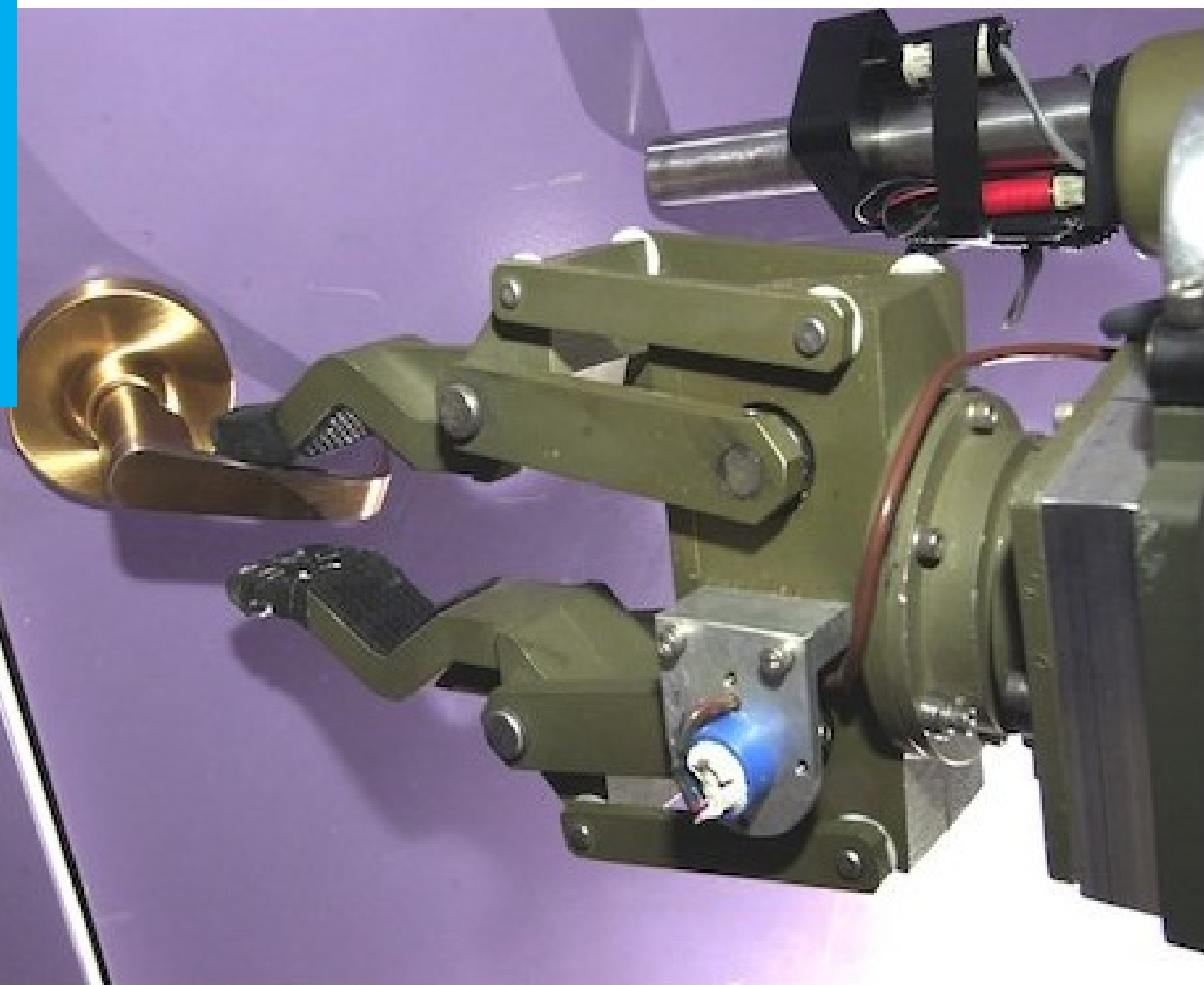


Learning, Transfer and Use of Affordances in Robotics Tasks

Osman A. Ciftci

Master of Science Thesis



Learning, Transfer and Use of Affordances in Robotics Tasks

MASTER OF SCIENCE THESIS

For the degree of Master of Science in Systems and Control at Delft
University of Technology

Osman A. Ciftci

February 12, 2014

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of
Technology

DELFT UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF
DELFT CENTER FOR SYSTEMS AND CONTROL (DCSC)

The undersigned hereby certify that they have read and recommend to the Faculty of
Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis
entitled

LEARNING, TRANSFER AND USE OF
AFFORDANCES IN ROBOTICS TASKS

by

OSMAN A. CIFTCI

in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE SYSTEMS AND CONTROL

Dated: February 12, 2014

Supervisor(s):

Prof. Dr. Robert Babuska

Asst. Prof. Gabriel Lopes

MSc. Chang Wang

Reader(s):

Prof.Dr. Robert Babuska

Asst. Prof. Gabriel Lopes

Asst. Prof. Koen Hindriks

MSc. Chang Wang

Abstract

Utilization of robotics in various applications increase with technological improvements. Few recent examples are Roomba vacuum cleaner robot for homes, industrial robots for production lines and unmanned robots for dangerous environments, such as NASA Mars Rover. One important aspect of robotics is the intelligence mechanism. However it is not always easy to design controllers and decision makers since mechanical designs of robots are becoming more complex as well as robotic tasks. Furthermore robots are expected to be able to cope with unforeseen interaction in these environments. Therefore a different approach than hard-coding the robotic agents beforehand should be used, such as learning.

The learning approach has its own difficulties, such as long learning sessions and not being able to transfer knowledge to new tasks efficiently to avoid performing another learning session. Usage of affordance concept is a solution for these difficulties since it maintains some kind of transition model of the environment and the robot. Affordance concept in combination with action planning enables to use the knowledge gained in previous tasks, environments or robots. This reduces or completely eliminates the learning time required for the new task, environment or robot.

This methodology also enables agents to successfully perform tasks in environments which have known properties but are new to the agent without additional learning, which is not always the case with transfer learning for Reinforcement Learning (RL) [1]. By successfully it is meant that the agent completes the task in an optimal manner. For example in a navigation task this will be reaching the destination in shortest time or lowest amount of steps.

Knowledge transfer between robotic agents will ideally decrease the required learning time and trials of new robotic agents in an environment. This can be done via passing some of the learned knowledge from the robotic agents which are already functioning in that very same environment. An agent which hasn't learned anything about the environment will not even know the basic properties, such as a wall is not being pushable and a docking station affords charging. A common representation will be time saving

for all agents that will benefit from such information. Indeed this is the case with the results presented in this thesis work.

