applied sciences

MDPI

Article

Reward Function and Configuration Parameters in Machine Learning of a Four-Legged Walking Robot

Arkadiusz Kubacki *, Marcin Adamek and Piotr Baran

Institute of Mechanical Technology, Poznan University of Technology, ul. Piotrowo 3, 60-695 Poznan, Poland; marcin.adamek@put.poznan.pl (M.A.); piotr.baran@put.poznan.pl (P.B.)
* Correspondence: arkadiusz.kubacki@put.poznan.pl

Abstract: In contemporary times, the use of walking robots is gaining increasing popularity and is prevalent in various industries. The ability to navigate challenging terrains is one of the advantages that they have over other types of robots, but they also require more intricate control mechanisms. One way to simplify this issue is to take advantage of artificial intelligence through reinforcement learning. The reward function is one of the conditions that governs how learning takes place, determining what actions the agent is willing to take based on the collected data. Another aspect to consider is the predetermined values contained in the configuration file, which describe the course of the training. The correct tuning of them is crucial for achieving satisfactory results in the teaching process. The initial phase of the investigation involved assessing the currently prevalent forms of kinematics for walking robots. Based on this evaluation, the most suitable design was selected. Subsequently, the Unity3D development environment was configured using an ML-Agents toolkit, which supports machine learning. During the experiment, the impacts of the values defined in the configuration file and the form of the reward function on the course of training were examined. Movement algorithms were developed for various modifications for learning to use artificial neural networks.

Keywords: walking robot; quadruped; artificial neural network; reinforcement learning; robots; unity; ML-Agents; ML-Agents toolkit; Crawler; reward function; configuration parameters

Citation: Kubacki, A.; Adamek, M.; Baran, P. Reward Function and Configuration Parameters in Machine Learning of a Four-Legged Walking Robot. *Appl. Sci.* **2023**, *13*, 10298. https://doi.org/10.3390/app131810298

Academic Editors: Charlie Yang and Hongbin Ma

Received: 31 July 2023
Revised: 8 September 2023
Accepted: 10 September 2023
Published: 14 September 2023

1. Introduction

1.1. Motivation

Today, robots are becoming more and more common and can be found in all branches of industry and private applications. Owing to the mechanism of movement, walking robots constitute a special group [1–3]. The current state of technology could suggest that, after so many years since the creation of the first machine of this type, this is a fully developed solution, leaving no room for improvement. However, the real situation differs from the theoretical one. Although walking robots have many advantages compared to other types of robots, such as ease of movement on difficult surfaces and potentially high payloads, they are still structures characterised by the most extensive control algorithm [4,5]. In the era of the significant development of artificial intelligence applications in many areas of human life, it seems reasonable to also use it to control walking robots. This solution will potentially simplify the complex control algorithm [6,7].

1.2. Literature

Walking robots can be divided based on many criteria, and some of them are closely related. The basic criterion is the number of legs. The minimum number of limbs can be zero; in this case, we are talking about crawling robots for which movement resembles the crawling of a snake. Robots with one leg (monopeds) move by jumping. This type of gait is very difficult to implement owing to the need to use drives with high power and dynamics, as well as a fast control system. The most used walking robots are two-legged

(bipeds), four-legged (quadrupeds), and six-legged (hexapeds). These three types of robots will be discussed in more detail in the following sections. In addition, there is also a group of multilegged robots, that is, with more than six legs [8–10].

For this study, a three-degree-of-freedom quadrupedal robot was developed in each leg, for which the biological reference is a reptile. This decision was conditioned by the fact that the biological pattern of the reptile is characterised by good stability and dynamics [11,12].

The concept of artificial neural networks refers to the concept of mathematical structures along with their software and models. With their help or through rows of elements, called artificial neurones, calculations are carried out on many given inputs. Originally, the inspiration for the creation of the described structures was the construction of natural neurones, the synapses that connect them, and the entire nervous system, consisting of these elements [13].

The basic features of artificial neural networks include:

- The ability to abstract, i.e., generalise, the content acquired in the training process;
- Relative fault tolerance refers to a situation where one of the network's neurones may not work properly, and yet the result obtained at the output will be close to correct;
- The ability to quickly "learn", by which we mean the ability to respond appropriately to specific stimuli.

The use of artificial neural networks is possible, for example, by using software that supports such an application. One of the programmes used for this purpose is Matlab Simulink, an interactive environment equipped with its own programming language. It allows you to create fully functional programmes and is equipped with libraries dedicated to building neural networks (Deep Learning Toolbox and Fuzzy Logic Toolbox). Another programme in this category might be Statistica®Neural Networks, one of the most technologically advanced tools that combines high efficiency, by implementing many rare functions, with ease of use, even for inexperienced people [14]. In addition, the creation of neural networks is possible using a set of tools that includes:

- Unity Engine Editor for visualising and creating a learning environment;
- A machine-learning-agent tool, supporting the learning process on the engine side;
- PyTorch library to perform the process, its configuration, and save the obtained results.

In the artificial neurone in Figure 1, there are input vectors that are multiplied by the weight values and then subjected to a given internal processing function. In the next stages, the result of the above task is subjected to the activation function on the basis of which the output value of the neurone is determined. The described process proceeds according to the following mathematical description:

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i\right) = \varphi(W \cdot X) \quad (1)$$

where X is the input data vector, W is the weight vector, φ is the activation function, and y is the output signal.

The described artificial neurone is a single structural element from which entire artificial neural networks are made. These networks are characterised by a non-linear character, because of which they are capable of solving complex problems, difficult from the point of view of linear modelling. The most popular ANN construction scheme is a layered structure, which consists of an input layer, an output layer, and hidden layers in between. Such a definition of its operation ensures significant efficiency in controlling multidimensional processes.

Despite the generally defined scheme of the artificial neural network, we can distinguish several of its types. The first, and most basic, of these is the one-way network. There is no feedback; that is, each piece of information passes through each neurone of the network only once. In addition, they can be differentiated into single-, double-, and

multi-layer networks. Other types are recursive networks (e.g., Hopfield networks and Boltzmann machines). We can call such a network a creation in which the connections between neurones constitute a graph with cycles. Another type of ANN worth mentioning is self-organising maps, called Kohonen networks. In their case, the corresponding coordinates placed in any n-dimensional space are juxtaposed with the neurones. In addition to the networks mentioned above, there are other networks, such as support vector machines, networks based on radial basis functions, token-transmitting networks, and networks based on impulsive neurones [15].

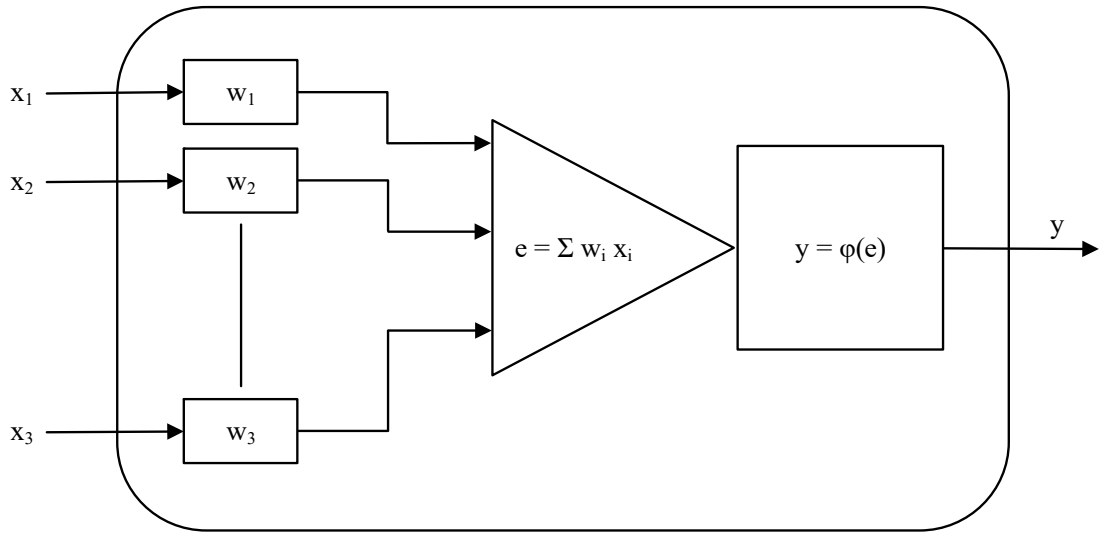

Figure 1. Model of the described neurone.

Owing to their versatility and an ever-wider spectrum of possibilities, artificial neural networks have found many applications in an increasing number of fields including:

- recognition, denoising, and classification of graphics (including letters);
- speech synthesis;
- in the aviation industry, to identify dangerous items in luggage;
- data analysis, especially finding correlations between data series;
- estimation, prediction, and approximation of the output based on inputs, without detailed knowledge of the process [16,17].

The current state of technology, in the context of teaching methods using artificial neural networks, distinguishes three basic trends:

- supervised learning;
- unsupervised learning;
- reinforcement learning.

In the study that is the subject of this article, the third type was used. Reinforcement learning is a type of dynamic programming for which the task is to train algorithms through a system of punishments and rewards. This means that the agent undergoing the learning process is tasked with minimising the punishments received and maximising the rewards based on entering appropriate interactions with the environment. The main difference, compared to the other two mentioned methods, is the lack of a prepared set of training data, but only in the configured environment [18].

