CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Elaborate Operational Requirements to Address Reward Hacking in Reinforcement Learning Agents

Bachelor of Science Thesis in Software Engineering and Management

SINA YAGHOOBZADEHTARI
COLIN OWUSU ADOMAKO
SIAVASH PAIDAR

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Supervisor: Pierguiseppe Mallozzi

Examiner: Rogardt Heldal

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

[Cover:
an explanatory caption for the (possible) cover picture
with page reference to detailed information in this essay.]

Elaborate Operational Requirements to Address Reward Hacking in Reinforcement Learning agents

Colin Owusu Adomako, *Siavash Paidar* and *Sina Yaghoobzadehtari*
Bachelor Thesis
Bachelor Program Software Engineering and Management
Department of Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden

Abstract

Autonomous agents, in recent times have been used to address several problems, but these agents in their course of achieving their task also emit side effects to the environment in which they operate. Paramount of these side effects is reward hacking. In this report, we try to address reward hacking using elaborate operational requirements. The results is evaluated on the unity machine learning platform using multi agents, a goalkeeper and a striker where the elaborate operational requirements helped address these agents from hacking or gaming their results.

I. Introduction

In the paradigm of reinforcement learning, a learning or autonomous agent learns to perform its task from interactions with its environment to receive a reward [1, 2]. At each time step, the agent observes the current state of the environment and performs an action, it receives a reinforcement value, also called a payoff or a reward, and a state transition takes place.

In recent times, autonomous agents are playing a pivotal role in most critical aspects of human life, ranging from the medical, transportation, Sports, aviation, manufacturing, mining, government and non-governmental sector and therefore quality requirements must be of priority [3] in order to a maximise their efficient utilisation. Notwithstanding the recent economic gains and successes chalked up with autonomous agents [6], these agents (robots) in the performance of task also pose risk to the environment in which they operate. A research into the risks posed by autonomous agents reveals five common defects, paramount of them, is reward hacking [4].

"Reward hacking" denotes an autonomous agent executing an unintended task to yield more reward. These agents in their line of duty tries to find a way to exploit problems in how the reward was specified to get high reward, whether its behaviour corresponds to the intent of the reward specifier or not [4]. When the agent discovers an easy way of gaining a reward, it will not be inclined to stop. This could be the result of wrong specification of reward function. For example, a soccer goalkeeper robot that is rewarded based on the number of goals scored is more likely to concede more goals since it will leave goal post in search of more goals. From the agents, point of view, this is not an illegitimate way of gaining a reward but simply how the environment works and thus a valid strategy like any other for achieving a reward. This could pose a big challenge especially if it must work on a long timescale and thus require urgent attention.

Early elicitation of operational requirements is vital for the successful design and development of the software [10] agent and could be adhered to address reward hacking. Operational requirements are those statements that "identify the essential capabilities, associated requirements, performance measures, and the process or series of deficiencies, evolving applications or threats, emerging technologies, or system cost improvements" [3]. .

This report, therefore seeks to address reward hacking in these autonomous agents using a goal model or Goal-oriented requirements engineering (a key concept of operational requirements). This is evaluated using the Unity Machine Learning platform in a soccer environment setting. The Unity Machine Learning platform is a suite of reinforcement learning environment. The goal model is be implemented using Goal-oriented Requirements Language (GRL) and Use Case Maps (UCM).

After evaluating the reward function implemented with goal model approach, it concludes that elaborate operational requirements can be implemented using goal modelling approaches to reduce reward hacking.

Concrete Problem

Software or Autonomous agents is their discharge of task, also engage in activities that pose risk (side effects) to the environment in which it operates. Since the reward system plays an essential role on how the agent achieves its task, it is a pressing AI safety need that require urgent attention [4].

II. Literature review

Reinforcement Learning

Reinforcement learning relates to a learning agent interacting with an environment at some discrete, lowest-level time scale, $t = 0, 1, 2,3\ldots$ At each time step, t, the environment is in some *state, st* $\in \{1,2,...,m\}$. The agent observes st and chooses an action, at, in response to which environment changes state to *st+1* and emits a reward, *rt+1* [16]. Below is a picturesque description of the the reinforcement learning framework [17].

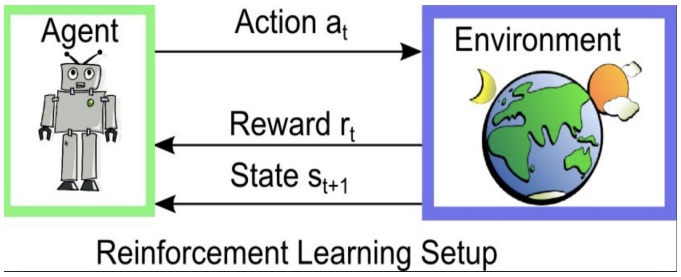

fig 1.0

Beyond an agent and the environment, one can identify four main sub elements of a reinforcement learning system: a policy, a reward signal, a value function and optionally, a model of the environment [18].

Elements of Reinforcement Learning

A Policy: A policy refers to the autonomous agent's way of behaving at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. It corresponds to what in psychology would be called a set of stimulus–response rules or associations. In some cases the policy may be a simple function or lookup table, whereas in others it may involve extensive computation such as a search process. The policy is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine behavior. In general, policies may be stochastic.

A Reward Signal: defines the goal in a reinforcement learning problem. On each time step, the environment sends to the reinforcement learning agent a single number called the reward. The agent's sole objective is to maximize the total reward it receives over the long run. The reward signal thus defines what are the good and bad events for the agent. In a biological system, we might think of rewards as analogous to the experiences of pleasure or pain. They are the immediate and defining features of the problem faced by the agent. The reward signal is the primary basis for altering the policy; if an action selected by the policy is followed by low reward, then the policy may be changed to select some other action in that situation in the future. In general, reward signals may be stochastic functions of the state of the environment and the actions taken.