Table of Contents

Acknowledgements	v
1 Introduction	1
1-1 Affordance	2
1-2 Related Work	3
1-3 Research Objectives	3
1-4 Methodology	4
1-5 Outline	5
2 Preliminaries	7
2-1 Affordances	7
2-2 Learning Classifier Systems	10
2-2-1 Reinforcement Learning	11
2-2-2 What is a Learning Classifier System?	13
2-2-3 Anticipatory Learning Classifier System	14
2-2-4 Anticipatory Classifier Systems (ACS) in Action Planning Task in Robots	16
2-3 Transfer Learning	18
2-4 Summary	23
3 Proposed Algorithm	25
3-1 Affordance Representation	25
3-2 Action Planning	32
3-3 Learning	33
3-3-1 Off-line Learning	33
3-3-2 On-line Learning	35

3-3-3	Hybrid Learning	37
3-3-4	Rule Generation	38
3-4	How Does Algorithm Handle Knowledge Transfer	38
3-4-1	Changes in the state space	39
3-4-2	Changes to the transition model	40
3-5	Suggestion for Knowledge Transfer Between Different Morphologies	40
3-6	Algorithm Used for Testing	41
3-6-1	Task Description	41
3-6-2	Grid-world	41
3-6-3	Learning	43
3-7	Summary	50
4	Results and Comparison	51
4-1	Off-line Learning	51
4-2	Transfer Learning in RL	54
4-2-1	Q-learning Agent (QA)	54
4-2-2	Without any Transfer Learning When Final States Change	54
4-2-3	With Transfer Learning When Final State is Changed	57
4-2-4	With Transfer Learning When the Room Design is Changed	58
4-3	Comparison of Off-line Learning Algorithm and Transfer Learning for Q-learning	60
4-4	On-line Learning	61
4-4-1	Initial Test	61
4-4-2	Advanced Test	64
4-4-3	Test Where the Capabilities of the Agent Have Been Changed	67
4-5	Summary	69
5	Conclusion and Future Work	71
5-1	Conclusion	71
5-2	Future Work	72
	Bibliography	73
	Glossary	75
	List of Acronyms	75
	List of Symbols	75

Acknowledgements

I would like to thank my supervisors, Prof. Dr. Robert Babuska, Asst. Prof. Gabriel Delgado Lopes and MSc. Chang Wang for their assistance during the writing of this thesis.

I cannot end the acknowledgements part without expressing my gratitude to my family and friends for their endless support.

Delft, University of Technology
February 12, 2014

Osman A. Ciftci

“I can only improve”

— *Osman Ali Ciftci*

Chapter 1

Introduction

Robots are getting more mechanically capable than ever before. This enables robots to be used in more difficult tasks. However it also creates a more complex problem of endowing robots with an intelligence mechanism that allows them to fulfil these difficult tasks. Two common ways to bestow intelligence to robots exists; the complete behaviour can be hard-coded to the robot or the robot can be programmed to learn the behaviour by itself. These behaviours should result in fulfilment of these tasks.

It is difficult to foresee every possible entity, problem or situation a robot can face while performing a complex or novel task. Therefore it is not feasible to hard-code a robot for every situation that can occur. In order to achieve complete autonomous behaviour in these tasks learning is required.

Learning process for complex tasks generally takes a lot of time and trial. For instance with Reinforcement Learning (RL), it usually takes a lot of time to learn how to complete a task. Furthermore if the task is changed RL agent has to execute another learning session for the new task. This learning can either be performed from scratch or by using transfer learning. Transfer learning can increase the speed of learning however this cannot be guaranteed [1].

The reason transfer learning for RL is not the best candidate for knowledge transfer is that the back bone of RL algorithm is the reward function that is defined according to a specific task, an environment and a robot. RL agents learn a policy that will maximize the reward they will get. When task, environment or robot is changed the policy obtained would most likely not be optimal for the new task, environment or robot. Therefore some more learning is required to achieve optimal behaviour.

In this study the aim is to come up with a methodology to decrease the amount of time and trials needed for robots to learn. This is possible via effective transferring knowledge between tasks and robots. In order to transfer knowledge between tasks and robots a common representation is needed. This representation can be seen as a common knowledge every human knows, such as a wall is not pushable by humans therefore no

one tries to push the walls. Even though walls of a Dutch house are different than an American house or people are different this example holds for everyone, everywhere and most of the time.

Affordance concept provides the required representation. In this thesis learning, using and transferring affordance knowledge are discussed. Furthermore a methodology is suggested for transferring affordance knowledge between tasks and robots.

1-1 Affordance

Affordance is a description of all action possibilities that are available to an individual [2]. Affordances are subjective therefore they are dependent on the capabilities of the individual [2]. Furthermore affordances are measurable [2]. For instance if a ball is liftable the displacement of the ball in Z-axis can be measured. However affordances are hidden in the environment [2]; there is no explicit indicator found in the environment about these affordances.

For instance in order to be able to open the door, handle should be twisted then pushed. Usage of the affordance knowledge here is that, the entity that is twisting the door handle knows that it is twistable, the door is pushable when the handle is twisted and the door frame is pass-throughable when the door is pushed wide open.

The term affordance has been accepted and interpreted differently by many hence there are so many different interpretations and frameworks [3]. Norman proposed that the word affordance means the design aspect of an object which suggests how the object should be used [4]. Most of the affordance definitions are not mathematical. In order to be able to use affordances in robotic applications a formalism is needed. In this study formalism of Sahin et al. [3] will be used. In this formalism, an affordance is defined by the following tuple:

$$(effect, (entity, behavior))$$

The ‘Entity’ is the thing the agent is interacting with. ‘Behaviour’ is the the action or set of actions the agent performs while interacting with this entity. The ‘Effect’ is the outcome of the behaviour performed while interacting with this entity.

This representation is general enough that it allows knowledge transfer between tasks for the same robot, which is shown in this study. In this representation transition model of the environment and the robot can be found hidden. This is a higher level knowledge than RL. In RL the knowledge is represented according to a reward function that is specifically adjusted for a specific task. Because of this, it is not always guaranteed that the knowledge transfer between RL tasks will increase the speed of learning and there is a possible that it can decrease the learning speed [1]. This is known as negative transfer.

It can be the case that the learned affordance knowledge may not hold any more, meaning that either the properties of the environment are changed or the properties

of the robot. However aim of this study to accelerate learning by using as much of the previously learned affordance knowledge as possible. In other words, if some of the affordance knowledge is not correct any more then it should be dismissed. The affordance knowledge that still holds should be used and not relearned.