Reinforcement learning can be represented by a mathematical model of the Markov Decision Process (MDP).

$$\text{MDP} =< X, A, \varrho, \delta > \tag{2}$$

where X is a finite set of states, A is a finite set of actions, ϱ is a reward function, and δ is a state transition function.

To fully understand the essence of reinforcement learning, it is necessary to describe the three most important components of the process. The first is the environment, that is, the simulation with which the agent interacts [19,20]. It is characterised by the following features:

- state—according to the name, it is a variable that defines the current state of the environment;
- step—a function which, based on the performed action, updates the state of the environment and returns a reward and an observation;
- episodes—a set of steps that, upon being reached, resets the state of the environment;
- reward—a variable returned by the environment, specifying the benefit (or loss) obtained after a given step;
- observation—a scalar, vector, or matrix returned by a step for which the task is to describe the state of the environment at a given moment.

The second part of the reinforcement learning process is the agent, that is, the element that interacts with the environment. As described above, its task is to determine the most favourable way to interact with the environment. The basic parameter of the agent is the function responsible for its behaviour, accepting the observation, and returning the action [21].

Last but not least, the most important part of the training process is the buffer. It acts as a container for the data collected by the agent during training. Then, they are used to continue training [22].

The learning process itself begins with the initial collection of data through the implementation of the random policy. This stage consists of carrying out a set number of episodes and then saving the data obtained in this way in the buffer. The rest of the training follows the loop [23]:

- data collection—each iteration begins with repeated data collection; however, unlike the initial stage, it is done using the agent's policy;
- learning—the calculation of the loss, or gain, of points by the agent based on the collected data.

In addition, two different types of learning should be mentioned [24,25]. The first of these can be described as negative learning. This means that training is effective on the basis of avoiding or discontinuing the punishing behaviour. The second possible approach is positive learning. In this case, the basis of training is to increase the frequency and intensity of the agent's rewarding behaviour.

On the basis of the set of reinforcement learning criteria described above, it is possible to distinguish the following advantages:

- It can be used for very complex problems that would be impossible or very difficult to solve using conventional methods;
- The process is somewhat immune to errors that arise during it because, by providing it with the possibility of carrying out the appropriate number of episodes, they will be automatically corrected.

Unfortunately, this method is not flawless. This includes the fact that, to obtain satisfactory results, a long time must be devoted to data collection, and a multitude of calculations is required to be carried out during training. Furthermore, reinforcement learning should not be used to solve simple problems, as this would involve much more work than would be necessary compared to conventional methods.

In article [26], various reinforcement learning algorithms were reviewed from the perspective of overcoming the track and avoiding obstacles caused by cars. The best effectiveness, characterised by a higher reward function, was achieved using the PPO training method after prior training with the behavioural cloning method. The authors in [27] created a computer game that they then transformed into a simulation environment to teach intelligent agents. They then used hyperparameter tuning to achieve the best possible performance of the agent in the final commercial production. In [28], deep Q learning algorithms and their implementation methods in the Python environment with the use of

the PyTorch library were analysed to solve the problem of high complexity reinforcement learning. The learning itself was carried out in a real environment, and this approach allowed the achievement of results above the assumed minimum. In article [29], the authors made a comparison of the hyperparameters epsilon, lambda, beta, and num_epoch in the context of the agent traversing the maze. Their settings depend, in this case, on the complexity of the maze itself and the complexity of the actions taken by the agent. Previous work [30] focussed on the use of automated hyperparameter optimisation, which is supposed to show greater efficiency compared to experts in a given field. Again, this issue is analysed in the context of model-based reinforcement learning. The authors in [31] present the use of a neural network in planning the route of a walking robot. The inputs of the neural network were filtered accelerometer indications, magnetometers, and camera images, based on which the robot recognised the given trajectories.

1.3. Contributions and Innovations

This paper presents an approach that is significantly different from that described above. First, the robot model that is used not only is a virtual object but also has a physical representation because of which it is possible to relate the obtained learning results to real situations. The second distinguishing feature is that hyperparameter tuning is preceded by studying the impact of the reward function on the quality of learning. Furthermore, instead of using a ready-made scene provided with the ML-Agents package, it was decided to create an environment that allowed the assumed learning process to be carried out.

1.4. Main Difficulties

The purpose of the first study was to check how different configurations of the reward function affect the course of the learning process by strengthening the robot model. Then, based on the optimal configuration, further modifications were made, this time, to the .yaml configuration file.

To implement this problem, a CAD model of the robot was developed, transferred to the Unity simulation environment, and then the main scene was configured, which is the place where the robot is trained. The last element of the learning environment is the configuration of the learning process itself. For this purpose, the configuration file responsible for the basic parameters of the learning process is adjusted accordingly. The whole is completed by creating a programme that determines the interaction of the robot with the environment, and the resulting conclusions are taken into account when creating a neural network.

1.5. Structure of the Paper

The first part of the work contains a preliminary description of the research that was carried out and the definition of the reward function that was the subject of one of the experiments. The following section describes the process of designing the robot model, its export to the learning environment, and the configuration of the environment itself. Then, the machine learning module that was used is characterised, along with a detailed description of the parameters defining it. In addition, the library used to visualise the learning results is presented. The next stage was the characterisation of the input data and description of the entire study.

The next section presents the results of the experiments divided into two stages. In the first, the form of the reward function was modified, while in the second, the hyperparameters of learning were changed. In a further stage, the results were analysed by taking into account the possible causes of the observed tendencies and their potential effects during the implementation of the learnt gait. The last two chapters focus on a comprehensive summary of the research that was conducted, and recommendations for further activities in this field are proposed.

2. Materials and Methods

2.1. Main Work

As a part of this work, two studies were carried out, each of which consisted of three different cycles. The first task was to find the optimal reward function in the context of the learning quality. The reward function used in this study is calculated using the following formula:

$$Rf = R_{vel\_look} + R_{encg} + R_{body\_pos} + R_{trig} \tag{3}$$

where Rf is the reward function, R_{vel_look} is a reward related to the speed and orientation of the robot, R_{encg} is an award given to encourage the agent to take action, R_{body_pos} is a reward related to the location of the corpse, and R_{trig} is the reward given only when the goal is achieved.

The above component functions of the reward are characterised by the following formulas and assume the following values:

$$R_{vel\_look} = \left[\left(1 - \frac{Actual\ Speed^2}{Target\ Speed}\right)^2\right] \cdot \left[\left(\left(\vec{A} \cdot \vec{B}\right) + 1\right) * 0.5\right] \tag{4}$$

where the first term is responsible for checking the speed, and the second term is responsible for the orientation as follows:

$$R_{encg} = -\frac{0.5}{MaxStep} \tag{5}$$

where MaxStep is the maximum number of steps in an episode:

$$R_{body\_pos} = -\frac{0.5}{MaxStep} \tag{6}$$

where MaxStep is the maximum number of steps in an episode:

$$R_{trig} = +500 \tag{7}$$

Given all the values, the following formula was obtained:

$$Rf = \left[\left(1 - \frac{Actual\ Speed^2}{Target\ Speed}\right)^2\right] \cdot \left[\left(\left(\vec{A} \cdot \vec{B}\right) + 1\right) * 0.5\right] - \frac{0.5}{MaxStep} - \frac{0.5}{MaxStep} + 500 \tag{8}$$

where the last value (+500) occurs only in the case of an episode in which the agent achieved the goal.

The second task, based on the previously defined reward function in Equation (8), was to examine the impact of various hyperparameters in the configuration file on the course of the training. It was found that the best results were achieved when the reward function highlighted the correct attitude of the robot during the process over the mere achievement of the goal. In terms of hyperparameters, the best results were obtained by reducing the number of hidden layers.

2.2. Robot Design, Export to the Environment, and Its Configuration

The robot, being the object of the study, was designed with the use of Dassault Systèmes Solidworks 2021 software. To estimate its weight, PLA plastic was assumed as its material. The model has four limbs, each with three joints, which translates into three degrees of freedom in each of them.

Using the Rhino 7 programme, the robot was transferred from the design software to the learning environment. The view of the converted robot is shown in Figure 2. Additionally, it was necessary to establish the appropriate hierarchy of connections between the components of the limbs. To enable learning, it was also necessary to create a scene in which it would take place.

