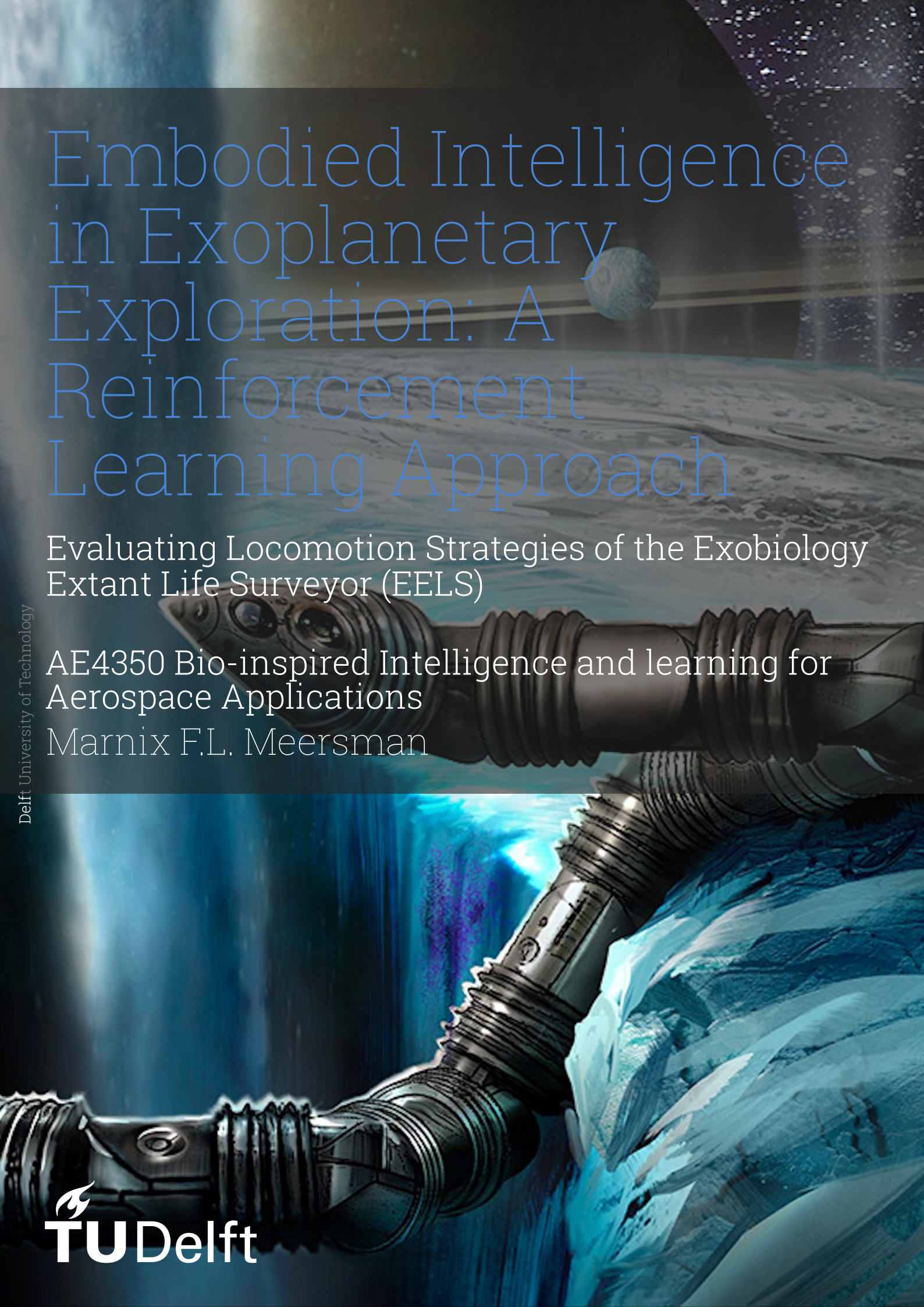


Embodied Intelligence in Exoplanetary Exploration: A Reinforcement Learning Approach

Evaluating Locomotion Strategies of the Exobiology Extant Life Surveyor (EELS)

AE4350 Bio-inspired Intelligence and learning for Aerospace Applications

Marnix F.L. Meersman

Delft University of Technology

TU Delft

Embodied Intelligence in Exoplanetary Exploration: A Reinforcement Learning Approach

Evaluating Locomotion Strategies of the Exobiology Extant Life Surveyor (EELS)

by

Marnix F.L. Meersman

Student Name	Student Number
Marnix Meersman	4650808

Software GitHub Code

Github Repo: https://github.com/MarnixMeersman/EEL_Embodied_Intelligence
Results Tensorboard: https://tensorboard.dev/experiment/oKkN7ge1QjSlHY9vKnE77Q/#scalars
Youtube video: https://youtu.be/LdKwgdQfkKw

Instructor:	prof. Salua Hamaza
Teaching Assistant:	Anton Bredenbeck
Project Duration:	July, 2023 - August, 2023
Faculty:	Faculty of Aerospace Engineering, Delft

Cover: Artistic impression of the EELS robot developed by NASA and JPL hanging over a vent on Enceladus (source: NASA-JPL)

TU Delft

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
BuS	Buffer Size
BS	Batch Size
CAD	Computer-Aided Design
CO2	Carbon Dioxide
EELS	Exobiology Extant Life Surveyor
EI	Embodied Intelligence
LR	Learning Rate
LRS	Learning Rate Schedule
MLAgents	Unity's Machine Learning Agents Library
NASA	National Aeronautics and Space Administration
NASA-JPL	NASA Jet Propulsion Lab
PPO	Proximal Policy Optimization
SAC	Soft Actor-Critic

1. Introduction

Background

Enceladus, one of Saturn's icy moons, has attracted significant scientific attention due to its subsurface ocean and the presence of hydrothermal vents, which are similar to those found at the bottom of Earth's oceans. Studies have indicated that these hydrothermal vents could offer the conditions necessary for life as we know it. For instance, Postberg et. al. [10] have reported complex organic molecules being ejected from these vents, suggesting a conducive environment for the possibility of life.