Furthermore if the behaviour equivalences between robots can be found, this representation will also allow knowledge transfer between different robots. By behaviour equivalences it is meant that, two robots interacting with the same object using specific actions and causing the same effect. For example an industrial robot can push an object and a lower body humanoid robot can push the very same object using one of its legs. The object that is interacted with and the resulting effect are same. Therefore according to formalism of Sahin et al. [3] it can be concluded that these two behaviours are equivalent and can be shown as follows:

$$\begin{aligned} & (Trolley\ pushed\ forward\ by\ 1m, (Trolley, Move\ arm\ forward)) \\ & (Trolley\ pushed\ forward\ by\ 1m, (Trolley, Kick)) \end{aligned}$$

Using this behaviour equivalence it can be concluded that ‘Move arm forward’ action of the industrial robot is equivalent to the ‘Kick’ action of the lower body humanoid robot. Next time the robotic arm knows that if the lower body humanoid robot can move an object with its ‘Kick’ action then the robotic arm can also move this object using ‘Move arm forward’ action.

1-2 Related Work

The concept affordances and the transfer learning fields have been out there for a while however a methodology to combine both hasn’t been encountered by the author. There are very few papers that address affordance transfer very briefly, for example work of Perkins and Solomon [5]. However none of these provide a methodology that can be applied. The reason for this might be that the concept affordances is still very controversial in the literature because Gibson [2], who coined the term affordances, was not clear about the concept [3].

1-3 Research Objectives

Aim of this thesis study is to improve task performance of robots by using affordance learning, knowledge transfer and using this transferred knowledge effectively. By improved task performance it is meant decrease in learning time, therefore reduced number of trials. Reduced number of trials also means reducing costs in applications where trials are expensive.

The key questions of this study are as follows:

- How is affordance represented, learned and used in the literature? It is important to see what has been done in the literature, to take some of the work done in the literature as a starting point.
- How can affordance be learned in off-line and on-line task settings? In this study robot learning is preferred over hard-coding the knowledge into the robots. Therefore learning affordances should be accomplished.
- How can affordance knowledge be transferred between tasks? Using the affordance knowledge gained while doing a task would decrease learning time for another task since this will eliminate relearning some properties of the environment. Traditional RL problems are not very suitable to tackle with changing tasks since it is not guaranteed that the transfer will increase the speed of learning [1].
- How can affordance knowledge be transferred between different robotic agents? Ideally training a couple of agents in an environment should give enough knowledge that the rest of the agents will be able to learn everything by transfer but not trial. This will be beneficial in a scenario where trials are expensive.

1-4 Methodology

In this study, knowledge transfer between tasks and environments is realized by using a representation that holds estimated transition models of the environment and the robot and action planning. Because a more general knowledge is captured within the representation using this knowledge effectively in novel environments is possible. By effective usage it is meant that, the agent will complete a task in an near optimal manner with minimum amount of learning. With optimal manner it is meant reaching to the desired final state in shortest amount of time. This is difficult to achieve with transfer learning for RL, as mentioned earlier in this chapter.

Knowledge transfer would be time saving for training robots with different specifications rather than training them all from scratch. For instance in a navigation task where two mobile robots of different sizes which are not capable of pushing an object, it will be time saving if only one of them experiments with the object and transfers the knowledge to the other one. This way less time will be consumed and less trials will be performed. This is only possible for objects which have same affordance properties for these two robots, in other words the affordance knowledge will be consistent between two robots.

There is going to be some knowledge that will not be consistent between two different robots. In this case knowledge transfer would not have much benefit and should not increase learning time much compared to learning without transfer learning. In other words negative transfer should be as little as possible. Negative transfer means the effect of knowledge transfer that slows the learning rather than increasing the learning speed. For instance in a scenario where a small mobile robot is not able to push a box whereas a big mobile robot can. In this scenario the big robot transfers its affordance knowledge to the small robot. If the smaller robot thinks it can push the box because

of this transfer, it should quickly learn that this knowledge is not applicable for it and start learning about this box from scratch.

In order to transfer knowledge there should be 2 assumptions to be made. First a mapping between the sensory inputs of the robots should be known. In other words the robot that receives the knowledge should understand whether it is in a situation which is suggested by the affordance knowledge that was obtained by another robot. For instance, consider a rule that suggest when the robot's battery level is below 100%, if the robot faces a docking/charging station and it uses "Plug in" action then it's battery level will increase. If the robot is in this situation than it should recognize that.

Second assumption is that the robot should know a mapping between the actions of the robots should be known. In other words the "behaviour equivalences" between two robots should be known. Behaviour equivalences are mentioned in the section 1-1 briefly. These two mappings can be obtained by running experiments with these two agents in a common environment. As a results these two agents will have common references to interpret rules.

The current representations of RL do not allow knowledge transfer since the representation is very specific to one agent. A modified version of the formalism of Sahin et. al. [3] allows usage of a representation that is general enough that enables knowledge transfer between agents. This modified formalism, a tuple, is as follows:

$$(robot_id, (effect, (entity, behavior)))$$

The added 'Robot ID' will enable the agent, which receives this affordance knowledge, know which agent learned this knowledge. Therefore the agent that receives this knowledge can use the mappings between sensory inputs and actions of itself and the other agent.

When two assumptions mentioned above are realized then the knowledge transfer can be realized as follows:

Affordance knowledge transfer:

- 1: Robot #1 learns the affordance knowledge of the environment
- 2: Robot #2 gets the affordance knowledge learned by robot #1
- 3: Robot #2 interacts with objects and the environment
- 4: The knowledge gained by robot #1 that also holds for robot #2 is approved and put into the knowledge of robot #2

This way affordance learning of robot #2 is accelerated since it only requires small amount of trials to see if knowledge from the other robot holds for itself. Otherwise it would have to learn everything from scratch. If there are more than 2 robots with different specifications this might save time and cost.

1-5 Outline

This study consist of five chapters. In Chapter 2 some important concepts are explained. These concepts are affordances, Learning Classifier System (LCS)s and trans-

fer learning. Chapter 3 describes the proposed algorithm, in which affordance learning, transferring and using are mentioned. In addition to that the algorithm used for the experiments is explained.

Chapter 4 exhibits the results of how several agents perform in goal oriented tasks using the proposed algorithms and compare these results to basic Q-learning algorithm and transfer learning. Chapter 5 concludes this study with discussion points and comments for future studies.

Chapter 2

Preliminaries

Before going into further detail in this study some background information will be given in order to make concepts described in this study easier to understand. In this Chapter three topics will be mentioned. These are: Affordances, Learning Classifier System (LCS) (Anticipatory Classifier Systems (ACS) will be focus) and Transfer Learning.

Affordances are the starting point to this thesis work. Therefore they are very important. ACS is a decision making mechanism that allows the usage of affordance in tasks by agents. Transfer learning is not directly related to this thesis work; however it is used as a benchmark to compare the results of this work.

2-1 Affordances

Affordance is a concept of how to interact with objects in an environment. For instance in order to be able to open the door, handle should be twisted then pushed. The term affordance has been accepted and interpreted differently by many hence there are so many different interpretations and frameworks. Sahin et al. [3] explained that this is because Gibson [2], who coined the term affordances, was not clear about the concept. Therefore in this section only selected few of the affordance descriptions and formalisms are described in this Chapter.