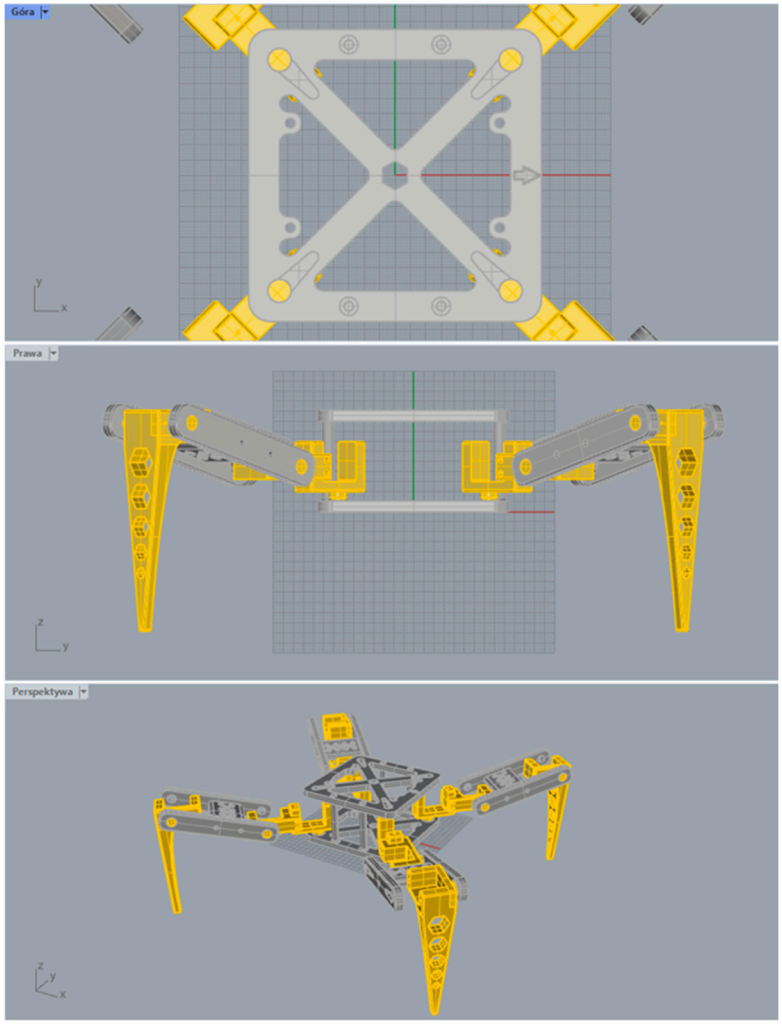

Figure 2. Robot model conversion using Rhino 7.

2.3. ML-Agents Module

After the appropriate mechanical adjustment of the robot model, it was possible to configure the scripts necessary to teach the robot. This process was based on an example from the ML-Agents documentation; on this basis, it was decided which scripts would be necessary for the correct conduct of the teaching process. These included:

- Decision Requester—This is a programme that is responsible for making decisions by the taught object after collecting observations. Here, you can adjust how many steps the decision will take. In addition, it can be used to force the model to make decisions between steps, not only after a certain number of them;
- Model Override—This is a script that is responsible for handling the teaching process, i.e., overwriting the model, and correct completion of the process;
- Rigid Body Sensor Component—This is a programme that is responsible for connecting all the rigid body components into one coherent whole;
- Behaviour Parameters—These are one of the main scripts responsible for teaching. It is from this level that numerous observations (the global position of the robot and position of each joint, number of actions, and their type) are made; and, here, it is possible to upload the learnt neural network. The actions that the trained model performs can be discrete (either 0 or 1) or continuous (from 0 to 1);
- Joint Drive Controller—from this programme, you can adjust the values of the forces in the joints. Three parameters can be adjusted: the maximum spring force, damping force, and maximum joint force;

- Crawler Agent—This is a script in which the learning process is defined, including what the learning model will be punished for, what it will be rewarded for, what its goal is, and how many steps it takes to achieve this goal. In addition, in this programme, you define all the actions that the model can perform and what input signals will reach it. This is also where you call all the scene-setup functions that need to be performed at each end and beginning of an episode.

One of the most important aspects of the simulation was the correct mapping of physics. For this purpose, two additional programmes that were used affect the basic physical values implemented in the Unity environment. The first script is "Centre of Mass". It is responsible for actively shifting the robot's centre of mass depending on the position of the limbs. Another script is "Custom Gravity". This is a programme that makes it easy to adjust the gravity force in such a way that the robot's fall speed is as realistic as possible.

The input data for the learning process include, among others, a learning process configuration file, which defines, for example, the training method (trainer type), hyperparameters, and some other additional values used during the training process. Depending on the choice of the first value, i.e., the choice of the trainer type, different values will be available. The trainers to choose from are as follows:

- PPO (Proximal Policy Optimisation)—This uses a neural network to approximate an ideal function that matches the agent's observations to the best possible action that the agent can take in a given situation;
- SAC (Soft Actor Critic)—This is an algorithm that optimises the random behaviour of the agent, guided by the criterion of achieving the highest possible entropy of the process;
- MA-POCA (Multi-Agent Posthumous Credit Assignment)—This is a trainer that is designed to teach multi-agent environments. Its task is to train a neural network that will act as a mentor for the trained agent.

If you have not decided to define your own .yaml file, the Unity environment will define this file itself with basic values. The block diagram according to which the learning process is carried out using the ML-Agents package is presented in Figure 3.

In this study, the type of PPO trainer was chosen, which was the default choice, and was the factor that defined which values could be adjusted in the configuration file. The most important values are as follows:

- hyperparameters batch_size—This is the number of data in each iteration. This value should always be much smaller than the buffer size. If continuous actions are used, it takes a value of 1000 s. For discrete actions, it takes a value of 10 s;
- hyperparameters buffer_size—This is the number of data that needs to be collected before updating the model. The buffer size should be much larger than batch_size. Typically, a larger buffer results in more stable model updates during the training process (default = 10,240);
- hyperparameters learning_rate—This is the initial learning rate for the simple gradient method. This value corresponds to the impact of each update step. If the training is unstable or the reward value does not increase consistently, it is probably set too high (default = $3 * 10^{-4}$);
- hyperparameters beta—This is the effect of the regularisation of entropy, which makes the agent's policy more random. Owing to this parameter, agents can properly explore the action space during training. The higher the value, the more random the agent's actions. This parameter should be adjusted so that the entropy value slowly decreases as the reward increases (default = $5 * 10^{-3}$);
- hyperparameters epsilon—This affects how quickly the agent's policy evolves during the learning process. A small value produces more stable updates between episodes but slows the learning process (default = 0.2);
- hyperparameters lambda—This is the learning regularisation parameter. It provides a trade-off between the set and actual reward values (default = 0.95);

- hyperparameters num_epoch—This is the number of epochs that pass through the buffer during the optimisation (default = 3);
- hyperparameters learning_rate_schedule—This determines how the rate of learning changes over time. For PPO, it is recommended to reduce the learning rate to max_steps to make the process more stable (default = linear);
- Network_settings normalises—This is a function that is applied to the vector input. It is useful for complex continuous control problems, but can be harmful for simpler discrete control problems (default = disabled);
- Network_settings hidden_units—This is the number of units in the hidden layers of the neural network. For simple combinations of input data (observations), it should take a small value (default = 512);
- Network_settings num_layers—This is the number of layers hidden in the neural network. For simple problems, fewer layers can have a faster learning rate and better performance. More layers may be useful for more complex problems (default = 3);
- Network_settings vis_encode_type—encoder type for encoding visual observations (default = simple)Extrinsic strength—This is the coefficient by which the award granted by the environment should be multiplied. Its value depends heavily on the reward signal (default = 1.0);
- Extrinsic gamma—This is the coefficient used to calculate successive rewards from the environment. It determines how much the agent should consider possible rewards received in the future;
- keep_checkpoints—This is the maximum number of model checkpoints to keep. The checkpoints are written after the number of steps is specified in the checkpoint_interval variable (default = 500,000). When the maximum number of records is reached, the oldest checkpoint is deleted when saving a new point (default = 5);
- max_steps—This is the total number of steps that the agent must take in the environment before the learning process ends. If multiple agents are trained simultaneously, all the steps they take will affect this number (default = 5,000,000);
- time_horizon—This defines how many steps the agent must take before adding data to the buffer. When this limit is not reached, the estimated value is used to predict the expected value. This number should be large enough to capture all the important agent behaviours in a given episode (default = 64);
- summary_freq—This is the number of data that needs to be collected before training statistics are generated and displayed. This is a very important parameter when using graphs generated by the TensorBoard library.

This description of hyperparameters contains guidelines contained in the module documentation. They tell what features of the environment should be guided by the selection of numerical values for each parameter. This allows for an initial selection that will allow for the best possible results of the initial learning cycles. This approach is known as grid searching. Subsequently, it is possible to automate the tuning of the hyperparameters. This consists of establishing criteria in terms of which successive iterations of learning will be validated. The selection of numerical values can be carried out using various strategies:

- random searches;
- Bayesian Optimisation;
- gradient-based optimisation;
- Evolutionary optimisation;
- population-based optimisation;
- radial basis functions;
- spectral methods.

