battery 1	23	0	-	0.150	0.07	0.0418	0.02068
Battery 2	38.5	0	-	0.258	0.094	0.084	0.023
Battery 3	77	0.004	-	0.5	0.093	0.086	0.041
Battery 4	115	0	-	0.750	0.07	0.0836	0.06204

Table A.18: Deployable solar panel components for the AltiCube case study system design of the design tool

Name	Factor [-]
Deployable solar panel 1	0
Deployable solar panel 2	1
Deployable solar panel 3	2
Deployable solar panel 4	3