Whereas the reward signal indicates what is good in an immediate sense, a value function specifies what is good in the long run. Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Whereas rewards determine the immediate, intrinsic desirability of environmental states, values indicate the long-term desirability of states after taking into account the states that are likely to follow, and the rewards available in those states. For example, a state might always yield a low immediate reward but still have a high value because it is regularly followed by other states that yield high rewards. Or the reverse could be true. To make a human analogy, rewards are somewhat like pleasure (if high) and pain (if low), whereas values correspond to a more refined and farsighted judgment of how pleased or displeased we are that our environment is in a particular state. Expressed this way, we hope it is clear that value functions formalize a basic and familiar idea.

Rewards are in a sense primary, whereas values, as predictions of rewards, are secondary. Without rewards there could be no values, and the only purpose of estimating values is to achieve more reward. Nevertheless, it is values with which we are most concerned when making and evaluating decisions. Action choices are made based on value judgments. We seek actions that bring about states of highest value, not highest reward, because these actions obtain the greatest amount of reward for us over the long run. Unfortunately, it is much harder to determine values than it is to determine rewards. Rewards are basically given directly by the environment, but values must be estimated and re-estimated from the sequences of observations an agent makes over its entire lifetime. In fact, the most important component of almost all reinforcement learning algorithms we consider is a method for efficiently estimating values. The central role of value estimation is arguably the most important thing we have learned about reinforcement learning over the last few decades.

Environment: The third and final element of some reinforcement learning systems is a model of the environment. This is something that mimics the behavior of the environment, or more generally, that allows inferences to be made about how the environment will behave. For example, given a state and action, the model might predict the resultant next state and next reward. Models are used for planning, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced. Methods for solving reinforcement learning problems that use models and planning are called model-based methods, as opposed to simpler model-free methods that are explicitly trial-and error learners—viewed as almost the opposite of planning.

Forms of Reward Hacking

Autonomous agents are said to have gamed or hacked results when it gets a reward for a task or goal it has not

accomplished. This can happen in a different ways. In this section, the various forms of gaming a reward presented as elaborated by Amodei et al.

Partially Observed goals: It is assumed that, in reinforcement learning systems, rewards are directly experienced, even if the other parts of the environment are partially known. Since the agent lack access to perfect measure of task performance, the engineer has no option than to design a reward that represent a partial or imperfect measure. An example is an autonomous soccer striker that is reward for preventing goals. In such situation the striker agent will not score goals. This imperfect objective function can be hacked, that is, the robot can create a mess to clean in order to gain more points.

Complicated Systems: Since emerging autonomous agents are made up of complicated systems, the tendency of having a high complexity of the program. And the higher the complexity, the higher the probability that there is a viable hack affecting the reward function. An example is the possibility in principle for an agent to execute arbitrary code.

Abstract Reward: Complex reward functions will need to refer abstract concepts which include checking whether a conceptual goal has been met. These concepts will possibly need to be learned by models like neural networks, which can be vulnerable to adversarial counterexamples. A learned reward function over a high-dimensional space may be vulnerable to hacking if it has pathologically high values along at least one dimension.

Goodhart's Law: There is also a possibility of reward hacking occurring, if a designer or engineer chooses an objective function that is seemingly highly correlated with accomplishing the task, but that correlation breaks down when the objective function is optimised. An example is a designer might notice that under circumstance, a cleaning robot's success in cleaning up the office is proportional to the rate at which it consumes cleaning supplies such as detergents. Therefore, if we base the robots, on this measure, it might use more detergents than it needs, simply by disposing the bleach to give appearance to success.

Feedback Loops: it has been realised that sometimes an objective function has a component that can reinforce itself and eventually getting amplified to the point where it draws out or severely distorts what the designer intended the objective function to represent.

Existing Solutions to Reward Function

These problems might not occur in today's simple systems or can be solved without much harm as part of an iterative development process. Below are the various current techniques by which gaming or hacking reward can be solved:

Model Lookahead: With this model, the autonomous agent consider future states sequence actions may lead and use that as its future plan. And it could be rewarded based on anticipated future states, rather than the anticipated future states in different setups. This could be very helpful in resisting situations where the model overwrites its reward function. This such instances, the reward cannot be controlled once it replaces the reward function, but it can be given a negative reward for planning to replace the reward function.

Adversarial Binding**:** These techniques can be used to blind a model to certain variable. It could be used to make it impossible for an agent to understand some parts of its environment, or even to have mutual information with it (or at least to penalise such mutual information). It could prevent an agent from understanding its environment. This solution could be described as "cross validation for agents".

Adversarial Reward Functions: Machine Learning system has an adversarial function and would find any means possible in exploiting problems in how the reward was specified in order to get more rewards whether or not its behaviour corresponds to the intent of the reward specifier. However, machine learning system, agents are powerful whilst reward function is a static object that cannot respond to the system's attempt to game it. If instead the reward function were its own agent and could take actions to explore the environment, it might be much difficult to game it. To solve this, reward checking agents must be made more powerful than the agent that is trying to achieve the rewards. There may also be situations where a system has multiple pieces trained using different objectives that are used to check each other.

Trip Wires: It is of importance to know whether an agent is going to try and hack its reward function. By doing that, the designer or engineer could introduce some plausible vulnerabilities and monitor them, alerting us and stopping the agent immediately if it takes advantage of one. Though such "trip wires" do not solve reward hacking in itself, it may reduce the risk or at least provide diagnostics.

Careful Engineering: According to the paper [4], some kinds of reward hacking, like buffer overflow, might be avoided by very careful engineering. This could be achieved through formal verification and testing of all parts. Computer security approaches that attempt to isolate the agent from its reward signal through a sandbox could also be possible. It may also be possible to create highly reliable" core" agent which could ensure reasonable behaviour from the rest of the agent.