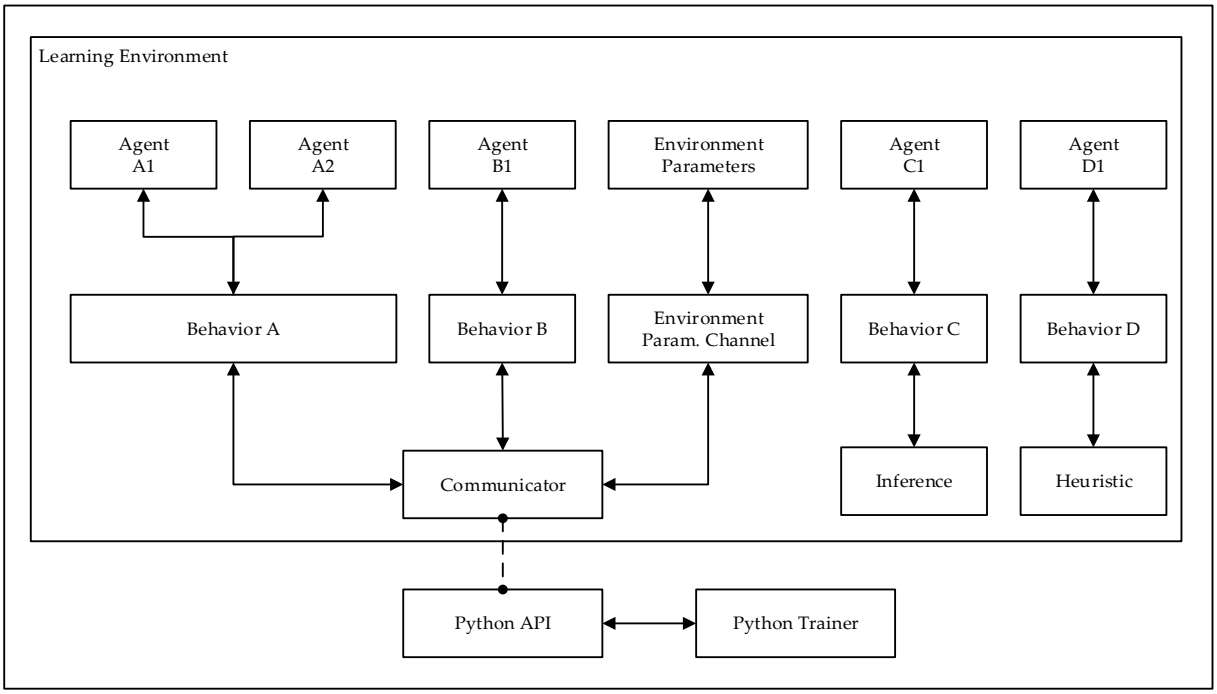

Figure 3. Block diagram of the ML-Agents package.

2.4. Visualisation of Research Results

The TensorBoard library is one of the tools of the TensorFlow library for data visualisation. It allows you to present the results of calculations of neural networks in a simple and legible way. Thanks to the ability to view many variables related to machine learning, it allows us to easily understand the progress of training and improve the model performance by updating hyperparameters. One of the many advantages is the ability to view the results for several neural networks on one graph.

2.5. Preparation of Study Input Data

This study will focus, among other things, on analysing the impacts of process parameters and the learning algorithm on the speed and quality of the training. For this purpose, it is necessary to define conditions that will be constant throughout all the trials. The first is the number of agents that will be taught in parallel, as shown in Figure 4. It has been selected so as not to overload the PC on which the calculations are carried out, while maintaining the highest possible learning speed. Their number was set at 25. The next constant attributes throughout the study will be the values declared in the Decision Requester script. They have already been described above; only the set values are presented here:

- The decision period, that is, the number of steps after which the agent makes decisions about the action, takes the default value of 5;
- Take Actions between Decisions, i.e., a logical variable determining whether the agent can take actions between decisions, takes the default value of inactive.

The last script that requires explanation is the Crawler Agent. The two most important values defined in its scope are as follows:

- Max Step—the maximum number of steps within one episode after which it is reset without reaching the goal;
- Target Walking Speed—the speed the robot is trying to achieve, for which it is also rewarded. It is set to the maximum value: 15.

The penultimately described script is the Joint Drive Controller. In the research that was carried out, it is characterised by the following values:

- Maximum spring of the joint—spring force of the joint: 40,000;

- Joint Dampen—Joint Damping force: 5000;
- Maximum joint force limit—maximum joint force: 20,000.

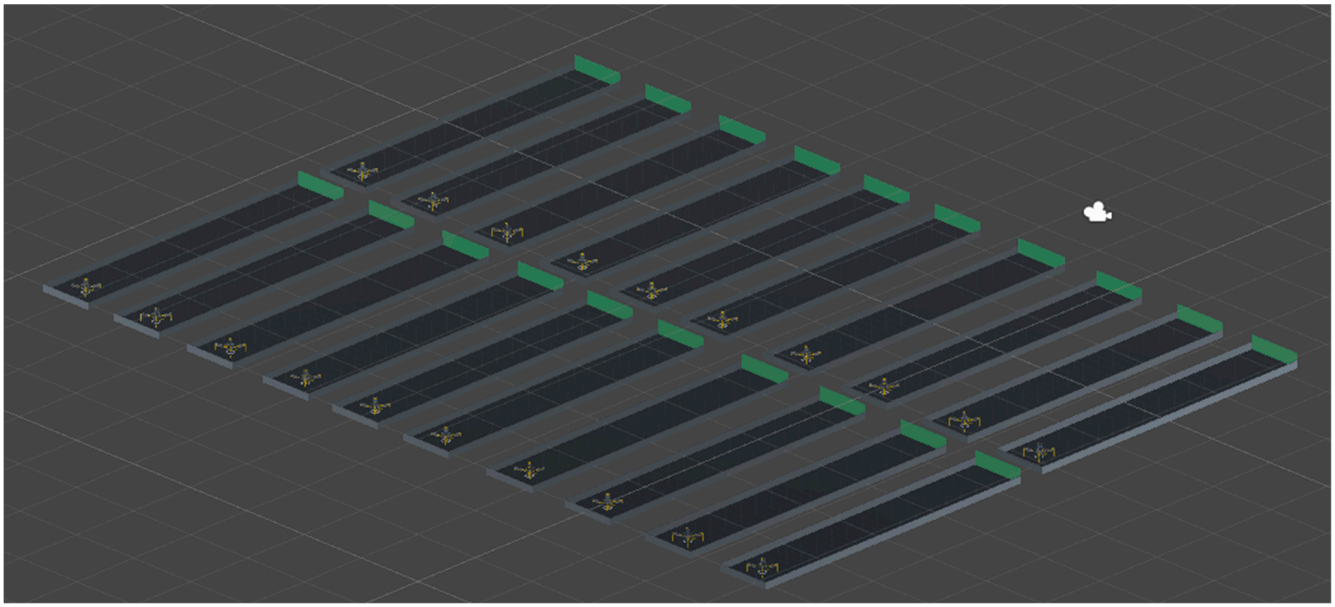

Figure 4. The learning environment.

2.6. The Course of the Study

At the outset, it should be noted that the maximum number of steps in the learning process has been set at 5,000,000 steps. The comparison will be within the given range because it was found that after this time, the trend of the process stabilises.

As intended, the first study focused on modifications of the reward function in the context of its impact on the learning quality and speed. For this purpose, three learning cycles were performed, each with a different form of the reward function, as follows:

- The first cycle included a balanced distribution of the reward awarded to the agent both for achieving the goal and for behaving in accordance with the assumptions during the process, as represented in green on the charts;
- The second focused on a significant increase in the value of the reward awarded for achieving the goal alone, while leaving the rewards awarded for the correct behaviour during learning, as marked in orange below;
- The third cycle had the same value for achieving the goal as the first award, and the value for behaviour during the process was doubled, as characterised by the navy-blue colour.

The second study used the best of the above variants in terms of the learning efficiency; and, on this basis, the impact of three key hyperparameters defined in the configuration file of the learning process was checked, with the remaining values unchanged, as follows:

- During the first cycle of the second series, the gamma parameter changed from 0.995 to 0.8. This is a factor in how much an agent will consider potential future rewards. It must be lower than 1; otherwise, the learning process may be degraded, as represented in brown on the charts;
- The second cycle focused on the change in the number of hidden layers present after introducing the observations made by the agent and ranged from three to one—as marked below in light blue;
- The last, third, cycle consisted of changing the number of units in each layer of the neural network from 512 to 32, as characterised by the pink colour.

All the changes were made within the ranges provided in the documentation of the ML-Agents module to prevent the learning process from being carried out under unreasonable conditions.

3. Results

As indicators of comparison, we decided to use the average value of the reward function achieved by all the agents and the average number of steps for each episode in the learning environment. Both indicators were analysed as a function of the total number of steps taken during the learning period. The smoothed graphs are presented, preserving the original greyed waveforms in the background. Furthermore, except for the plot of the average duration of each episode for the first study, a logarithmic scale on the y-axis was used to capture the full spectrum within the plot.

The first set of results in Figure 5 was obtained for a study aimed at determining the impact of the reward function definition on the course of the learning process. The average cumulative value of the reward function for all the agents over the course of the study is presented below. It should be borne in mind that the absolute values of the reward function should not be analysed, owing to the different values in the formulas used for their calculation, but only the tendencies that each of the curves shows. As can be seen from this criterion, the process ran the best when the reward function was focused on the agent's correct behaviour during learning.