According to Norman the word affordance means the design aspect of an object which suggests how the object should be used [4]. The entity which uses these objects is called organism, actor, human and animal interchangeably in the literature. However they all represent the entity which interacts with objects in different environments. McGrenere and Ho [6] state that even though affordance concept was popularized by Norman in Human-Computer Interface (HCI) Community it was Gibson [2] who first coined the word affordance.

Gibson defined affordances as follows “...the affordances of the environment are what it offers the animal, what it provides or furnishes, either for good or ill.” [2].

Three important properties of affordances are mentioned by McGrenere and Ho [6]. These are:

- An affordance exists relative to the action capabilities of a particular actor
- The existence of an affordance is independent of the actor’s ability to perceive it.
- An affordance does not change as the needs and goals of the actor change.

In order to make these properties easier to understand some examples will follow. First property states that an affordance depends on capabilities of actor. For instance a box can offer pushability. Although a human can easily push the box, a sparrow might not have the strength.

Another example is a surface that affords support. This surface might provide support for one actor but may not for another because of different weights and shapes of actors [2]. In the former case affordance of support exists however it does not in the latter case.

Second and third properties basically state affordances are invariant of intentions and awareness of the actor. A drunk person can knock over a vase without knowing that the vase was there. In this case the vase offers knock-overability affordance. If objects do not have an intelligence mechanism to understand and act according to intentions of the actor they will exhibit same behaviour.

An ordinary floor will act the same regardless of the intentions of the actor. However an anti-vehicle landmine might be equipped with a Friend-Foe Identification system. This landmine will act different according to different vehicles. This example is only valid if one assumes Friend-Foe Identification system exists and works properly.

An example of a world where affordances are not invariant of intentions and awareness of the actors could be, people walking on water or walking through walls. In other words people would be able to manipulate affordances of objects, entities etc. with their intentions.

Gibson thought that an actor does not have to classify the object in order to understand what they afford: “The theory of affordances rescues us from the philosophical muddle of assuming fixed classes of objects, each defined by its common features and then given a name. You do not have to classify and label things in order to perceive what they afford.”[2]. Gibson’s claim was actually proved by work of Humphreys [7]. Humphreys experimented with a subject who was often not able to name the objects. However the subject was able to gesture how the objects should be used.

Gibson’s affordance has been interpreted differently by many. Quoting Gibson reveals why there are so many different interpretations of this concept:“...an affordance is

neither an objective property nor a subjective property; or both if you like. An affordance cuts across the dichotomy of subjective-objective and helps us to understand its inadequacy. It is equally a fact of the environment and a fact of behaviour. It is both physical and psychical, yet neither. An affordance points both ways, to the environment and to the observer.”[2].

Sahin et al. believes that there are multiple reason why there are more than one interpretations of affordances [3]. Reasons are Gibson’s own understanding of affordances changed in time, his concept was not finalized, he did research on perception aspect of affordances only but he did not go into action and learning aspects and there are few more reasons.

In order to clarify the concept, McGrenere and Ho argue that Gibson implies two properties but never directly states them [6]. First one is affordances being binary, for example a stair is either climbable or not. This binary definition does not address the gray in between. For the stair example it is a stair which is climbable but with great difficulty. Making affordances fuzzy might be an interesting extension.

Second property Gibson implies is that affordances can be nested when an action possibility is composed of one or more action possibilities [6]. Here is an example: “... an apple affords eating but eating is composed of biting, chewing and swallowing, all of which are afforded by the apple.”[6].

Since there are many different interpretations and controversies in the field of affordance a formalism has been selected as a starting point. This can be found in the following Section.

A Formalism for Affordances

Since the affordance concept is still vague and controversial a formalism was selected as a starting point to this thesis study. This formalism is from Sahin et al. [3]. They proposed a 3-tuple formalism to describe affordances, which is:

$$(effect, (entity, behaviour))$$

Entity denotes sensory information, behaviour denotes agent’s perception-action mechanism and effect speaks for itself. This tuple is very similar to the ACS, which is mentioned in this Chapter, since ACS has Condition-Action-Effect representation. Similarities are between condition and entity, action and behaviour and finally both of the effects. Therefore using ACS in affordance applications might prove useful.

Following example explains formalism of Sahin et al., “...the lift-ability affordance implicitly assumes that, when the lift behaviour is applied to a stone, it produces the effect lifted, meaning that the stone’s position, as perceived by the agent, is elevated.”. This formalism also suggests that entity and behaviour is a pair which can potentially generate a certain effect. Effect part of the affordance can be used as an index for scenarios where certain effects are wanted. In other words appropriate entity-behaviour pair can be selected to serve a purpose.

Sahin et al. [3] has mentioned the affordance acquisition in their formalism. The robot uses its repertoire of behaviour and records the effects as relation instances such as:

(lifted, (black-can, lift-with-right-hand))

This tuple says that black-can has been lifted with the right hand. They however emphasize that these are not relations as they do not have any predictive ability over future experiments including novel objects. Therefore they propose a method in which relation instances can be bound together toward discovering affordances of these objects. According to concept of equivalences, elements of the tuple can become more general. For example:

(lifted, (black-can, lift-with-right-hand))

(lifted, (blue-can, lift-with-right-hand))

These two 3-tuples can be represented as:

(lifted, (<*-can>, lift-with-right-hand))

Where <*-can> denotes invariant of the entity equivalence class. In this case <*-can> represents cans of every color. This equivalence class can be used to generalize and enable the agent to predict what is going to happen when it tries to interact with a novel object, for instance lifting a green-can.

Apart from entity equivalences, there also exists behaviour, affordance and effect equivalences. In the formalism of Sahin et al., an observers idea of affordance watching another agent behaving in a certain way can also be represented. It can be shown as follows:

(<effect>,(<agent>,<(entity, behaviour)>))

They believe this representation can help learning species concept. Observer can also watch an agent of the same class to learn affordances without performing the action, without trial. This is just like mirror neuron activity in some animals. These neurons fire when individual performs the action or watches someone else does it. However this representation is only valid if an agent is dealing with an inanimate object. For instance, if there are two agents interacting with each other, this representation might not be sufficient.

2-2 Learning Classifier Systems

Idea behind LCS is that rather than having one single best fit, evolving a population of rules. This is well suited for complex robotic tasks since robotic tasks are generally

not linear and dynamic. These rules are also known as classifiers and they can be used for classification and modelling problems. Classification tasks require a decision maker to be able to correctly name the class of the input, for instance a four legged stable surface with a back support can be classified as a chair.