Operational Requirement Models

One of the key processes in the development phase is to define the operational requirements of the system. Operational requirements are those statements that "identify the essential capabilities, associated requirements, performance measures, and the process or series of deficiencies, evolving applications or threats, emerging technologies, or system cost improvements" [9]. Operational requirements assessment starts with the Concept of Operations (CONOPS) and goes to a greater level of detail in identifying mission performance

assumptions and constraints and current deficiencies of or enhancements needed for operations and mission success [6]. Operation Requirements forms the basis for system requirements. It must also be emphasised that there is a positive correlation between system requirements and software quality. Therefore, one assured way of addressing the current AI defects or challenges in reinforcement learning, is to use some improved operational requirements.

The operational requirements for an agent will specify under which conditions it must operate and what actions it must take depending on the setup. This can be correctly designed with the help of a goal model. Goal models play a prominent role in the operational requirements process. They provide a rationale for requirements; a requirement exist because of some underlying goal which provides a base for it [10]. There two types of frameworks in which goals can be modelled, that is, the formal [11] and qualitative. In this report, we will dwell on qualitative framework.

In qualitative framework, the achievement of"soft" goals are not absolute since they do not satisfy any clear goal but contributes partially to the realisation of the overall goal. If a goal is AND- decomposed into subgoals or "soft" goals and they are all satisfied, then the goal is satisfied and vice versa. In the AND/ OR goal graph, the goals are indicated by names, parameters, and degree of satisfaction or denial by sub goals.

In this paper, we will resort to Goal-oriented Requirements Language and Use Case Maps. The Goal-oriented Language (GRL) is used for modelling goals and other intentional concepts (mainly for non-functional requirements, quality attributes and reasoning about alternatives and tradeoffs). It focuses on the why questions of a system.

The Use Case Map (UCM) notation is a visual modeling language that allows the high-level description of object-oriented systems. It was first introduced by Buhr and Casselman in the mid- 1990s. Over the years UCM notation has gained attention from both researchers and industry. It has been successfully used for telecommunication systems, web applications, agent based systems) and operating systems.

A UCM consists of one or more paths each of which represent a use case scenario. A path starts at a start point (filled circle) and ends at an end point (bar).The actions performed by the system or use case actor along these paths are responsibilities (cross).These responsibilities can be bound to components—actors, agents, teams, objects and processes. Anactor component (rectangle including a stickman) represents a stakeholder who is associated with the system through a number of usage scenarios. Software agents in agent-oriented systems can be represented by the agent component (rectangle with a dark border). Teams (rectangle) represent high level abstract components that can be further decomposed into multiple levels of other component types. However, objects (box with rounded corners), which represent instances of a class, cannot be further decomposed. Processes (slanted rectangle) are executing components of a system and may include object components. An OR-fork divides a path into one or more alternative paths based on a guard condition. Concurrent paths emerge from AND-forks (bar). Common paths are merged by OR- joins and concurrent paths are synchronized by AND-joins (bar). Stubs (diamond) are containers for nested maps. Stubs are useful for refactoring complex UCMs via modularization. Erroneous situations that may stop the flow of a path are

represented by failure points (ground). Timers(clock) express the amount of time to wait before a path can progress further. A waiting place (filled circle and bar) allows a path to wait for another path to finish before it can continue. The interested reader may refer to Buhr and Casselman"s (1996) book on UCMs for more details on its notation.

III. METHODOLOGY

Research Questions

RQ1: How state-of -the-art goal modelling approaches can be used to model a reward function?
RQ2: How can we ensure that an autonomous agent will never game its function?

For example, if we reward a goal keeping agent for the number of goals scored, it will neglect its goal keeping task to score more goals in order to gain more rewards.

Strategy for the investigation of the problem (Design Science)

A research method aims to guide researchers in the search for necessary answers to the proposed research problem [7]). To achieve the objectives of this report or answer the proposed questions, we will adopt a design science approach. Below is a description basic design science principles that were adhered to in this research paper.

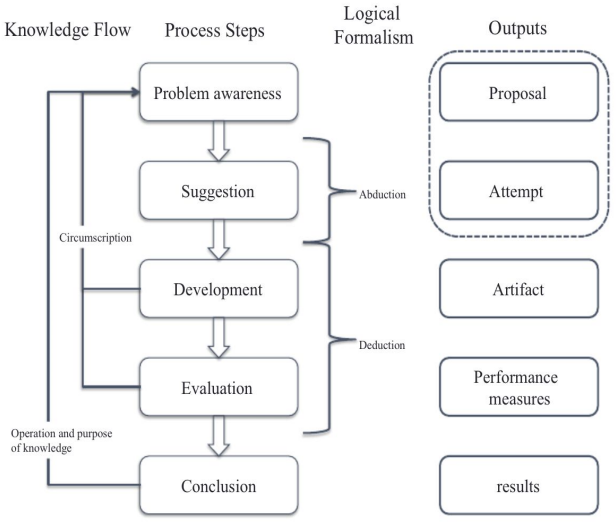

fig 3.0

Phase 1: Problem Awareness

Since design science approaches first focuses on first clarifying the goals of an artifact and then building on its validity [8], in this phase, we will be concerned with systematically assessing whether a given reward function satisfies a given operational requirement specification. It must be emphasised the sources of information for this study are execution logs from training RL agents using the unity machine learning platform.

The goal of the reinforcement agents are as specified below:

Goalkeeper: To prevent the ball from entering its goal net.

Striker: To score goals and to defend.

In our case, after assessing the performance of the agents on the platform, we realised that the both agents were gaming their rewards. The goalkeeper ws gaming its reward by leaving the goal area and the striker was also gaming its reward by moving its goal area which clearly yields undesirable results.

Phase 2: Suggestions

This phase requires suggesting elaborate requirements from the counterexample generated from the problem awareness phase. For examples, how the system should and should not behave. The next thing will be to elaborate a set of positive and negative scenarios from a given violation trace. In the case of the goalkeeper, the elaborate requirements will to defend the ball from entering its goal net. Also it shall not go beyond the penalty box area in order to be able to defend the area.