The Exobiology Extant Life Surveyor (EELS) mission, as proposed by NASA[9], is an ambitious project aimed at probing the liquid ocean beneath Enceladus' icy crust. The centerpiece of this mission is a unique robot, characterized by its multiple linked segments. This design choice facilitates whatis named as 'screw locomotion', enabling the robot to maneuver effectively through varied terrains, especially the challenging terrains expected on Enceladus. At present, the prototype tests for EELS are being conducted on Earth to refine its operational capabilities, as evidenced by NASA Jet Propulsion Lab (NASA-JPL) [7].

While the screw locomotion and multi-segmented design of EELS appear optimal from an engineering perspective, it's vital to validate this assertion within a physically accurate environment. This brings forward the concept of Embodied Intelligence (EI). As defined by Angelo Cangelosi et al. [1], Embodied Intelligence emphasizes the importance of the interaction between a robot's body and its environment in shaping intelligent behavior.

For the scope of this project, we leveraged the Unity game engine, particularly due to its compatibility with the OpenAI Gym API. Unity, being the largest game engine in the world [13], provides a realistic, three-dimensional simulation environment that allows for the recreation of this snake like robot as depicted on the cover image. The MLAgents library [14] offers a framework to implement reinforcement learning strategies effectively. Together, they form a connected platform ideal for evaluating and optimizing the locomotion modes of complex structures like EELS.

Report Outline

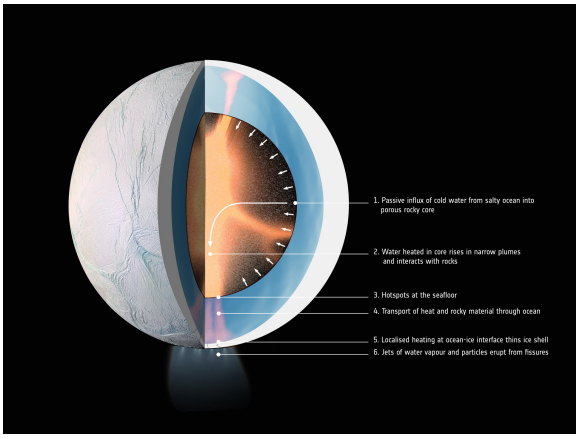
In Chapter 1, a comprehensive literature review is written to explore the existing theories relevant to this subject and to form a foundation for a broader context of the research. Chapter 2 focuses on the methodology followed, with the aim to form a reproducible description of methods, code, variables, software, and parameters used. Chapter 5 presents the results both of the models performance and training procedure. A broader interpretation of the outcomes, including of statistical analysis on multiple runs is discussed as well. Lastly, a sensitivity analysis on the hyperparameters provides insight into the most vital parameters of the learning process. Finally, Chapter 7 concludes the report, summarizing the key findings and outlining potential future research topics to explore.

1. Literature Review

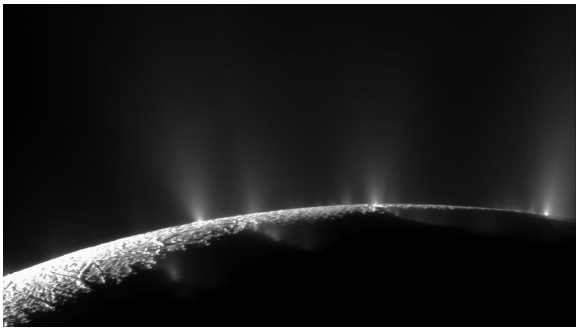
1.1. Enceladus and the probability to harvest lifeforms

Enceladus, the sixth-largest moon of Saturn, has been a point of interest for astronomers and astrobiologists [6]. With a diameter of approximately 500 kilometers, it is covered by a thick layer of water and carbondioxide ice. Underneath this ice shell with an estimated thickness up to 20 km [10], there's evidence of a liquid ocean [15]. The Cassini spacecraft has identified this by evaluating the fluctuations of the gravity field of the moon, which points to the presence of a large water body. However, this liquid ocean couldn't be present due to tidal friction caused by Saturn alone. The scientific consensus is that due to a granulated core instead of a solid core like on Earth, the core can heat up sufficiently to melt the ice layer. This is the cause of a liquid ocean.

The discovery of its icy plumes jetting into space and the potential subsurface ocean has made Enceladus a prime location for the search for extraterrestrial life. These plumes suggest a dynamic environment beneath the icy crust, possibly driven by hydrothermal activity. A schematic overview of the interior is shown in Figure 1.1a



(a) Schematic overview of sectional view of Enceladus [5]



(b) "The icy crust at the south pole of Enceladus exhibits large fissures that allow water from the subsurface ocean to spray into space as geysers, forming a plume of icy particles. NASA's Cassini spacecraft, which captured this imagery in 2009, sampled those particles to reveal the chemicals contained in the ocean." - Greicius T. [8]

Figure 1.1: Enceladus images and findings from Cassini spacecraft.

During its mission around Saturn, the Cassini spacecraft discovered interesting findings. It saw water plumes coming out from the south pole of the moon as shown in Figure 1.1b. Cassini used a mass spectrometer to study the vapor ejected by the plumes and found water, CO2, salts, and other organic molecules [4]. These vapors were collected by performing a fly-bye through one of the ejected plumes. These findings resulted in many mission proposals for a landing mission on Enceladus in search of life. Amongst which the EELS mission proposal which is the topic of this project.