Modelling problems require a decision maker to predict what will be the future state of the system considering the present state of the system. A good model of a car is expected to estimate roughly what will happen if the throttle is opened at a certain rate. A well known LCS for modelling is ACS, which is explained in the following.

LCS is basically a population of classifiers with the ability to improve its performance due to its learning and discovery properties. The discovery property here means the ability to introducing new rules to the population; this is generally done by Genetic Algorithm (GA). Please note that there are other techniques of rule discovery as well, such as Neural Networks [8] or Anticipation Learning which is a very direct method used in ACS [9]. The aim of the discovery action is to find the rules which are more fit and offer better performance.

The learning property is required in order to judge how each rule performs. In literature learning is also called reinforcement or credit assignment as pointed out by Urbanowicz and Moore [10]. There are other ways of learning such as Supervised Learning, however these are not commonly used. Reinforcement learning which is a popular learning method used with LCS, will be introduced briefly in section 2-2-1.

LCS are good candidates for this study since LCSs are adaptive, easily customizable. LCSs also have generalization ability and they perform well in big-scaled environments. Furthermore ACS is more suitable for our study since it employs a representation which is very similar to the affordance concept.

In future work, GA will be used as the discovery tool and LCS will be applied to reinforcement learning problems therefore emphasis is on these topics rather than other various alternatives.

2-2-1 Reinforcement Learning

Reinforcement Learning (RL) gets its name from reinforcements, also known as rewards, given to the agent while learning; these rewards exist in the environment the agent acts in. The name ‘reward’ might be confusing since both positive rewards and negative rewards are included in the term ‘reward’.

An example of a reward is a cookie which can be given to a dog after it obeys a command, such as ‘sit’. The purpose of an RL agent is to find a mapping from its sensory input to its actions that will maximize the cumulative reward; this mapping is also called policy. In this example the dog is trying to get as many cookies as possible.

Grondman et al. [11] described an RL problem, which was in a deterministic environment, as a deterministic Markov Decision Process (MDP). In this study their description will be used. MDP is defined as:

$$M : M(X, U, f, \rho)$$

Which is also called a tuple in the literature. X is the state space, possible configurations of the environment perceived by the RL agent. U is the action space, every action possibility available to RL agent. The state transition function is shown as:

$$f : X \times U \rightarrow X$$

In the deterministic environments, it is known what will be the next state, x_k , of the system when an action, u_{k-1} , is executed at a specific state, x_{k-1} . Finally the reward function is defined as:

$$\rho : X \times U \rightarrow R$$

Reward function indicates what will be the reward the RL agent gets when a specific action, u_{k-1} , is selected in a specific state, x_{k-1} .

Policy was mentioned earlier briefly. In technical terms policy is:

$$\pi : X \rightarrow U$$

It is a function from current state, x_{k-1} , to selected action, u_{k-1} .

The RL agent first explores the environment it is acting in; then exploits the knowledge it learned to maximize the reward. There are number of strategies to act in an environment. For instance RL agent can fully exploit its knowledge in the expense of learning more information about the environment. This very agent can also exploit and explore consecutively in the expense of receiving maximum reward. If the environment is static, after some time further exploration will not be needed. However, if the system is dynamic some amount of exploration might prove to be useful as the optimal policy could change with the varying environment properties.

The policy is based on previous experiences. In this study Q-value is the chosen method to store previous experiences. Q-value described by Watkins and Dayan [12] as "... Q value is the expected discounted reward for executing action a at state x and following policy π thereafter". Mathematically to calculate the Q-value following formula is used:

$$Q^\pi(x, a) = R_x(a) + \gamma \sum_y P_{xy}[\pi(x)] V^\pi(y)$$

Please note that Watkins and Dayan [12] use a to represent action which was represented by u earlier in this section.

2-2-2 What is a Learning Classifier System?

LCSs have many variations; however there are four main components to a LCS [13]. These components are:

- 1) Population of classifiers
- 2) Performance component
- 3) Reinforcement component
- 4) Discovery component

These components are described in more detailed in the following. However since in this thesis study only ACS is used among all the LCSs the emphasis will be more on the ACS.

Population of classifiers

This population is the knowledge representation of the system perceived by the LCS. In the ACS this is represented by Condition-Action-Expectation tuples [9]:

$$C - A - E$$

It should be noted that this representation is similar to affordance formalism of Sahin et al. [3]. Affordance concept and this formalism are discussed previously in this Chapter. These tuples can be perceived as rules that suggest ‘If in condition C performing action A will yield the effect E’.

Performance Component

This component is responsible for interacting with the environment. In other words, this component manipulates the agent. For example, in a navigation task this component can be the action planning in case of ACS. For different LCSs and tasks there can be different performance components however they will not be mentioned in this study.

C-A-E representation of knowledge enables ACS to do action planning by taking advantage of the expectations of the future. Action planning is explained briefly in Chapter 3.

Reinforcement Component

In order to determine how good the classifier is there are classifier performance measures found in classifiers. Reinforcement component is responsible of validating the classifiers performance. In other words if a classifier is not correct or not performing well then this component either deletes this classifier from the population or decreases the trust in this classifier. The exact opposite also hold. If a classifier is creating wonder for the tasks then its performance measure is increased by the Reinforcement Component.

Discovery Component

Discovery component governs discovering new rules that hopefully perform better than the available ones. It is common to see GA as the discovery component for LCS. However no further details will be given in this study about GA since the method used in this study is explained in Chapter 3.

2-2-3 Anticipatory Learning Classifier System

The representation of ACS is mentioned earlier as:

$$C - A - E$$

In ACS perceived results, R , from the environment are compared with the expectation, E [9]. By doing so ACS can learn without environmental rewards like most other LCSs need. Expectation occur before the results. Therefore comparisons are always done after predicting an expectation and performing an action according to the expectation.

ACS unlike most of LCS can learn without any rewards. It doesn't need rewards to create a model of the environment. Please note that this model is based on the expectation of the ACS and they are not always correct or accurate.

Learning without rewards idea can be supported by an experiment which Stolzman [14] reports. This experiment [15] has been conducted with rats. Seward [15] claimed that rats can learn without rewards. In his experiment rats started exploring a simple T-maze from the lower part of the T without getting any rewards. However each end of the T-maze were in distinguishable colours therefore rats were able to learn the maze.

After the training, rats were placed in one of the distinguishable ends of the T-maze where they received food for one minute. Later, rats were placed somewhere else in the maze and 28 out of 32 rats went directly to the correct end of the T-maze where they received food before. This shows that most of the rats learned this simple maze without receiving rewards. One can argue that food was the reward. However food was used to direct the rats to a location since it is not possible to communicate with the rats in another way.

SRE formation which was proposed by Edward Tolman [16] indicates that in certain situation S , reaction R of the agent lead to the effect E . However according to Stolzmann [9] it was Hoffmann [17] who formulated a learning mechanism which is called anticipatory behavioural control; which can be seen in Figure 2-1. No further detail about Tolman and Hoffmann's work will be given here.