The striker on the other hand shall score goals and defend within the midfield area. Furthermore, it shall not go within its own penalty box area in order to be well placed all the time to score goals.

Phase 3: Development of Rewards Allocation

At this juncture the reward function is implemented or developed using a goal model approach where rewards and penalties shall be allocated for various tasks as as specified in phase . For example if the striker scores a goal a it will be rewarded with one point whilst when they concede a goal a point shall be deducted from its accumulated reward. The same is done for the goalkeeper.

Phase 4: Training and Evaluation

With the reward function implemented in phase 3, the autonomous agents shall be trained to ascertain the efficiency with which it accomplishes task or whether it can game or hack its reward by accomplishing unintended task to achieve more rewards. In the case of the autonomous goalkeeper, since it was rewarded the same points for defending and scoring goals, it was abandoning its goalkeeping task to score goals which also yielded more points to it. By so doing, it left its team vulnerable to conceding a goal. This was not the task assigned to the autonomous goalkeeper so it considered as reward hacking.

After this, a set of elaborate operational requirements will be derived and added to the selected requirements or specifications in phase one and two. This will help strengthen the reward function to address the reward hacking or prevent the agent from gaming its reward. Since the goalkeeper was hacking its reward by moving to the opponents goal area to score goals, there was an additional specification to give less points for scoring goals and more points for defending goals. Also, since it was not to move to the opponents goal area, we placed existential penalties for moving beyond its goal area.

The results of this training shall be evaluated using a case study on Unity Machine Learning platform and an evaluation framework within the Unity platform and a python evaluation framework called pythotorch.

Phase 5: Conclusion

The initial reward function which is being hacked or gamed by the agent will be compared with the goal modelled reward function to ascertain whether the autonomous agents will be able to still game their reward. By comparing the set of system goals and specification to how the agents execute their task we will find out which method (the initial or goal modelled reward function) best satisfies the system goals. We can then conclude based on the conclude based on the evaluated report from the training of the autonomous agents.

IV. Analysis

Operational requirements of a soccer autonomous agent

An implicit operational requirement inherent in every reinforcement agent is to be able to interact with the outside world, evaluate its interactions with the environment, remember what is important and adjust actions on the basis of current interactions and earlier recollections. Similarly, the soccer agent or robot must be capable of:

- Interacting with its environment, requiring some form of perceptions and some means of altering itself or the environment.
- Evaluate its interactions with the environment.
- Storing these evaluated interactions or perceptions, popularly referred to as “memory”.
- Adjusting its interactions based on the evaluated values attained through the previous interactions or perceptions, colloquially referred to as “learning” [14].

It is worth emphasising that these three requirements interact and overlap, allowing for complex and changeable

behaviour of the soccer agents. Hence these requirements do not exist in a vacuum.

In the following paragraphs, the paper discusses how these requirements are met in the soccer agents in order to address the reward hacking.

Interaction with environment

Sensors play an essential role in the robot's interaction with the environment. It will drive both the hardware and software requirements. It is argued that the surest way to guarantee coverage, reliability and safety for the lowest overall cost is to use many redundant, low-cost sensors. Some sensors will be used in mapping and, position estimation. Example of the sensors are passive light detection, ultrasonic sonar, odometer, reflective IR, pyro-IR, compass system, to mention but a few.Additional sensors will be used to cope with operational contingencies. These would drop-off detectors, inclinometers, moisture indicators, a lift detector, an external temperature sensor, a remote-control receiver and a charging-dock.

To enhance the vision or perceptions of the autonomous agent, low cameras will be used for cost effectiveness. A structured-light sensing system may also be cost effective if the light intensity could be used kept at a safe level.

The vision of the agent will me mapped in a grid-like environment for the robot to plan its movement. Each step leads the agent into a in grid-box. The image below is a representation of the how the agents environment is mapped (gridworld).

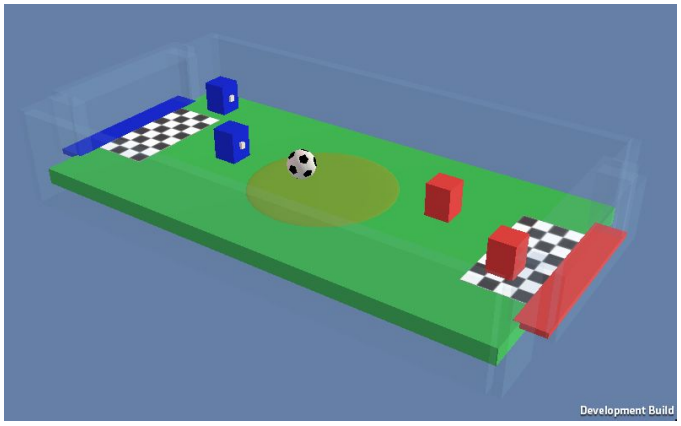

fig. 2

Evaluating Interactions

The autonomous agent must capable of creating and maintaining internal models of its soccer field. It must also be capable of planning and executing an efficient path.

There could be multiple scenarios in a soccer field environment but the agent continues throughout the environment, it starts eliminating the other scenarios until it finally lands on the most likely possibility. The word likely is emphasised because the autonomous agent has the tendency of doubting itself and re-localize if it finds itself elsewhere.

The agent will also record the telemetry data along with its results to analytics data warehouse which will be evaluated or analysed using statistical and machine learning techniques to provide him with more accurate mapping and paths for the next game.

Storing evaluated Interactions

The agent is equipped with a memory to save its operation or activities. An operation with the highest value will supersede all operations in memory and is readily available whenever the autonomous agent must engage in similar operation.

Adjusting its interaction based on previous evaluations

As aforementioned, the autonomous agent is capable of localizing its environment making it possible to adjust to current environment conditions. The numerous pre-installed scenarios also make it possible for the agent to make suitable adjustments.

Also with the help of its memory, it can recollect the operation with the highest value and adjust its current operation to suit the one in memory.