1.2. The EELS Mission proposal

The Enceladus Explorer Life Search mission proposal has been conceptualized as a high-tech effort to discover the possible secrets of Enceladus. This mission is centered around the Exobiology Extant Life Surveyor (EELS) platform, a snake-like robotic uniquely designed for surface and subsurface exploration, especially in icy environments.

1.2.1. Specifications and Locomotion

The EELS platform is approximately 4.9 meters in length and has a mass of around 100 kg [7]. One of its notable features is its segmented design, which grants it versatility in mobility. This design ensures that the robot can navigate the uneven and challenging terrains of Enceladus, be it on the icy/glacial surfaces or subsurface areas filled with soft materials like sand or snow. This unique locomotion mode,

inspired by the serpentine (the motion of snake) motion, allows EELS to reach areas that were previously deemed inaccessible. The left image of Figure 2.2 shows a rendering of the EELS robot with some additional explanations.

1.2.2. Actuation and Movement

Driving this unique movement is the platform's Shape Actuation and Active Skin [7]. These components allow the robot to adjust its form as per the terrain demands, maneuvering smoothly over ice sheets and delving into subsurface crevices. The Active Skin, in particular, provides an additional layer of adaptability, allowing the robot to interact efficiently with its surroundings.

1.2.3. Embodied Intelligence to verify and explore locomotion types

Embodied Intelligence (EI) is a concept that emphasizes the role of the robot's physical interaction with its environment in shaping intelligent behavior [1]. This approach can offer insights into new locomotion types for the EELS robot by reconstructing it through Computer-Aided Design (CAD) and optimizing its movement towards a specific target at a defined speed.

Utilizing the Unity game development environment is ideal for this purpose, as it provides tools to kinematically define the robot and construct a realistic training environment that approximates the terrain of Enceladus. Within Unity, two popular reinforcement learning algorithms, namely Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC), can be employed though the OpenAI Gym API.

Proximal Policy Optimization (PPO)

PPO is a policy gradient method used for optimizing the control of a system [11]. It balances the trade-off between making large policy updates (to learn fast) and avoiding updates that change the policy too much (to maintain stability). PPO optimizes the loss function with a constraint that prevents the new policy from being too different from the old policy. This method is widely used in various robotic applications and has demonstrated robust performance.

Soft Actor-Critic (SAC)

SAC is an off-policy actor-critic algorithm that aims to optimize a stochastic policy in a way that balances reward maximization and entropy maximization [3]. This results in policies that are high-performing and more exploratory, which can be beneficial when training robots to navigate complex environments like Enceladus. SAC's ability to adapt to uncertain environments and learn more efficiently makes it an attractive choice for the EELS mission.

By testing these two algorithms, the proposed approach offers a way to investigate the locomotion types of the EELS robot within a simulated environment.

2. Methodology

2.1. Existing Algorithms

For this project, Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) were selected as the reinforcement learning algorithms for training the EELS robot. These algorithms are chosen for their state-of-the-art performance in robotics applications and their compatibility with the Unity ML Agents library [2, 11, 3]. The choice is also influenced by the project's specific needs for handling high-dimensional and continuous action spaces, which these algorithms are well-suited for.

Other reinforcement learning algorithms, such as Q-Learning, Model-based learning, and imitation learning, were considered but ultimately excluded from the study. The main reasons for their exclusion are the limited integration with the Unity ML Agents library or specific constraints. For example, imitation learning was ruled out because it would require an expert demonstration for training, which would require the development of a controller for humans. This goes against the principles of Embodied Intelligence, which emphasizes the importance of allowing the agent to independently explore and perform in the action space. The comparative performance of PPO and SAC in this context is further discussed in Chapter 4.

2.2. Problem Definition

The primary objective is to train a representative model of the EELS robot for autonomous navigation in different terrains. We do this by leveraging reinforcement learning techniques. The methodological approach comprises of several steps:

1. **CAD Model Construction:** A Computer-Aided Design (CAD) model of the EELS robot was constructed with nine body segments, as shown in Figure Figure 2.2. The choice of nine segments is consistent with NASA's real-life testing and aims to balance fidelity and computational efficiency. 13 segments –like is shown in Figure 2.2 (a)– was tested and trained but excluded due to a significant increase in computational cost.
2. **Unity Integration:** The CAD model was imported into the Unity game engine. Physical links between body segments, as well as their operating boundaries and forces were defined.
3. **Initial Training Environment:** The agent was first trained in a simplified environment (flat ground), inspired by Unity ML Agents examples like the Walker, Crawler and Worm example [1]. Similar to the examples, the agent's task was to move towards a randomly spawned target. This stage served as a basis for more advanced training scenarios. Hyperparameters for the initial training in the Gym wrapper provided by OpenAI were all from the advised defaults provided by Unity ML Agents documentation[2]
4. **Hyperparameter Sensitivity Analysis:** Upon successful initial training, a sensitivity analysis was performed to optimize hyperparameters. The evaluated parameters are:

   - **Beta:** Controls the exploration-exploitation trade-off.
   - **Epsilon:** Ensures numerical stability during training.
   - **Number of Neural Network Layers:** Affects the learning "capacity" of the neural network.

   Every permutation of these three variables was tested as well to identify the optimal set for training in the complex environment.
5. **Complex Training Environment:** After identifying the optimal hyperparameters, the agent was trained in a more complex environment featuring uneven terrain, aiming to mimic conditions on Enceladus.

It should be noted that vision is excluded from the problem, meaning that the agent knows the exact location of the target at the beginning of the simulation. This is fine as we are investigating locomotion types solely.