Figure 2-2 shows the basic schema of ACS. Just like any other LCS algorithm, detectors of ACS receive the environmental state S_t and it is recorded in the message list. Message list can be seen as a memory of the past. S_t is compared with the population of rules and match set is formed. Again there can be many action selection possibilities. However Stolzmann mentions only one, which is a cooperation of roulette wheel selection and exploration where explorations refers to random selection [9].

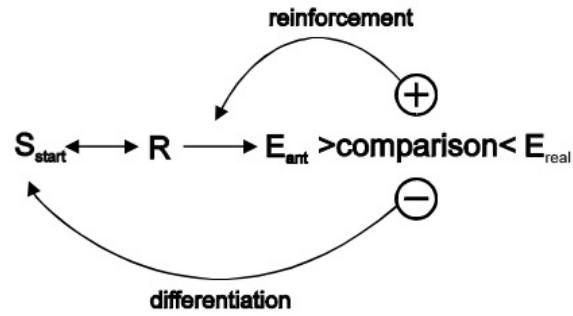


Figure 2-1: Anticipatory behavioral control [9]

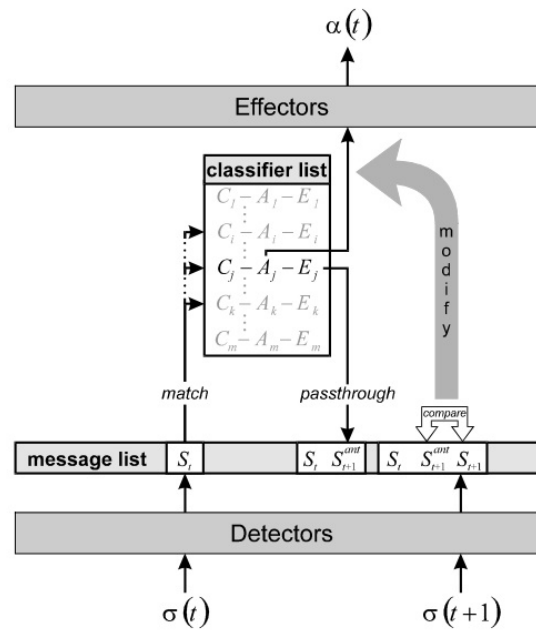


Figure 2-2: ACS Algorithm in one cycle [9]

After action selection ACS starts to differ from other LCSs; expectations are recorded to the message list as S_{t+1}^{ant} and suggested action by the classifier is executed. After performing the suggested action new states are perceived by the detectors and saved to the message list as S_{t+1} . S_{t+1} and S_t are then compared to see how the classifier performed predicting the outcome of the action. Depending on the results, classifier is updated via a process called ‘Anticipation Learning’ [9]. If S_{t+1} was anticipated correctly by the classifier, i.e. if S_{t+1}^{ant} is the same as S_{t+1} quality of the classifier is increased; if not new classifier should be generated or quality of the classifier should be decreased.

Please note that Stolzmann [9] did not mention anything about genetic algorithm to come up with new classifiers or get rid of the old ones, instead he mentioned 4 cases which lead to modifications to the population of classifiers. These cases are: useless case, expected case, correctable case and not correctable case, which refer to the classifier. In ‘expected case’, S_{t+1}^{ant} and S_{t+1} matches and t changes the environment, in other words when classifier changes state of the environment and predicts the results accurately the quality of the classifier is increased.

In the ‘useless case’, it does not matter whether S_{t+1}^{ant} and S_{t+1} matches or not, only concern is “Did action at time instance t caused any changes in the environment?” if not then the quality of the classifier is reduced. However this may be misleading in environments where stabilization of the system is a concern. For instance it might be preferred to have classifiers which hold the system in unstable equilibrium points, such as stabilizing an inverted pendulum at upright position. But because LCS are customizable easily this will not be a big problem in different applications.

‘Correctable case’ is where the classifier predicts effects of the action correctly up to a point, for example if S_{t+1}^{ant} is 11#1 and S_{t+1} is 1101, then this is a ‘correctable case’. In ‘correctable case’ a new classifier is created where the condition is S_t and the expectation is S_{t+1} . Stolzmann [9] did not mention what to do with the old classifier directly after creating another one based on it. However he mentioned that if a classifiers quality is reduced below a threshold value it is removed from the population. Fourth and the last case is ‘not correctable case’ where S_{t+1}^{ant} and S_{t+1} does not match and it is not possible to correct it as in ‘correctable case’, in this case quality of the classifier is reduced.

Stolzmann [9] also issued reward learning with the ACS, it is also mentioned as pay-off prediction in the literature. He said “Anticipation-learning enables an ACS to learn without environmental rewards. However, if environmental rewards are given, then an ACS can be treated like a usual classifier system that uses a reinforcement learning technique.” and he described bucket brigade algorithm for cases where environmental rewards are available.

2-2-4 ACS in Action Planning Task in Robots

Stolzmann and Butz [14] used ACS in two experiments. First experiment results showed that ACS could learn without environmental rewards and it could do action planning. Second experiment showed goal-directed learning increases learning capabilities of ACS.

























		Position in the maze					
		S	1	2	3	F	E
Orientation	N						
		10000	01110	00110	01100	10001	10000
	W						
		00010	11100	01100	11000	00011	00010
	S						
		00100	11010	11000	10010	00101	00100
	E						
		01000	10110	10010	00110	01001	01000

Figure 2-3: Representation of the T-maze in ACS [14]

For their first experiment they did not use environmental rewards, this type of learning is called latent learning. Because reward is always zero for all time instances reinforcement learning techniques cannot be applied [14]. Stolzmann and Butz used a Khepera robot which is shown in the Figure 2-5. Khepera robot is a mini mobile robot, has 2 independent DC motors with encoders and 8 infra-red sensors (emitter/receiver). For their experiment they put an infra-red light in the goal-box so the robot can distinguish end of the T-maze; T-maze can be seen in the Figure 2-4. By using predefined situations and actions, such as ‘there is something in front of the robot’ or ‘there is nothing on the left of the robot’ and ‘make a 90 degree left turn’ or ‘go forward’, they initiated a simulated learning because it would take a long time with the real robot. The situations defined by Stolzmann and Butz can be seen in the Figure 2-3

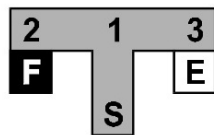


Figure 2-4: T-maze which was used in Stolzmann and Butz work [14]

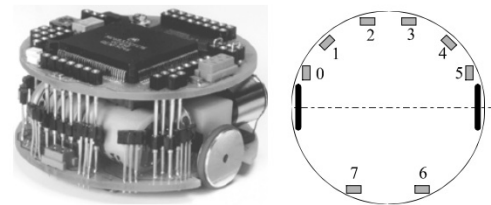


Figure 2-5: Khepera Robot [14]

Expectations of the reliable classifiers are taken into account when action planning since. This process can be seen as analogous to Model Predictive Control in Control Engineering domain. Stolzmann and Butz [14] came up with two ways of action planning. One of them is ‘forwards-directed breadth-first search’ where ACS tries to find consecutive reliable classifiers which will lead to the desired state from the initial state. ‘Forwards-directed breadth-first search’ can be done by assuming the estimates of the classifier will become true. Therefore it is already known which classifier will be active in the next time step. This might sound wrong and in most cases it is. However since this action planning is repeated at each time step, the false assumption of ‘estimations of the classifiers will always become true’ does not harm the action planning process.