Initial Reward Function for Soccer autonomous agents

Goalkeeper

- ***Goal***
  - ➔ *Prevent the ball from entering its own net.*
- ***Reward Function***
  - ➔ *+1 When the enters opponents's net.*
  - ➔ *-1 When ball enters own team's net.*
  - ➔ *-0.001 Existential penalty*

Striker

- ***Goal***
  - ➔ Get the ball into the opponents net.
- ***Reward Function***
  - ➔ +1 When ball enters opponent's goal.
  - ➔ -1 When ball enters own team's goal.
  - ➔ -0.001 Existential penalty.

Problem Awareness

When this reward function was implemented on the Unity Machine Learning platform, several observations were made.

This includes the goalkeeper leaving the goal post in search of goals since it got more reward that way. This rendered the team vulnerable . Also since the striker lost the same points as the

goalkeeper, it also mostly dropped too deep to defend against conceding a goal although that's not its task.

This unintended behaviour of the autonomous goalkeeper and striker led to an implicit elicitation of more operational requirements.

Development of a Reward Function using a goal model

After several deliberations and consensus (Suggestion) building, we emerged with a goal model appraoch to help address the hacking of reward by the autonomous agents.

The diagram below is a representation of the goalkeeper's goal model.

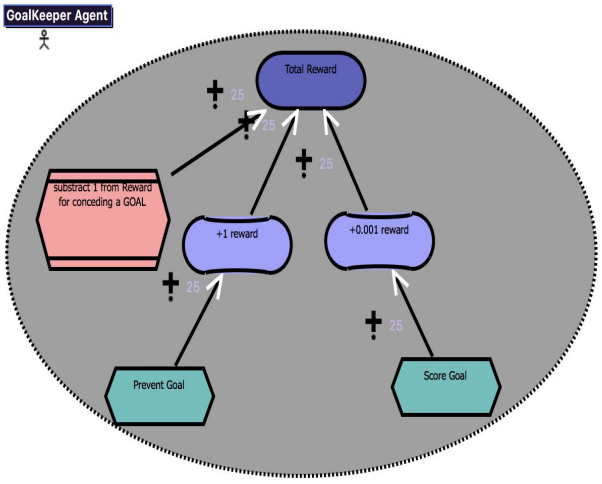

fig 4.0 Goalkeeper Agent Goal Model

From the above diagram, it could be realised that, the goalkeeper is reward 1 point for preventing a goal and 0.001 for scoring a goal. This makes it unattractive for the goalkeeper to score a goal and thereby keeping to its goalkeeping task. Also, it could be seen that it was penalised for by subtracting 1 from its cumulative reward. By this model, we then came up with the following requirement and reward function for the goalkeeper autonomous agent.

Goalkeeper

- ***Goal***
  - ➔ *Prevent the ball from entering its own net.*
- ***Reward Function***
  - ➔ *+0.1When the enters opponents's net.*
  - ➔ *-1 When ball enters own team's net.*
  - ➔ *+1 When ball is prevented from entering own team's net.*
  - ➔ *-0.001 Existential penalty.*

Striker Agent

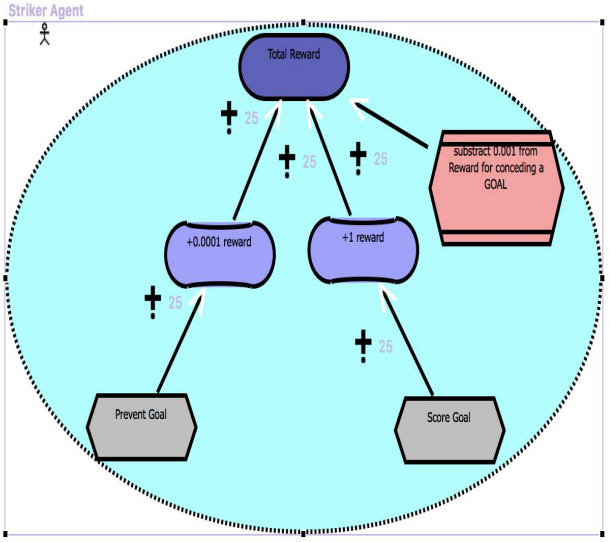

fig 4.1 Striker Agent

Figure 4.1 is a goal model for the striker agent. From this diagram, it can be realise that the striker is also rewarded 1 point for scoring a goal and less points for defending a goal (0.001), thereby making it more rewarding to score a goal than to defend a goal. With in place, the striker is compelled to score goals for maximum points since thats its ultimate aim (to maximise its reward).

Striker

- ***Goal***
  - ➔ Get the ball into the opponents net.
- ***Reward Function***
  - ➔ +1 When ball enters opponent's goal.
  - ➔ -0.1 When ball enters own team's goal.
  - ➔ -0.001 Existential penalty.

In addition to this reward function, the goalkeeper's movement was restricted or confined to the goal area. Below is a representation of this using a Use Case Map (UCM).