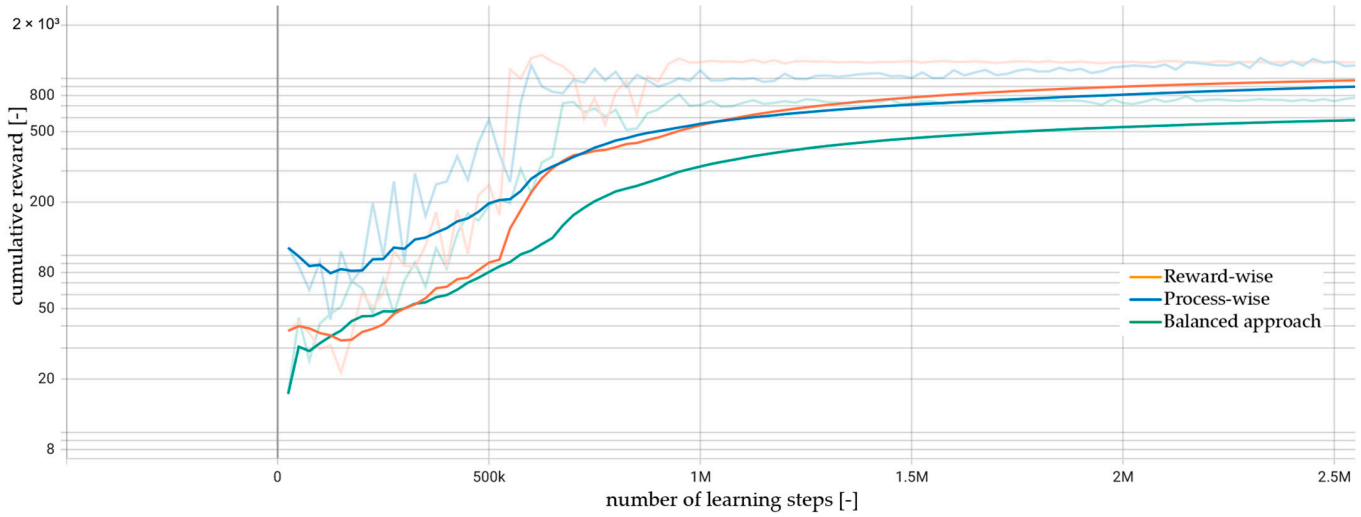

Figure 5. Graph showing the cumulative value of the reward function achieved during learning, depending on the form of the reward function.

The second graph, in Figure 6, shows the average number of steps for each episode in the learning environment. In this case, the goal–reward function turned out to be better; but, again, the process orientation was a better variant than the balanced-reward function.

In the next part, to examine the course of the learning process in more detail, we decided to present, in Figure 7, the dependence of the policy loss on the duration of the simulation. It is a value that determines how much the optimal behaviour determined by the agent deviates from the actual best policy in a given situation. According to the documentation of the ML-Agents module, the policy loss should maintain the lowest possible value; and, during learning itself, it should decrease and stabilise. This curve would be difficult to analyse owing to the unsmoothed waveform present on the graph in a damped form; therefore, the analysis itself was made on data smoothed by the exponential moving-angle method. As you can see, each of the waveforms behaves in accordance with the expected trend. At first, the green curve is characterised by the worst results in this analysis; but, over the course of the study, the balance between goal setting and the correct course of learning turns out to be the best approach in terms of policy loss.

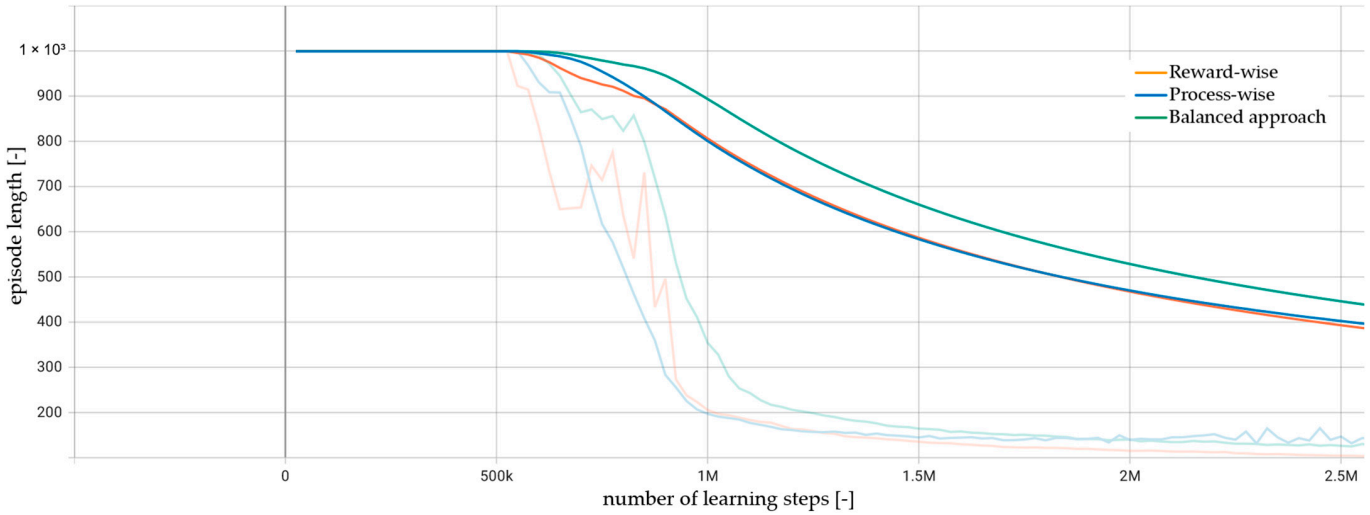

Figure 6. Graph showing the number of steps needed to achieve the goal throughout the learning process, depending on the form of the reward function.

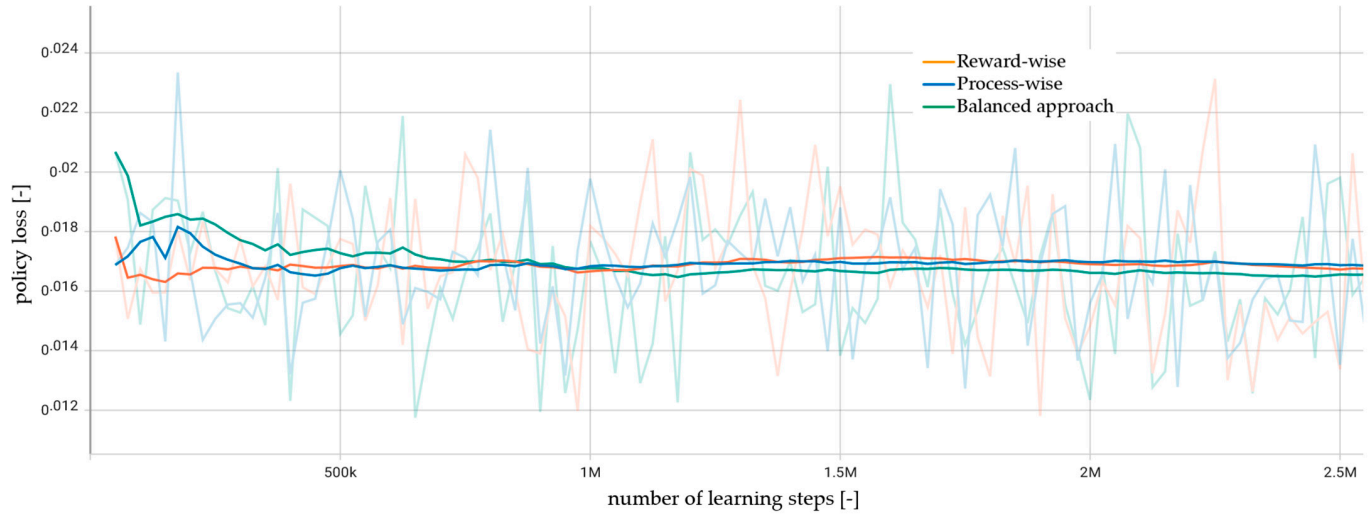

Figure 7. Graph showing changes in the agent's decision-making process over time depending on the form of reward function.

The last value taken into account in the analysis of the learning process in Figure 8 is the value loss. It describes the difference between the value of the reward function predicted by the machine-learning algorithm and the one achieved by the agent. According to the definition, this value should increase rapidly when the agent begins to achieve the goal and then gradually decrease later in the learning process. All the curves follow the expected course; but, as one might expect, the agent focused on achieving the goal in the shortest time possible to fulfil this task. However, in this case, the error value of the analysis remains at the highest level. Although the green curve is the last to reach the target, it shows a significant advantage over the others for the rest of the study in the form of the lowest value loss.