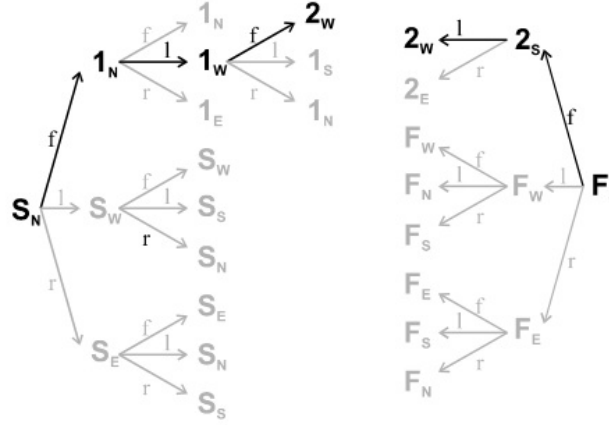


Figure 2-6: ACS doing action planning using bi-directional search method [14]

Action planning is done until a chain of classifiers are found that will lead to the desired state, or position in case of a mobile robotic task.

Other action planning method is ‘backwards-directed breadth-first search’ where consecutive chain of classifiers is tried to be find which starts from the initial state and arrives to the destination state. In Figure 2-6 a combination of ‘forwards-directed breadth-first search’ and ‘backwards-directed breadth-first search’ is illustrated; this Figure gives an idea how these action planning methods work.

T-maze problem can be seen in the Figure 2-4. The task is not as simple as it looks as Stolzmann and Butz notes “The experiment was replicated with students: The states were presented as shown in figure 3 [in this study it corresponds to Figure 2-3] and the students were told to press three different keys [three available actions] to explore this finite state world for 15 minutes. After that only two of 11 students knew how to reach one of the goal-boxes”.

The second experiment of Stolzman and Butz [14] will not mentioned in this study since it is out of the scope.

2-3 Transfer Learning

One method for machine learning is RL, which has already been briefly described in this Chapter. RL can help agents learn a policy in environments where the agent doesn’t receive steady rewards. This is beneficial in scenarios where it is easy to just specify the desired state of the system and assigning a reward to that state. An example can be a chess game where the agent receives only rewards when the game ends. However one problem with RL is that it is slow and infeasible when the agent is acting in complex environments [1].

Taylor and Stone [1] mention there are several methods to accelerate learning rate for the RL in literature, such as hierarchy of subtasks, learning with higher level actions

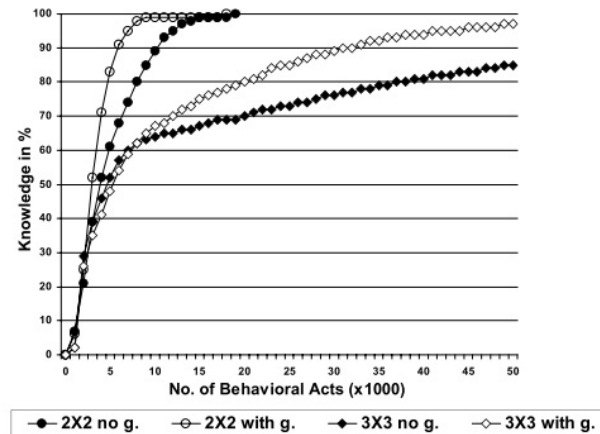


Figure 2-7: Results of the 'no g.' (non goal-directed) and the 'with g.' (with goal-directed) learning [14]

and using function approximation. However their survey is about the relatively new approach to the RL, which is transfer learning.

The idea behind transfer learning is that "...experience gained in learning to perform one task can help improve learning performance in a related, but different, task"[1]. The question is how different these tasks can be.

In their survey Taylor and Stone mention the changes and differences in tasks that allow the usage of transfer learning in MDP tasks [1]. These can be found in the following bulletin; some of these are explained briefly and some of them are already self-explanatory:

- Transition function (denoted by t)
- State space (s)
- Start state (s_i)
- Goal state (s_f)
- State variables (v), state representation of the agent in physical tasks this will correspond to combination of sensors
- Reward function (r)
- Action set (a)
- Problem state (p), how the agent uses its sensory information
- Some objects ($\#$), number of objects

In order to get a better grasp of these possible changes authors have given an example of a mountain car task. In this example the changes can be:

- t : using a different engine for the car or different surface type
- s : increasing the range of state variables
- s_i : different starting position
- s_f : different destination
- v : representing the agent with only its velocity
- r : a different reward function
- a : disabling an action which was previously available, for instance ‘Neutral’ action
- p : agent can add previous speed values of itself into the state representation
- $\#$: the agent may control more than one cars

With transfer learning these changes between tasks can be tackled. This is done by transferring some kind of knowledge. Some of the transferred knowledge which are mentioned by Taylor and Stone [1] are as follows:

- Low level transfers, such as (s, a, r, s') instances
- Action-value function (Q values)
- A policy
- Rules

Not all of the transferred knowledge is mentioned here since it is not in the scope of this study. The whole purpose of introducing transfer learning here is just to create a common ground with this thesis work and transfer learning domain. Therefore the results of this study will become comparable to the results of transfer learning domain.

Some of the transfer methods assume the same state variables and actions are used by the source and target tasks [1]. The source task is the task in which the prior knowledge is gained and the target task is the task in which this prior knowledge is used to complete the task.

When the same state variables and actions are used there is no need for an explicit mapping between tasks. However this is not the case for every knowledge transfer. An example for this situation can be a task where the agent performing in this task has x and y positions as states, in this order. If this agent transfer knowledge to another agent which has its states as y and x positions, in this order, then the knowledge will not be consistent when transferred between two tasks. In this case an explicit mapping is needed. A visual representation of this mapping can be seen in the Figure 2-8.