[1] https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Learning-Environment-Examples.md
[2] https://unity-technologies.github.io/ml-agents/Python-Gym-API/

2.3. Implementation

2.3.1. Agent Definition

This subsection explains in more depth how the agent is designed to provide a realistic yet computationally efficient model for training.

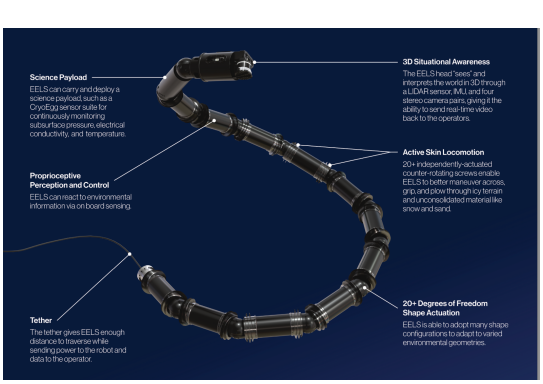
Model Geometry and Simplifications

The robotic agent consists of 9 body segments as is shown in Figure 2.1. All are modeled as capsules which have a low mesh vertex density (green lines surrounding the body). This to minimize the computational effort during the physics simulations.

While the actual EELS robot incorporates a self-propelling screw mechanism (see Figure 2.2, left), this feature is excluded. Initial trials showed that the complexity of the screw mechanism frequently caused the physics engine to break or glitch, rendering the simulation environment unstable, hence untrainable.

In the comparison shown in Figure 2.2, the white ”eyes” serve dual purposes: First, indicating the robot’s forward direction and second, serving as a visual cue to identify body axis rotations (hence 2 ”eyes” instead of 1 in the middle).

Figure 2.1: Side view of the robot with 9 segments



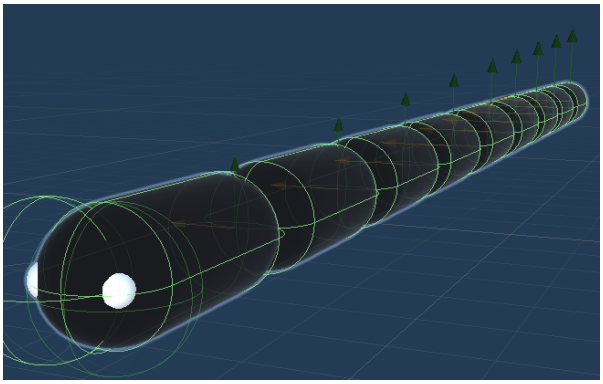


Figure 2.2: Side-by-side comparison of the actual EELS robot (left), as reported in [7], and its simplified simulation model (right). The simplified model excludes the self-propelling screws for simplicity. The white ”eyes” on the model serve two purposes: 1) to indicate the direction the robot is facing, aiding in observing if it rotates around the body axis and 2) to facilitate the reward scheme by confirming target contact through the sphere surrounding the eyes.

Degrees of Freedom

The robot’s joints have configurable degrees of freedom as shown in Figure 2.3. These joints are capable of a sideways range of motion up to $\pm 60^\circ$ around either the x or y axis –respectively the red and green sime-circles in Figure 2.3–, in alternating fashion. This was done to act similarly to the earlier prototypes of the EELS robot, which also has their rotation motors in alternating orientation as shown in Figure 2.4. This configuration provides the robot with a versatile range of motion. For the simulations, the robot’s coordinate system, displayed in the upper right corner of Figure 2.3, remains constant. Note that we used a 9-segment configuration for training to mimic the most complex configuration tested by JPL, while also optimizing computational cost.

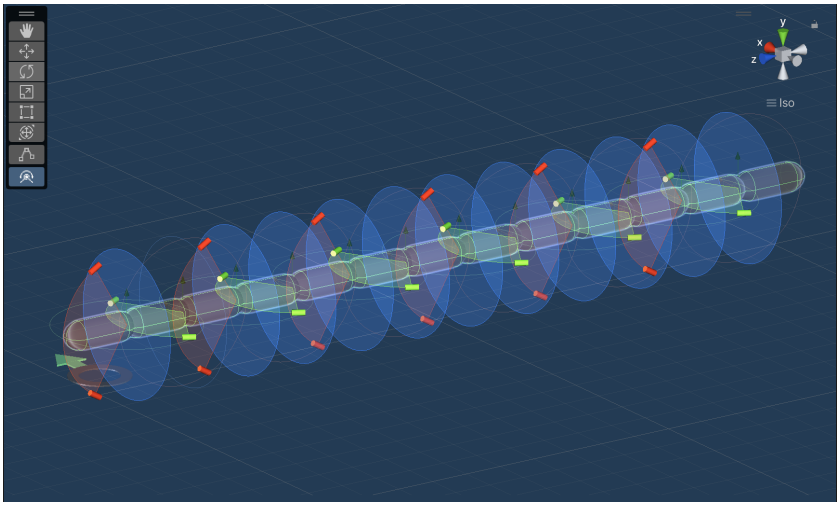


Figure 2.3: Joint configuration of the simulated robot. This picture depicts 13 segments, however 9 were used in simulations.

Figure 2.4: Motor orientation of the early prototypes of the EELS robot.[3]

Observation and Action Spaces

The observation vector for the worm agent has a size of 108, composed of the following elements:

- **Ground Contact:** 9 elements, one for each body segment, indicating ground contact.
- **Velocities:** 54 elements, with 6 elements per segment for linear and angular velocities.
- **Relative Position:** 27 elements, capturing 3D positions of each segment relative to the root segment (which is the average orientation of all body elements).
- **Local Rotation and Joint Strength:** 18 elements, comprising local rotation (9 elements) and normalized joint strength (9 elements).

The observation vector enables the agent to make informed decisions during training.