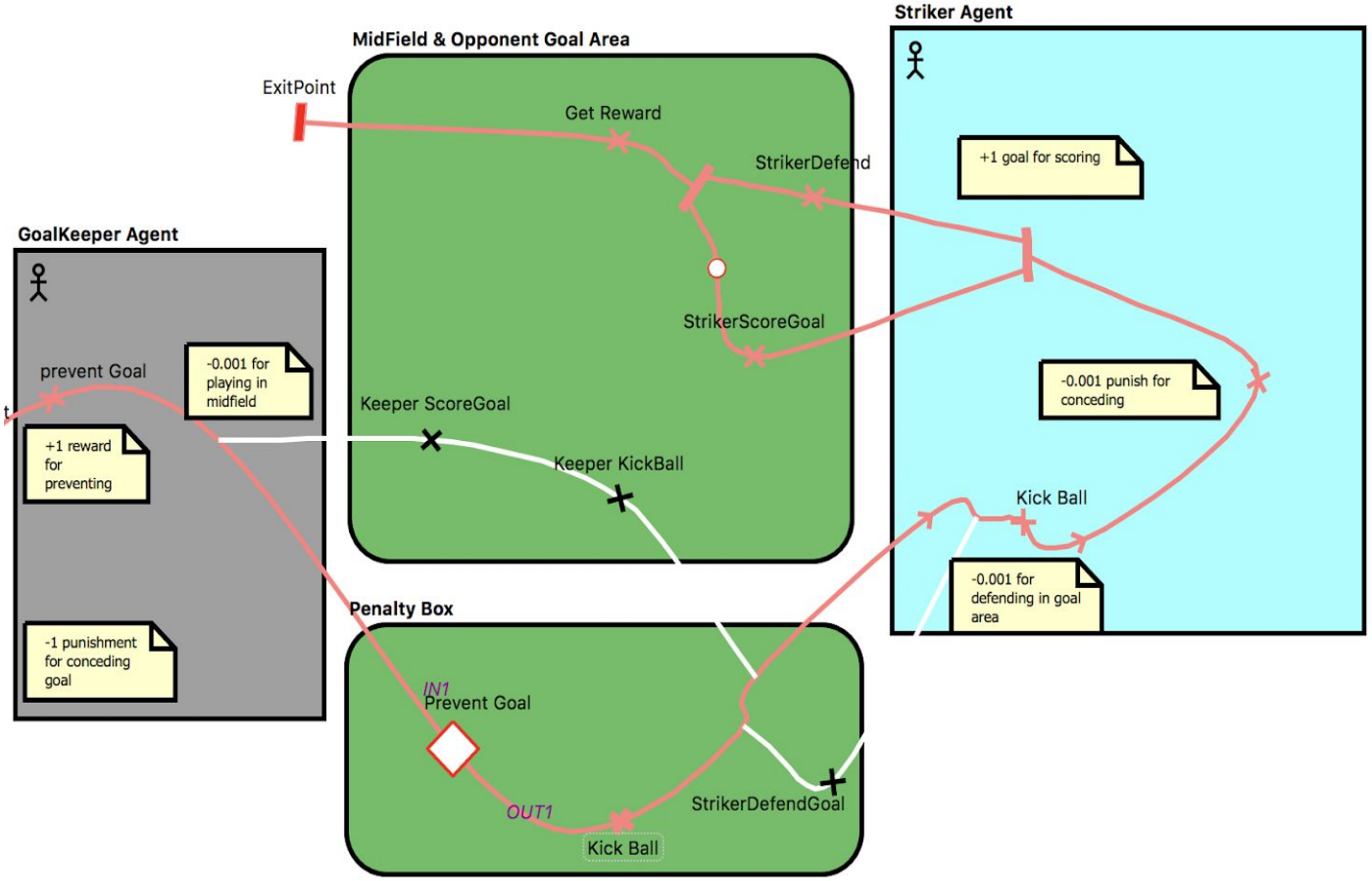

fig 4.3

The red path exhibits the accepted path of both the goalkeeper and the striker autonomous agent whilst the white path depicts the existential penalties. It could be realised that the goalkeeper has is now confined to stay stay within the penalty box area to execute its goalkeeping task. The white path from the goalkeeping autonomous agent is an illustration of its existential penalty, meaning that whenever it goals to the midfield area, it is penalised. And vice versa, the red path from the striker autonomous area is an illustration of its permissible path. That means the striker is also confined from the midfield area to the opponents goal area. The white path leading to the its goal area is also an illustration of its existential penalty, meaning that its not allowed to move to its own goal area to defend since that the task of the goalkeeper autonomous agent.

After the implementation of this reward function, it was realise that the goalkeeper was performing its intended function since it was highly rewarded (+1) for balls saved from entering the net and less reward for scoring (+0.001). It was also highly penalised

for conceding for a goal. On the other hand, the striker got more reward for scoring and less punishment for conceding a goal, thereby sticking to its goal scoring task.

Below is the evaluation report before and after modelling the elaborate operation requirements was modelled for the goalkeeper agent. The red and blue lines represent before and after modelling the elaborate operation requirements respectively.