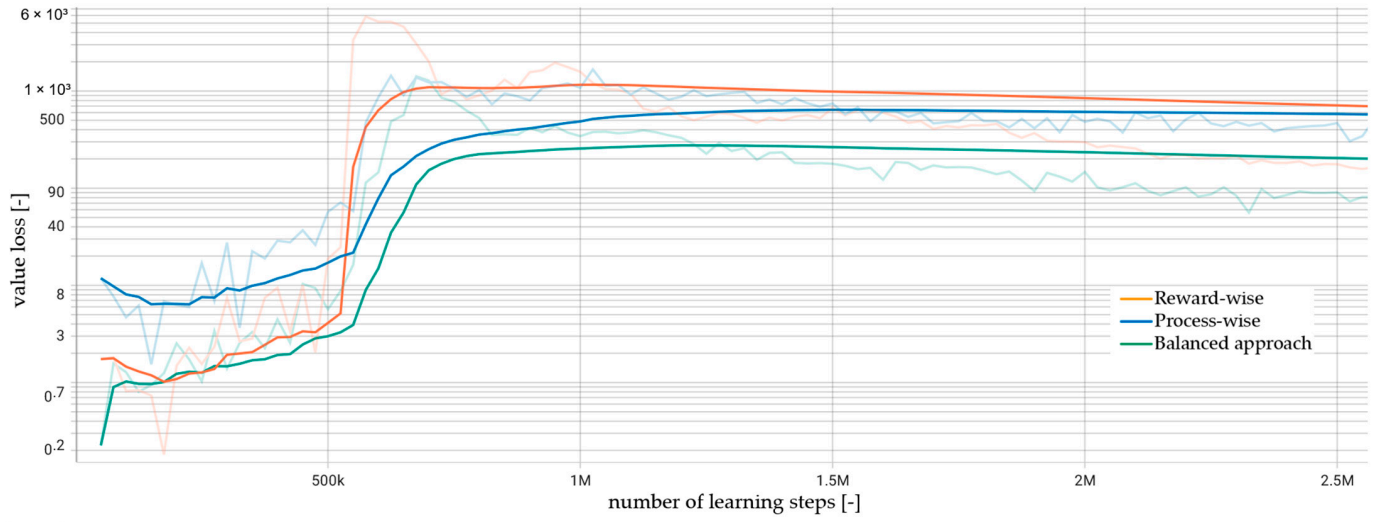

Figure 8. Change in the agent's ability to predict the outcome of actions over time depending on the form of reward function.

The following summary in Table 1 contains a comparison of the characteristic values of each of the learning cycles:

- Time from the start of learning until the agent reaches the goal for the first time (t_g);
- The average number of steps taken by the agent to reach the destination after learning (n_s);
- Percentage improvement in the number of steps required to reach the goal at the end of the learning process compared to its beginning (ns).

Table 1. List of parameters that characterise the course of the learning process for various forms of the reward function.

Number of Cycles	t_g (min)	n_s (-)	Δn_s (%)
1.	9.158	128.3	−87
2.	7.432	83.51	−92
3.	7.149	149.8	−85

For further research, the form of the reward function was used, in which multipliers were increased for the agent's behaviour during learning in accordance with the assumptions. The first graph presented in Figure 9 shows the average cumulative value of the reward function for all the agents over the course of the study. Owing to the same function of the reward this time, the comparison of the nominal results can be considered reliable. On this basis, it can be concluded that the best learning results were achieved when the number of hidden layers was reduced from three to one. However, the most rapid increase in the value of the reward function can be observed when the gamma hyperparameter is reduced.

The second graph in Figure 10 compares the average number of steps for each episode in the learning environment. Again, despite the faster achievement of the goal in cycle 2, fewer hidden layers allowed the maximum reduction in the number of steps needed to achieve the goal by the end of learning.

As in the case for examining the impact of the reward function on the course of learning, the analysis of hyperparameters also took into account the error values in the process. The first, described in Figure 11, is the loss of policy. Contrary to the graph of this value, this time, all the dependencies maintain the same relation. This means that the brown curve, corresponding to the modified gamma value, maintains the highest value both at the beginning and at the end of the learning process, while the pink curve has the lowest

value throughout the learning process. This means that in the policy loss category, the best results are achieved by maximising the number of units in each layer of the neural network.

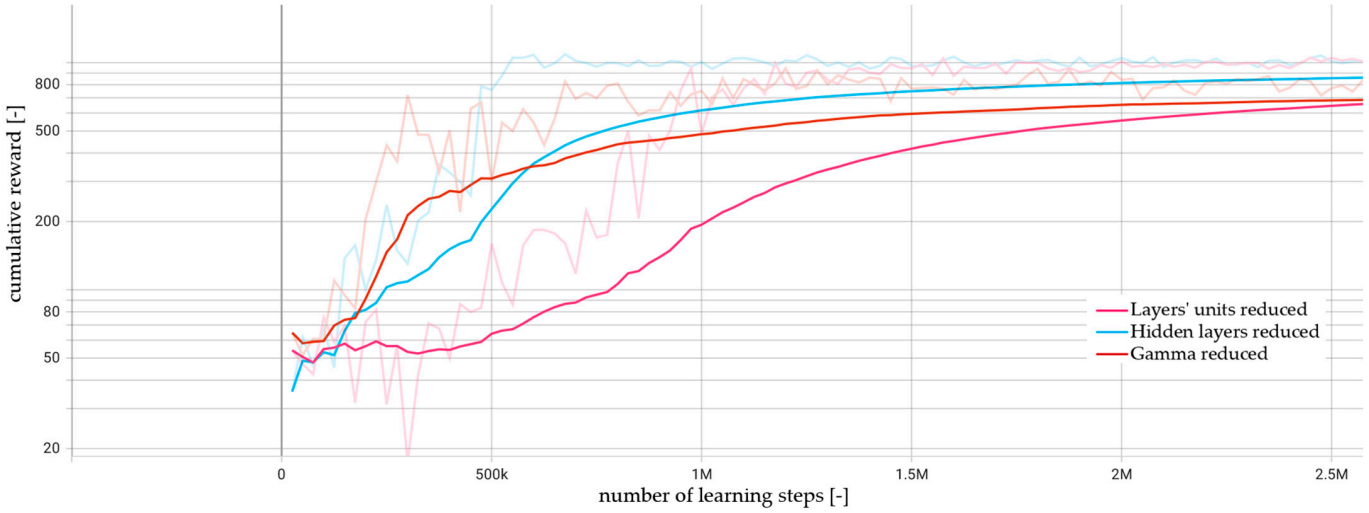

Figure 9. Graph showing the cumulative value of the reward function achieved during learning, depending on the modified hyperparameter.

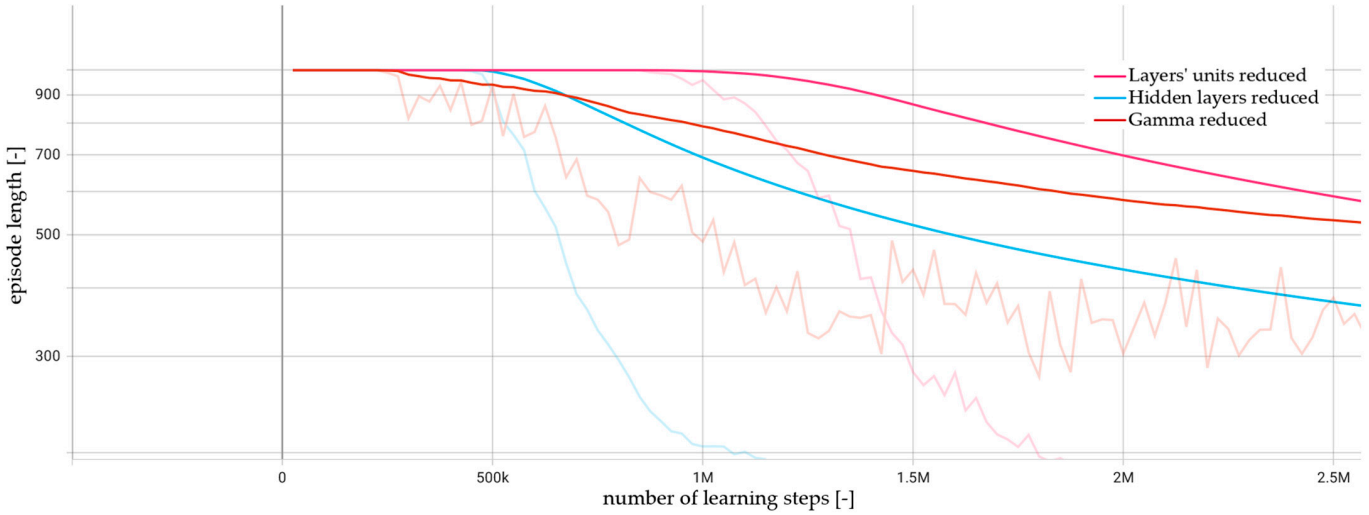

Figure 10. Graph showing the number of steps needed to achieve the goal throughout the learning process, depending on the modified hyperparameter.

The last graph analysed in the entire cross-section of the study in Figure 12 is the dependence of the value loss on the number of steps of the study. Although, in the case of the error analysed above, the pink curve showed by far the best trend and the brown curve the worst, in this case, the situation was completely reversed. Modification of the gamma factor led not only to the fastest achievement of the goal but also to the preservation of the value loss at the lowest level. A definitely undesirable tendency is characterised by the dependence representing the change in the number of units in the layers of the neural network. As you can see, not only did this agent start to achieve the goal at the latest time but also after achieving it, the error value, instead of decreasing, was constantly increasing. It should also be noted that of the three analysed curves, it is the one describing the decrease in the number of hidden layers that leads to the highest value losses.