In terms of actions, the agent operates in a continuous action space with 24 actions per frame in the simulation. Each of the 8 configurable joints (8 joints between 9 segments) can take 3 continuous actions: two for setting the target joint rotation and one for setting the joint strength. This adds up to $8 \times 3 = 24$ actions.

The learning algorithms uses these observations and actions to control the joints' target rotations and strengths, thereby allowing the robot to move in the simulated environment.

[3] Foto taken from MechDesignTV youtube channel: `https://www.youtube.com/watch?v=XwZ6-5rcoPc`

2.3.2. Training Environment

The training environments for the agent was designed to simulate conditions resembling those on Enceladus. Two different types of environments have been used for the agent's training: a simple, flat terrain for initial learning and algorithm comparison, and a more complex, Enceladus-like setting for advanced training. Both the initial and complex training environments accounted for the gravitational and frictional properties of ice/snow on Enceladus, as suggested by Shulson et. al. [12].

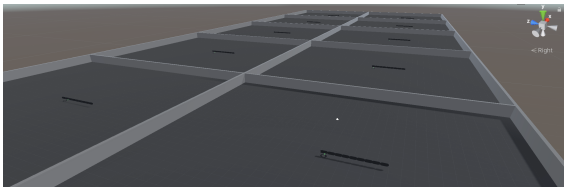


Figure 2.5: Simple training area with flat ground. Training area is multiplied 10 times to speed up training by taking advantage of GPU capabilities.

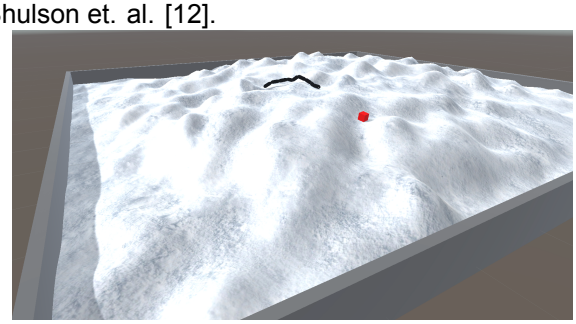


Figure 2.6: A more complex Enceladus-like environment with realitively simple obstacles. Red cube is the target.

Simple (flat) environment for pre-training and algorithm trade-off

The simple, flat training environment serves as a starting point for the agent's initial learning. This less complex setting provides a stable context to train the foundational behaviors and perform algorithm trade-off using a sensitivity analysis, offering a controlled condition for benchmarking the two aforementioned reinforcement learning algorithms. In order to accelerate training, the simple area is duplicated 10 times to speed uptraining as the OpenAI Gym API just takes the mean cumulative rewards as an input. The time-scale was not altered as this can have strange effects on the physics engine.

Complex environment for further training

Upon performing the basics, the agent is transitioned to a more challenging training environment that mimics the surface conditions of Enceladus. This phase aims to teach the agent to navigate through more challenging terrains, such as slopes and crevices, thereby enabling more robust performance when deployed in real-world scenarios.

2.3.3. Reward scheme

The reward scheme for the agent was designed to promote certain behaviors that facilitate reaching a specified target in the environment. The scheme focuses on three main aspects: reaching the target, maintaining orientation towards the target, and sustaining a desired velocity.

1. **Reaching the Target:** The agent receives a significant positive reward when it successfully reaches or touches the designated target.
2. **Velocity Matching:** A continuous reward is given based on how well the agent's actual velocity matches a pre-defined desirable velocity towards the target. The closer the match, the higher the reward. The desired velocity was set to 1 m/s to match the velocity of the EELS robot in real life testing. This velocity was estimated from video's as no conclusive sources could be found.
3. **Orientation Matching:** The agent is also rewarded for aligning its orientation (the head of the snake) with that of a target. The reward increases as the agent better faces the direction of the target, up to a certain angular tolerance.
4. **Composite Reward:** The rewards for velocity and orientation matching are not isolated. They are also combined through unweighted multiplication. This encourages the agent to balance the two objectives of maintaining speed while staying correctly oriented.
5. **Penalties:** A penalty is implicitly given if the agent falls below a certain height from its initial starting position, leading to the termination of the episode and requiring the agent to start over. This was implemented to prevent the training during game "glitches" where the agent somehow penetrates the surface and ends up below the training region. This only happens rarely. There is also a time limit of 5 minutes before the game is restarted. This also implicitly would penalise the agent.

This reward and penalty scheme containing multiple factors aims to guide the agent towards effective locomotion behavior.

2.3.4. Initial Hyper-Parameter choice

Table 2.1 presents the initial hyper-parameter settings, which are the recommended default configurations meaning a good starting point for this problem. Both algorithms use a learning rate of 0.0003, but differ in their learning rate schedule: linear decay for PPO and constant for SAC. This reflects the guidance to allow PPO's learning to stabilize over time, while SAC benefits from a constant rate. The batch size is considerably larger for PPO as more experiences are needed per training iteration. In contrast, SAC operates with a smaller batch but has a much larger buffer size, allowing it to learn from both old and new experiences effectively.

The architecture of the neural networks is configured by the hidden units and number of layers, set to 512 and 3 respectively for both algorithms. Parameters such as gamma and strength are also identical, set at 0.995 and 1.0 respectively, indicate consistent settings for future reward discounting and reward signal scaling. Other similar settings include max steps, time horizon, and summary frequency, which are only important for monitoring training. During the sensitivity analysis chapter, the hyper parameters are further explored.