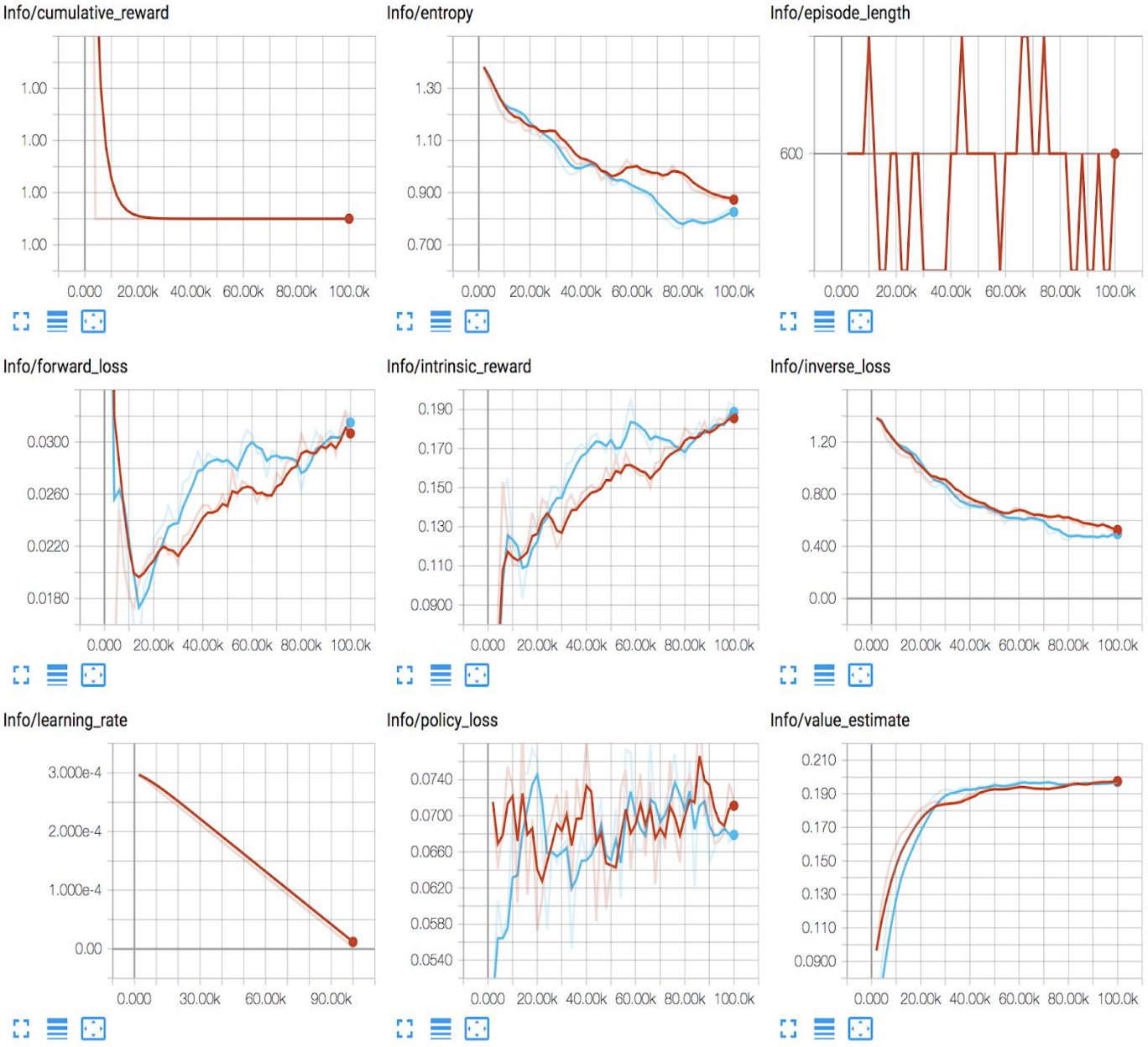

fig 5 evaluation report for goalkeeping autonomous agent.

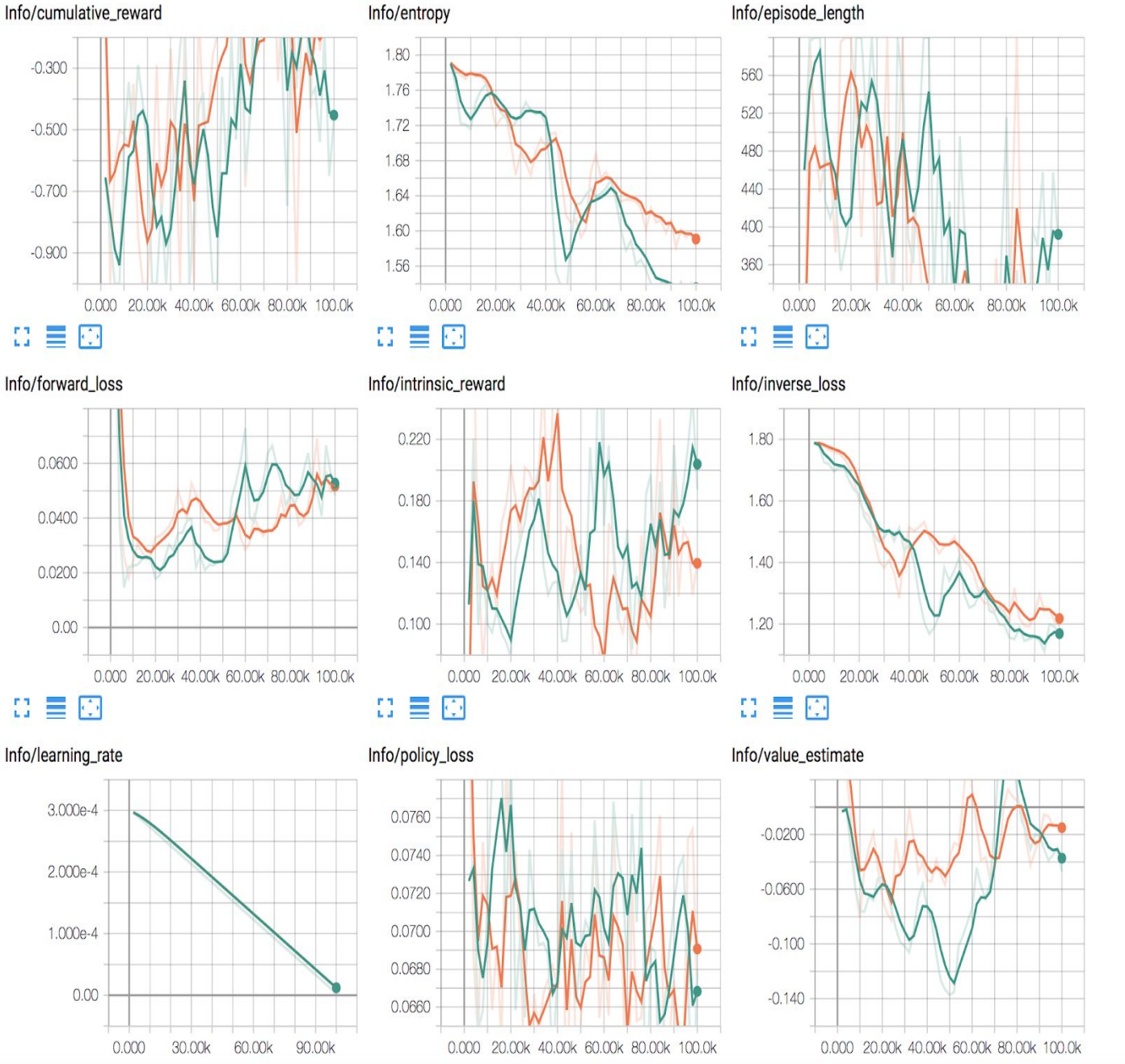

fig. 6 evaluation report for goalkeeping autonomous agent.