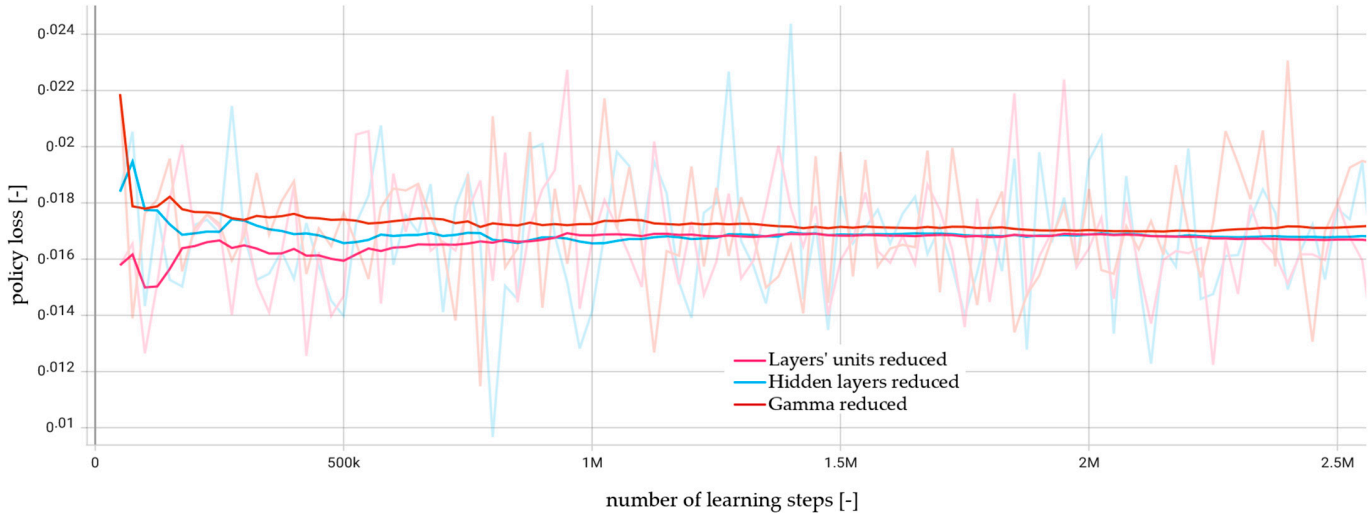

Figure 11. Graph showing changes in the agent's decision-making process over time depending on the hyperparameters values.

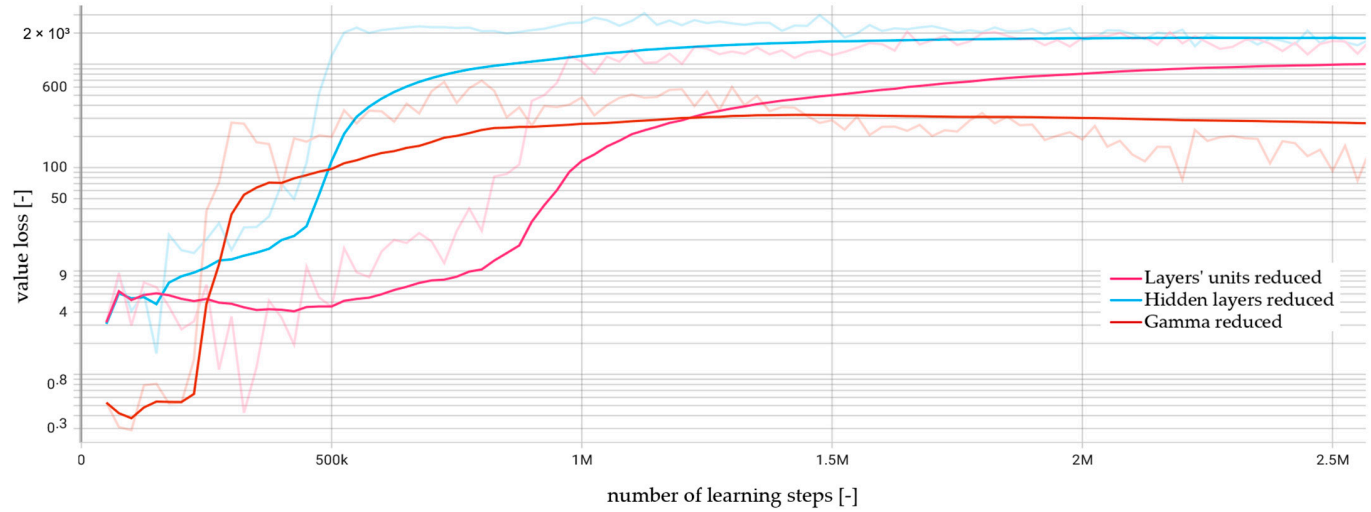

Figure 12. Change in the agent's ability to predict the outcome of actions over time depending on the hyperparameters values..

In Table 2, a summary of the characteristic values of each learning cycle is presented for the second study. The letter designations used are the same as those used above. However, owing to the constant formula that defines the reward function, a column was added showing the maximum value achieved, marked as f_r.

Table 2. List of parameters that characterise the course of the learning process during the modification of the configuration file.

Number of Cycles	t_g (min)	n_s (-)	Δn_s (%)	f_r (-)
1.	3.21	251.8485	−75	775.6
2.	6.037	129.2604	−87	935.3
3.	12.01	143.815	−86	840

4. Discussion

The first of the studies conducted focused on examining the impact of the reward function on the course of the reinforcement learning process. Owing to the differences in the absolute value of this function for each of the learning cycles, it is difficult to compare them directly; however, some general dependencies can be clearly seen in the graphs that compare the characteristic values of the process. During the first cycle, when parts of the reward function for achieving the goal and the robot's gait were the highest, the worst quality of learning is noticeable. The probable reason for this is that the fewest points are awarded to the agent for its actions. Too little motivation to achieve the goal also means that the longest time had to pass before the total reward began to grow rapidly.

During the second cycle, when the reward for achieving the goal is doubled, the beginning of effective learning is seen as the fastest of all the cases analysed. The downside of this solution is that such a heavy emphasis on achieving the goal minimises the robot's efforts to increase the reward function from other sources. This shows a much gentler slope of the cumulative reward curve at the end of the learning process.

The third cycle emphasised the acquisition of the reward function while walking to a greater extent compared to the previous cycles. This translated into a more delayed start towards achieving the goal but significantly decreased the final number of steps in the episode. Furthermore, the steep slope of the cumulative reward curve at the end of the learning process suggests that if it continues, the agent would be willing to further improve its score, in contrast to the situation in the second cycle. The values adopted in the third cycle were characterised as the best among those considered; and, on that basis, the second study was conducted.

It, again, consisted of three cycles, each of which examined the influence of a different hyperparameter on the course of the learning process. In the first cycle, the default value of the gamma parameter was changed from 0.995 to 0.8. This parameter, according to the documentation of the model that was used, is responsible for how far into the future the agent should go in the context of a potential reward. Typically, this parameter should have large values if the value of the reward function is to increase in the future; and, if the value increases immediately, gamma may be smaller. Of the three modifications analysed in the configuration file, this change, as well as increasing the reward for achieving the goal, led to the fastest increase in the reward function. However, from the perspective of the entire study, this modification resulted in the lowest cumulative value of the reward function and, at the same time, the most steps necessary for the agent to be able to achieve the goal.

In the second cycle, the number of hidden layers in the learnt neural network was reduced from three to one. In the case of this hyperparameter, it should be expected that the more complicated the learning environment, the more layers are needed for effective learning. On this basis, we can conclude that the created environment was interpreted by the trainer as not very complex because reducing the number of layers was the best modification among the analysed configuration file changes. Although the agent reached the goal for the first time after a longer time than in the case of reducing the gamma value, the number of steps needed for this and the value of the reward function itself quickly began to reach much more favourable values than those in the other cases that were analysed.

The last tested configuration was the change in the number of units in each hidden layer from 512 to 32. Numerous units are recommended in environments where there are many interactions between variables representing observations. The analysed case can be considered as such an environment because few units not only slowed the learning process but also deteriorated its quality (as indicated by the very low value of the cumulative reward).

On the basis of the above correlations, among the analysed configurations, a few hidden layers were considered in combination with setting the value of the reward function to optimal for the high quality of the agent's gait. Note that the performed tests do not eliminate all the possible parameter configurations. Therefore, it cannot be ruled out that

an untested configuration of the reward function combined with a change in one of the unmodified parameters would allow for a further improvement in the quality of learning. This leaves considerable room for further research in this field.

A useful functionality that could be added to the learning environment would be to create a heuristic method to control the robot. Because of this, before learning is started, it would be possible to test the defined relationships between the individual components of the robot. To achieve this, it would probably be necessary to develop a continuous sequence of movements that would be performed by the robot when a given movement is called.

An intriguing direction for further work would be to change the way the robot moves. As it stands, rotational motion is assumed at each joint of the robot due to the assumed use of rotational servos. However, it should be borne in mind that it would be possible to use linear drives (pneumatic or hydraulic), which would imply a change in the kinematic constraints defined in the scientific environment.

5. Conclusions

The best effects of reinforcement learning are possible when, through the reward function, it focusses on the correct course of the process more than on achieving the goal itself. Among the analysed hyperparameters, the reduction in the number of hidden layers had the most beneficial effect. It should be expected that with a more complex learning environment or a different type of chosen trainer, it would be necessary to choose other parameters for modification, as the currently configured ones could have the opposite effect.