Parameters	**PPO**	**SAC**
Trainer Type	ppo	sac
Learning Rate	0.0003	0.0003
Learning Rate Schedule	linear	constant
Batch Size	2024	256
Buffer Size	20240	500000
Hidden Units	512	512
Num Layers	3	3
Vis Encode Type	simple	simple
Gamma	0.995	0.995
Strength	1.0	1.0
Max Steps	7000000	5000000
Time Horizon	1000	1000
Summary Frequency	30000	30000

Table 2.1: Comparison of Hyperparameters for PPO and SAC

3. Results

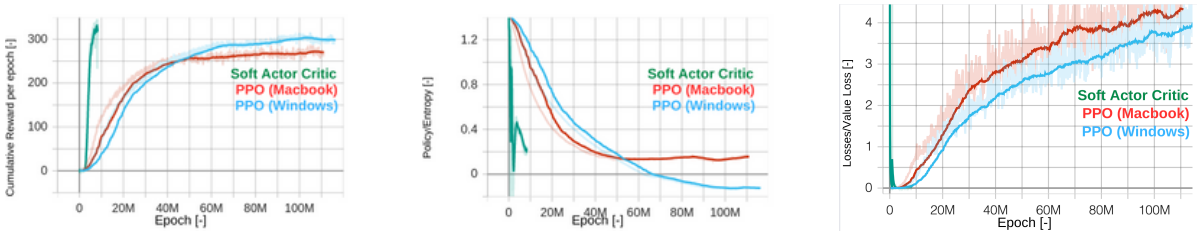
In this chapter we discuss the results after training both in terms of visual performance and in terms of receiving the most rewards.

3.1. Performance analysis by visual inspection

Visual judgement on performance is difficult with only static images. Therefore a short video was created `https://youtu.be/LdKwgdQfkKw` to shown the behaviour of the agent. One can conclude from the movement behavior that the PPO algorithm has a pedidactable behavior. SAC has a less predictable reward scheme and failed to train successfully. Also, PPO trained substantially faster then SAC –8 hours vs. 3 days for 120M epoch–, hence PPO was chosen as the preferred reinforcement learning algorithm.

3.2. Policy Performance in Rewards

The three plots depicted in Figure 3.1 present a comparative evaluation of the PPO and SAC algorithms. These algorithms were executed on two different inference devices: a borrowed NVIDIA GPU on a Windows device and an Apple Macbook. This setup aimed to investigate the model's sensitivity and uncertainty across different computational platforms. Please note that the presented results are specific to the simplified training environment discussed in Section 3.4.4.



(a) Environment/Cumulative Reward **(b)** Policy/Entropy **(c)** Losses/Value Loss

Figure 3.1: Comparison of Environment/Cumulative Reward, Policy/Entropy, and Losses/Value Loss metrics for PPO and SAC algorithms on NVIDIA GPU (Windows) and Apple Macbook.

From the first plot, Environment/Cumulative Reward, it is clear that the Windows device outperforms the Macbook in terms of mean rewards when utilizing the PPO algorithm. Interestingly, SAC (represented by the green line) seems to converge quicker to higher rewards than PPO. However, as shown in the aforementioned Youtube video, the SAC algorithm appears to exploit a loophole in the reward scheme, achieving high rewards without fulfilling the intended objectives.

The second plot, Policy/Entropy, indicates that the model on the Macbook converges to a less random, more stable policy more rapidly than its Windows counterpart, though with lower mean rewards. The SAC algorithm's quicker convergence could be attributed to its off-policy nature, which allows for broader exploration of the action space compared to PPO.

The third plot, Losses/Value Loss, reveals that the Macbook has higher losses in state-value prediction, implying less efficiency [1]. This inefficiency is the reason why further simulations were carried out on the Windows device.

Given the possible loophole exploitation by SAC and the difficulties in adjusting the reward scheme without compromising comparability with PPO, the decision was made to proceed with PPO for the more complex environement.

For those interested in diving deeper into the performance metrics, an interactive and publicly available TensorBoard containing various runs and other relevant data can be found at `https://tensorboard.dev/experiment/oKkN7ge1QjSlHY9vKnE77Q/#scalars`.

[1] `https://unity-technologies.github.io/ml-agents/Using-Tensorboard/`

3.3. Statistical Analysis

To better understand the efficiency and predictability of the Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) algorithms, three histograms are presented in Figure 3.2 that capture their performance metrics. These histograms, displayed from left to right, represent: PPO on Windows, PPO on Mac, and SAC.

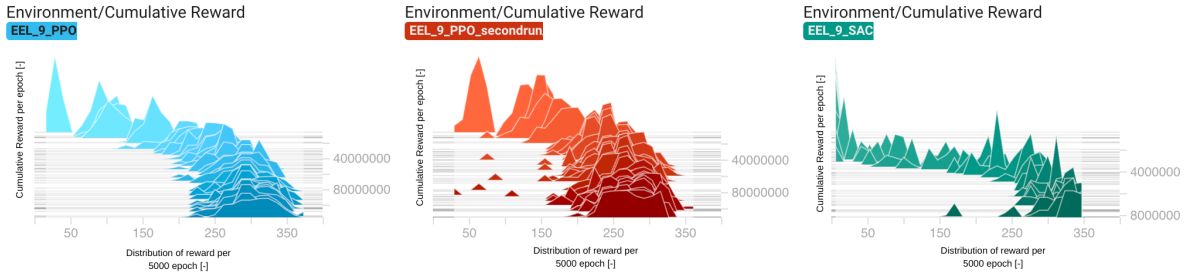


Figure 3.2: Cumulative rewards histogram distributions for every 5000 epoch. Legend showing secondrun meaning run on Macbook computer.

The most striking observation upon examining these histograms is the narrower distribution of SAC when compared to PPO. The SAC performance metrics are not only more concentrated, but they are also marginally higher. This is consistent with the findings from the previous section that highlighted the ”fictional” superior performance of SAC likely due to a loophole in the reward scheme. The narrow distribution suggests that SAC operates with higher consistency, possibly due to its balance of reward and entropy maximization which explore the action space maximally, finally leading to repeating this loophole.