Discussion of Evaluation Report on the Training of the autonomous agents

The red lines illustrates the evaluation of the goal modelled reward function whilst green and blue represents the initial reward function for the goalkeeåer and striker reward function respectively.

Below is the the meaning of the various graphs:

- Cumulative Reward - The mean cumulative episode reward over all agents. Should increase during a successful training session. From both graphs it could be realised that, there was not a significant difference between the cumulative rewards hence the graph for the goal modelled reward function exhibits a higher cumulative reward than the initial graph.
- Entropy - How random the decisions of the model are. Should slowly decrease during a successful training process. If it decreases too quickly, the beta hyperparameter should be increased.
- Episode Length - The mean length of each episode in the environment for all agents.

- Learning Rate - How large a step the training algorithm takes as it searches for the optimal policy. Should decrease over time. In both the initial and the modelled goal function, it is realised that the agents learned their task to its maximum.
- Policy Loss - The mean loss of the policy function update. Correlates to how much the policy (process for deciding actions) is changing. The magnitude of this should decrease during a successful training session.
- Value Estimate - The mean value estimate for all states visited by the agent. Should increase during a successful training session.
- Value Loss - The mean loss of the value function update. Correlates to how well the model is able to predict the value of each state. This should decrease during a successful training session.

RQ1: How state-of -the-art goal modelling approaches can be used to model a reward function?

From the evaluation of results, it could be realised that there was not significant difference in the cumulative and intrinsic rewards for before and after the goal modelling. However, the autonomous agents trained with the goal modelled reward function perform their respective tasks efficiently without gaining undesirable rewards (*refer to appendix for platform*). Therefore by using methodological goal modelling approaches like UCM paths and Goal-Oriented Language (KAOS), a reward function can be implemented efficiently.

RQ2: How can we ensure that an autonomous agent will never game its function?

The correct and early elicitation of the agents operational requirements can help address reward hacking. This prevents wrong specifications in the reward function as clearly demonstrated by the goalkeeper and striker autonomous agents.

V. Conclusion

From the above evaluation and the experiment on the Unity Machine learning platform, it is evident and can be concluded that, Goal Models and early elicitation of the autonomous agents specifications as specified in [10], can help address reward hacking. This is because, having applied the goal model approach in the formulation of the reward function, the goalkeeper and striker autonomous agents were not able to perform undesirable task in order to gain more reward.

We propose future research, to be expanded further to see which specific goal model approach is the best fit in addressing reward hacking in reinforcement agents.

APPENDIX

Platform for evaluation
https://github.com/Unity-Technologies/ml-agents/tree/master/unity-environment/Assets/ML-Agents/Examples/Soccer

References

[1] R. S. Sutton, A. G. Barto and R. J. Williams, "Reinforcement learning is direct adaptive optimal control," in *IEEE Control Systems Magazine*, vol. 12, no. 2, pp. 19-22, April 1992.

[2] L. J. Lin, Self Improving reactive Agents based on Reinforcement Learning, Planning and Teaching Volume 8, Issue 3–4, pp 293–32

[3] C. Lindholm, J. Pedersen and M. Höst, A case study on risk analysis and planning in medical device development. Springer Science+Business Media New York, 2013.

[4] D. Amodei, C Olar, J. Steinhardt and P. Christiano, J. Schulman and D. Mane. Concrete Problems in AI Safety, 2016

[5] K. Arulkumaran, M. P. Deisenroth, M. Brundage and A. A. Bharath. A Brief Survey of Deep Reinforcement Learning, *IEEE SIGNAL PROCESSING MAGAZINE*, 2017

[6] A. Kossiakoff, and N. Sweet, Systems Engineering Principles and Practices, Hoboken, New Jersey, John Wiley & Sons, 2003

[7] Saunders, M., Lewis, P., & ornhill, A. (2012). Research methods for business students (6th ed.). London: Pearson Education

[8] Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. MIS Quaterly, 28(1), 75-105

[9] A. J. Champandard. Reinforcement Learning. 2002. *Retrieved From* http://reinforcementlearning.ai-depot.com

[10] A. V. Lamsweerde, Building Requirements Models for Reliable Software, Unpublish Report, *Retrieved From:* https://pdfs.semanticscholar.org/4ee1/9929335a964eb15e9af4660c2af0be92ae51.pdf

[11] A. Dardenne, A. van Lamsweerde and S. Fickas, "Goal-Directed Requirements Acquisition", *Science of Computer Programming*, Vol. 20, 1993, 3-50.

[12] N. R. B. Perdijk, Artificial Reward and Punishment, Grounding Artificial Intelligence through motivated learning inspired by biology and the inherent consequences for the Philosophy of Artificial Intelligence**.** Unpublished master's thesis. Retrieved from: https://dspace.library.uu.nl/handle/1874/301226

[13] F. Jenkins, Practical Requirements for a Domestic Vacuum-Cleaning Robot, AAAI Technical Report FS-93-03, 1993

[14] P. Shukla and S. L. Shimi, Design of inspection and cleaning robot, International Journal of Scientific Research Engineering & Technology (IJSRET), ISSN 2278 – 0882 Volume 3, Issue 6, September 2014

[15] M. Srinath, Artificial intelligent Homes: The Robot Vacuum cleaner, August, 2017.

[16] R. S. Sutton, TD Modeling the world at a Mixture of Time Scales, Machine-Learning-Proceeding, p.532, 1995

[17] H. Kimura, M. Yamamura and S. Kobayashi, Reinforcement Learning by Stochastic Hill Climbing on Discounted Reward, Yokohama, Japan, p. 295, 1995

[18] P. Cichosz and J. J. Mulawka, Fast and Efficient Reinforcement Learning with Truncated Temporal Differences, Machine-Learning-Proceeding, p.99, 1995.