This research area also shows considerable usefulness from a practical point of view. Simulating the robot's motion with the use of artificial neural networks in a computer environment would eliminate the need for multiple attempts using a physical prototype. Owing to this, the scale of the production of elements devoted to failed attempts would be limited. In addition, it would be possible to check the defined assumptions without endangering the environment and the robot itself from scratching.

A direction worth investigating seems to be the change in the mechanical aspects of the experiment. One of them may be the analysis of the influence of the number of limbs and their spacing on the obtained results. It should also be borne in mind that in the reward function defined by the authors, the robot is somehow forced to maintain an appropriate posture and direction of movement. Changing the mechanical structure would probably also require adjusting these factors.

6. Future Recommendations

Owing to the very extensive field of knowledge covered in this article, it is possible to specify many directions for the further development of the raised issues. The obtained results are characteristic of the type of learning selected with the use of neural networks, which was reinforcement learning. To make a cross-sectional review of the learning methods with the use of artificial intelligence, it would be possible to compare the chosen type of learning with supervised learning, unsupervised learning, and genetic algorithms.

The developed structure has been designed in a way that enables its physical implementation. In the case of transferring the learning results to a real robot, it might be necessary to design interactions related to friction on various surfaces in the simulation environment and to develop the circumstances of more complicated motion paths than just a linear trajectory on a flat surface.

Subsequently, it would be possible to use automatic tuning to match more hyperparameters and more configurations. However, in such a situation, it would be necessary to correctly define the criteria that would be interpreted by the tuning algorithm as desired. In addition, the greater their number, the longer the tuning time should be expected—a better overview is possible with a longer learning time.

Author Contributions: Conceptualization, A.K.; methodology, A.K., M.A. and P.B.; software, M.A. and P.B.; validation, A.K.; formal analysis, A.K.; investigation, M.A. and P.B.; resources, M.A. and P.B.; data curation, M.A. and P.B.; writing—original draft preparation, M.A. and P.B.; writing—review and editing, A.K., M.A. and P.B.; visualization, M.A. and P.B.; supervision, A.K.; project administration, A.K.; funding acquisition, A.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Polish Ministry of Science and Higher Education, grant number 0614/SBAD/1565.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors do not have any conflict of interest to declare.

References

1. Geva, Y.; Shapiro, A. A Novel Design of a Quadruped Robot for Research Purposes. *Int. J. Adv. Robot. Syst.* **2014**, *11*, 1. [CrossRef]
2. Shukla, A.; Karki, H. Application of robotics in onshore oil and gas industry—A review Part I. *Robot. Auton. Syst.* **2016**, *75*, 490–507, ISSN 0921-8890. [CrossRef]
3. Roman, H.T.; Pellegrino, B.A.; Sigrist, W.R. Pipe crawling inspection robots: An overview. *IEEE Trans. Energy Convers.* **1993**, *8*, 576–583. [CrossRef]
4. Qiu, Z.; Wei, W.; Liu, X. Adaptive Gait Generation for Hexapod Robots Based on Reinforcement Learning and Hierarchical Framework. *Actuators* **2023**, *12*, 75. [CrossRef]
5. Arents, J.; Greitans, M. Smart Industrial Robot Control Trends, Challenges and Opportunities within Manufacturing. *Appl. Sci.* **2022**, *12*, 937. [CrossRef]
6. Zhu, W.; Rosendo, A. PSTO: Learning Energy-Efficient Locomotion for Quadruped Robots. *Machines* **2022**, *10*, 185. [CrossRef]
7. Murphy, R.R. *Introduction to AI Robotics*; MIT Press: Cambridge, MA, USA, 2019.
8. Kajita, S.; Espiau, B. Legged robot. In *Springer Handbook of Robotics*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 361–389.
9. Fang, Y.; Wang, S.; Cui, D.; Bi, Q.; Yan, C. Multi-body dynamics model of crawler wall-climbing robot. *Proc. Inst. Mech. Eng. Part K J. Multi-Body Dyn.* **2022**, *236*, 535–553. [CrossRef]
10. Shi, Y.; Li, S.; Guo, M.; Yang, Y.; Xia, D.; Luo, X. Structural Design, Simulation and Experiment of Quadruped Robot. *Appl. Sci.* **2021**, *11*, 10705. [CrossRef]
11. Sokolov, M.; Lavrenov, R.; Gabdullin, A.; Afanasyev, I.; Magid, E. 3D modelling and simulation of a crawler robot in ROS/Gazebo. In Proceedings of the 4th International Conference on Control, Mechatronics and Automation (ICCMA '16), Barcelona, Spain, 7–11 December 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 61–65. [CrossRef]
12. Pfeiffer, F. The Tum Walking Machines. *Phil. Trans. R. Soc. A* **2006**, *365*, 109–131. [CrossRef] [PubMed]
13. Mahesh, B. Machine learning algorithms—A review. *Int. J. Sci. Res. (IJSR)* **2020**, *9*, 381–386. [CrossRef]
14. van Otterlo, M.; Wiering, M. Reinforcement learning and markov decision processes. In *Reinforcement Learning*; Springer: Berlin/Heidelberg, Germany, 2012; Volume 12, pp. 3–42, ISBN 978-3-642-27644-6. [CrossRef]
15. Kaelbling, L.P.; Littman, M.L.; Moore, A.W. Reinforcement learning: A survey. *J. Artif. Intell. Res.* **1996**, *4*, 237–285. [CrossRef]
16. Wiering, M.A.; Van Otterlo, M. Reinforcement learning. *Adapt. Learn. Optim.* **2012**, *12*, 729.
17. Shukla, N.; Fricklas, K. *Machine Learning with TensorFlow*; Manning: Greenwich, UK, 2018.
18. Hafner, D.; Davidson, J.; Vanhoucke, V. Tensorflow agents: Efficient batched reinforcement learning in tensorflow. *arXiv* **2017**, arXiv:1709.02878.
19. Zhou, Z.H. *Machine Learning*; Springer Nature: Berlin/Heidelberg, Germany, 2021.
20. Shinde, P.P.; Shah, S. A Review of Machine Learning and Deep Learning Applications. In Proceedings of the 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India, 16–18 August 2018; pp. 1–6. [CrossRef]
21. El Naqa, I.; Murphy, M.J. What Is Machine Learning? In *Machine Learning in Radiation Oncology*; El Naqa, I., Li, R., Murphy, M., Eds.; Springer: Cham, Switzerland, 2015. [CrossRef]
22. Eysenbach, B.; Salakhutdinov, R.R.; Levine, S. Search on the replay buffer: Bridging planning and reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 3.
23. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
24. Dayan, P.; Niv, Y. Reinforcement learning: The Good, The Bad and The Ugly. *Curr. Opin. Neurobiol.* **2008**, *18*, 185–196, ISSN 0959-4388. [CrossRef]
25. Barto, A.G. Chapter 2—Reinforcement Learning. In *Neural Systems for Control*; Omidvar, O., Elliott, D.L., Eds.; Academic Press: Cambridge, MA, USA, 1997; pp. 7–30, ISBN 9780125264303. [CrossRef]

26. Savid, Y.; Mahmoudi, R.; Maskeliūnas, R.; Damaševičius, R. Simulated Autonomous Driving Using Reinforcement Learning: A Comparative Study on Unity's ML-Agents Framework. *Information* **2023**, *14*, 290. [CrossRef]
27. Dung, V.D.; Hung, P.D. Building Machine Learning Bot with ML-Agents in Tank Battle. In *International Conference on Information Systems and Intelligent Applications. ICISIA 2022*; Lecture Notes in Networks and Systems; Al-Emran, M., Al-Sharafi, M.A., Shaalan, K., Eds.; Springer: Cham, Switzerland, 2023; Volume 550. [CrossRef]
28. Awoga, O. Using Deep Q-Networks to Train an Agent to Navigate the Unity ML-Agents Banana Environment (July 7, 2021). Available online: https://ssrn.com/abstract=3881878 (accessed on 27 July 2023).
29. Hung, P.T.; Truong, M.D.D.; Hung, P.D. Tuning Proximal Policy Optimization Algorithm in Maze Solving with ML-Agents. In *Advances in Computing and Data Sciences. ICACDS 2022. Communications in Computer and Information Science*; Singh, M., Tyagi, V., Gupta, P.K., Flusser, J., Ören, T., Eds.; Springer: Cham, Switzerland, 2022; Volume 1614. [CrossRef]
30. Zhang, B.; Rajan, R.; Pineda, L.; Lambert, N.; Biedenkapp, A.; Chua, K.; Hutter, F.; Calandra, R. On the Importance of Hyperparameter Optimization for Model-based Reinforcement Learning. In Proceedings of the 24th International Conference on Artificial Intelligence and Statistics, Virtual, 13–15 April 2021. Available online: https://proceedings.mlr.press/v130/zhang21n.html (accessed on 27 July 2023).
31. Białek, M.; Nowak, P.; Rybarczyk, D. Application of an Artificial Neural Network for Planning the Trajectory of a Mobile Robot. *J. Autom. Mob. Robot. Intell. Syst.* **2019**, *14*, 13–23. [CrossRef]