Another interesting observation is the platform-dependent behavior of PPO. The red curve –PPO on Mac–, shows some instances of lower rewards and includes outliers that are notably absent in the PPO on Windows dataset. These outliers may be attributed to system-specific complexities or computational variations. SAC also shows outliers, but less significant.

3.4. Notes on performance in the complex environment

3.4.1. Learning Rate and Stability

The learning rate in the complex environment is significantly slower than in flat-ground tests. Even when initialized with a pre-trained brain from flat-ground simulations, the learning rate remains slow. While attempts were made to speed up the learning process by experimenting with hyperparameters, these changes all resulted in unstable behavior. Although the learning rate is slow, one can still observe stable increasing rewards. as shown in Figure 3.3. One can also see that the rewards are much more wide spread between simulations due to the uneven terrain. It means that, although on average the agent performance improves, there are still many outliers in which the performance is poor. Please note that these remarks only concern the PPO algorithm. SAC was discarded and not tested in the complex environment as discussed earlier.

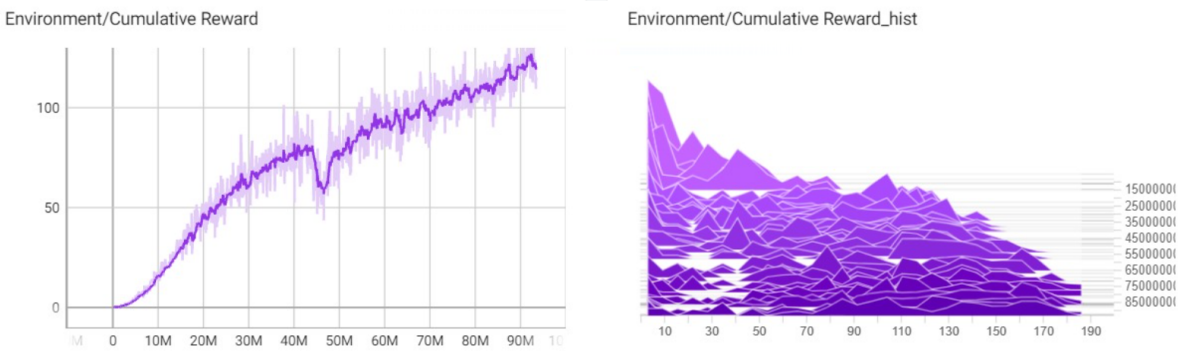


Figure 3.3: Cumulative Rewards for increasing epoch (left) and the histogram of distributions of rewards over per 5000 epoch (right)

3.4.2. Locomotion and Environmental Factors

The performance of the EEL robot in the complex environment, as showcased in the Youtube video, is noticeably worse than in flat-ground tests. While it is expected that the behavior might not be as predictable and consistent, the reasons for these discrepancies could be from other factors as well;

- First, it is essential to note that the self-propelling screw, a fundamental mechanism for this robot as highlighted by NASA, has been excluded from simulations.
- Second, the body elements were modeled as capsules and the ground as a hard object with only the frictional coefficient resembling that of ice on Enceladus. This approximation might not capture the true interaction of the robot with the environment, leading to unnatural bouncing upon impact.
- Lastly, no specific parameters were provided for motor strength, torque, and other mechanical properties. Consequently, reasonable but arbitrary values were assumed.

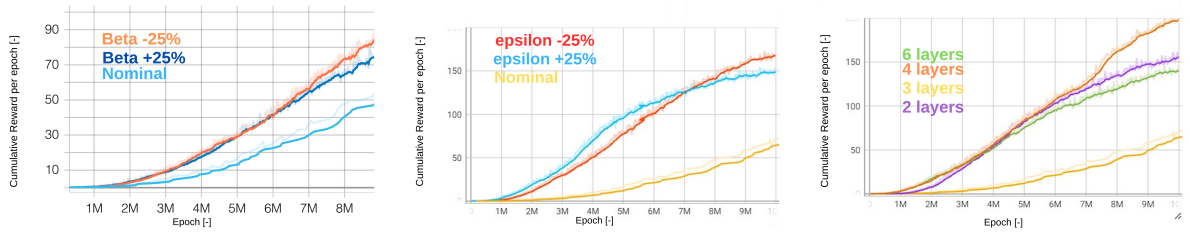
By including these insights, the aim is to provide a more complete view of the challenges and limitations involved in simulating the complex environment for this project.

4. Sensitivity Analysis

This chapter evaluates the influences of certain hyper-parameters on the learning process. It should be noted that due to the high computational cost of the problem, simulations were only performed up to 10 million epochs. Hence, some conclusions that are drawn are from extrapolations of the curves.

4.1. The effect of Beta, Epsilon, and Number of Layers

Three parameters that are explained in the literature review were varied, namely the beta hyper-parameter was varied with ± 25%. The same was done for epsilon. To gain more insight in the effect of number of layers in the CNN, 3 permutations were tested. The cumulative rewards for the first 10 million epoch are shown in Figure 4.1.



(a) Beta **(b)** Epsilon **(c)** Number of Layers

Figure 4.1: The effect of certain hyperparameters (Beta, Epsilon, and Number of Layers) on the learning rate.

Beta

Beta refers to the strength of the entropy regularization, which makes the policy "more random."[1] This ensures that agents properly explore the action space during training. A 25% decrease in the default beta value of 5.0×10^{-3} allows the agent to reach higher rewards earlier as can be seen in Figure 4.1 (a). This is counterintuitive as a lower beta would generally result in a less random policy and potentially less exploration. However, it could be hypothesized that in this specific environment, the agent benefits more from a stable, less random policy. These conclusions are tentative and based on simulations capped at 10 million epochs.

Epsilon

Epsilon influences how rapidly the policy can evolve during training. A 25% increase and decrease from the default value of 0.2 were tested. It was observed in Figure 4.1 (b) that an increased epsilon accelerates the initial learning rate but seems to result in a subsequent decline. Beyond 10 million epochs, the behavior is uncertain due to computational constraints.

Number of Layers

The number of hidden layers in the neural network after the observation input determines the complexity of the model. For simple problems, fewer layers are likely to train faster. In this test, 2, 3, 4, and 6 layers were tried, with 3 being the default. Figure 4.1 (c) shows 4 layers seem to yield better performance starting from 7.5 million epochs. This could be because a more complex model might be better at approximating the intricacies of the environment. However this doesn't explain the poor performance of 6 layers. These observations should be interpreted cautiously as they are based on a 10 million epoch cut-off.

In summary, for all variations in Beta, Epsilon, and the number of layers, faster learning rates than the default configurations were observed. While it's tempting to attribute these differences to the changes in hyperparameters, it's important to remember that the conclusions drawn above are hypothetical and not definitively conclusive due to the 10 million epoch limitation. Additionally, all permutations of the variations of these 3 parameters were tested and can be accessed via the Tensorboard URL included in the previous section. No further discussion is give due to page limitations.

[1] `https://unity-technologies.github.io/ml-agents/Training-Configuration-File/`

5. Conclusion

5.1. Concluding Remarks

For this project, we leveraged the Unity Game Engine and the OpenAI Gym API to train reinforcement learning algorithms on a simulated robot designed to explore Enceladus, one of Saturn's icy moons. A simplified 3D model of the EEL robot was developed and evaluated in two training environments. The first was a simplified flat terrain, while the second more closely resembled the complex surface of Enceladus.

In the initial environment, we compared two reinforcement learning algorithms: Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC). PPO was ultimately chosen due to its superior and more stable performance. This trained model was then deployed into the more complex Enceladus-like environment for further training.

Visual inspection of the agents performance showed that the agent was still actively learning in the complex environment, although at a significantly slower rate. This was unexpected, considering that the model initiated training with a relatively well-performing neural network that had been trained on the simpler environment. Still, performance was deemed rather satisfactory, although questions remain about the realism of the simulated physical interactions between the robot and its environment.

Sensitivity analyses were conducted to explore the impact of various hyperparameters such as β, ϵ, and the number of CNN layers on the learning rate and overall rewards gathered. While some improvements were observed, these findings are inconclusive due to the limited epoch for which all these configurations could be trained with the current computer.

5.2. Recommendations for Future Work

Several options could be explored to improve the robustness the current work:

- **Improve Environmental Physics:** Focusing on the accurate modeling of the physical properties of Enceladus' surface and environmental conditions should be a priority.
- **Enhance Robot Model:** The model could benefit from the inclusion of self-propelling screws, to evaluate whether the robot can independently discover this form of locomotion through Embodied Intelligence.
- **Hyperparameter Optimization:** Further research could explore varying hyperparameters, especially to resolve the ”loophole” issue noted with the SAC algorithm in the reward scheme.

References

[1] Angelo Cangelosi et al. “Embodied intelligence”. In: *Springer handbook of computational intelligence* (2015), pp. 697–714.

[2] Tuomas Haarnoja et al. “Soft actor-critic algorithms and applications”. In: *arXiv preprint arXiv:1812.05905* (2018).

[3] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[4] MM Hedman et al. “Spectral observations of the Enceladus plume with Cassini-VIMS”. In: *The Astrophysical Journal* 693.2 (2009), p. 1749.

[5] NASA/JPL-Caltech/Space Science Institute and LPG-CNRS/U. Nantes/U. Angers. *Surface and Interior Studies of Celestial Bodies*. Graphic composition: ESA. 2022.

[6] Konstantinos Konstantinidis et al. “A lander mission to probe subglacial water on Saturn′ s moon Enceladus for life”. In: *Acta astronautica* 106 (2015), pp. 63–89.

[7] Jet Propulsion Laboratory. *Exobiology Extant Life Surveyor (EELS)*. Accessed: 12/07/2023. 2023. URL: https://www-robotics.jpl.nasa.gov/how-we-do-it/systems/exobiology-extant-life-surveyor-eels/.

[8] Ian J. O’Neill et al. *NASA Cassini Data Reveals Building Block for Life in Enceladus Ocean*. Jet Propulsion Laboratory, Pasadena, Calif.; NASA Headquarters, Washington; Last Updated: Jun 15, 2023; Editor: Tony Greicius. 2023. URL: https://www.nasa.gov/feature/jpl/nasa-cassini-data-reveals-building-block-for-life-in-enceladus-ocean.

[9] Masahiro Ono et al. “Exobiology extant life surveyor (EELS)”. In: *AGU Fall Meeting Abstracts*. Vol. 2019. 2019, P21D–3410.

[10] Frank Postberg et al. “Refractory organic compounds in Enceladus’ ice grains and hydrothermal activity”. In: *AGU Fall Meeting Abstracts*. Vol. 2015. 2015, P11D–03.

[11] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[12] Erland M Schulson and Andrew L Fortt. “Friction of ice on ice”. In: *Journal of Geophysical Research: Solid Earth* 117.B12 (2012).

[13] Unity Technologies. *Unity*. 2023. URL: https://unity.com/industry.

[14] Unity Technologies. *Unity ML-Agents Toolkit Documentation*. 2021. URL: https://github.com/Unity-Technologies/ml-agents.

[15] J Hunter Waite et al. “Cassini finds molecular hydrogen in the Enceladus plume: evidence for hydrothermal processes”. In: *Science* 356.6334 (2017), pp. 155–159.