Further detail about transfer learning will not be given in this study since transfer learning is just issued here to have a common ground with this domain. Therefore the

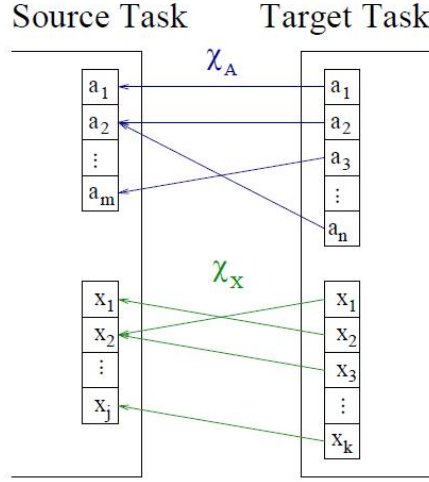


Figure 2-8: Visualisation of an explicit mapping to relate the knowledge from one task to another [1]

results of this study will become comparable to any study in the domain of transfer learning.

There are some transfer measures that measure the performance of the transfer learning compared with a learning without any transfer learning. In other words, this measure reveals if it is a good idea to perform transfer learning or just wipe the previous knowledge and start learning from scratch. In their study Taylor and Stone [1] mentioned five transfer measures. These are:

- Jumpstart: This answers the question of “Can transfer be used so that the initial performance is increased relative to the performance of an initial (random) policy”[1]
- Asymptotic: This measure compares the performance when the learning converges between the target tasks with and without learning. However it is not easy to tell when the learning agent converges.
- Total reward: This measure sums up the performance values of the target task for each trial. For RL this would be summing up the accumulated reward from each trial.
- Ratio of areas under the performance curves: This measure is depicted as follows:

$$\frac{A_t - A_{nt}}{A_{nt}}$$

Where ‘ A_t ’ is the area under curve with transfer and ‘ A_{nt} ’ is the area under curve without transfer.

- Time to threshold: This measure reveals which agent, agent with transfer learning or agent without transfer learning, is the first to reach a certain level of performance value. In the case of RL performance value can be the total amount of rewards gained by the agent after a trial if the RL agent is being punished for taking a long time to complete a task. In another setting this value can be reaching the desired state in some amount of steps.

In their study Taylor and Stone did not assign names for each of the measures. For easier usage of these term in this study names have been assigned. The visualisation of above mentioned performance measures can be seen in the Figure 2-9.

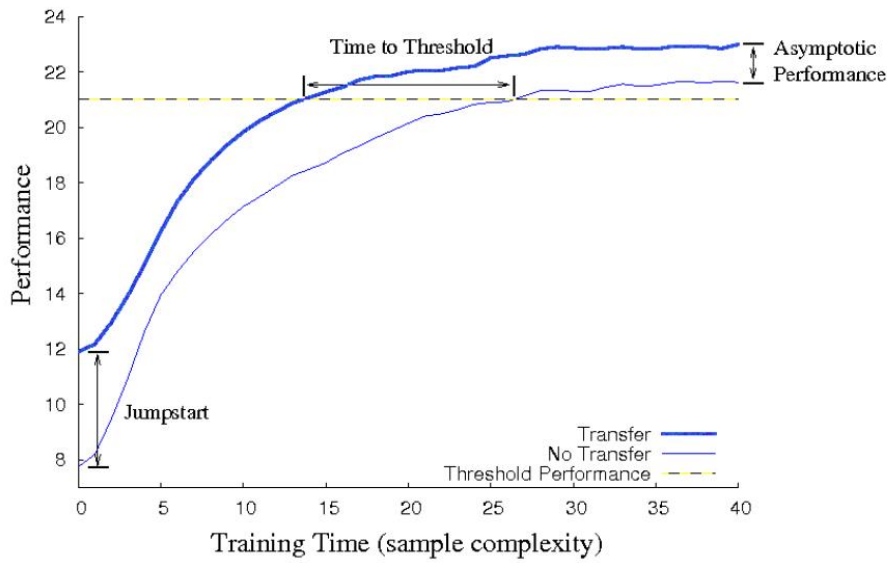


Figure 2-9: Visualisation of performance measures for transfer learning [1]

With these transfer measures effectiveness of transfer learning can be measured. This is important since there can also be cases where the transfer learning actually makes target task learning worse. In other words, the learning agent that receives the knowledge might perform worse than the case where it doesn't receive any knowledge. This means negative transfer happens [18].

Negative transfer happens when the source and the target tasks are not a good match [18]. And as Taylor and Stone states, none of the transfer learning algorithms for RL can guarantee that this negative transfer will not happen [1]. Furthermore they say avoiding this is an open question.

The reason negative transfer happens is that the back bone of RL, rewards, are defined for specific tasks. Transferring this reward or a function of this reward (such as Q-values or the policy) is not going to be optimal for a different task. In a navigation scenario if an RL agent learn a Q-table that enables it to reach a final state and the new state is a very different then this Q-table will direct the RL agent to the old final state. The RL agent will take time to learn that this Q-table is not getting it to the new final state.

Furthermore if this time is bigger than just starting a new learning from scratch to get to this new final state then this means there is negative transfer between these tasks.

2-4 Summary

In this Chapter three important concepts for this study have been introduced; namely Affordance, ACS and Transfer Learning. Some of this information has been directly used to develop the algorithm of this thesis work (Affordances and ACS) and Transfer Learning is used to use a formalism to compare the results of the algorithm.

Proposed Algorithm

In this Chapter an algorithm for transferring affordance knowledge is proposed. This algorithm allows learning affordances in off-line and on-line manners, using this affordance knowledge for completing tasks and transferring this affordance knowledge between tasks, environments or robots. The difference between tasks and environments can be explained in the following example. The agent might be in the same environment and performing different tasks, such as arrive at destination A, arrive at destination B, put boxes together, charge its batteries and so on.

Before going into the algorithm the affordance concept will be mentioned briefly and the affordance formalism used in this study will be defined. There are examples to make the concepts defined in this Chapter easier to be understood. Then action planning will be mentioned briefly because it allows using the affordance knowledge to fulfil the tasks given to the agent.

Three variants of the algorithm will be described after the action planning. These variants are: Off-line, on-line and hybrid learning. Then the subject of how this algorithm handles knowledge transfer between tasks, environments and robots will be mentioned. Lastly the algorithms used for the tests of the methodology are explained.

3-1 Affordance Representation

In the Chapter 4 results of the experiments with transfer learning for Q-learning are shown and it is concluded that as it was mentioned in the previous Chapters that transfer learning for Q-learning is not suitable for transferring knowledge between tasks, environments and robots. The reason is that the representation of the knowledge is only for a specific task, a environment and a robot. It is possible to transfer knowledge between very similar tasks, environments and robots. However there is no guarantee that there will not be negative transfer [1], which is with the transfer learning the learning becomes worse than without any transfer learning.

A good idea to overcome problems of transfer learning can be teaching the robot the transition model of the environment and the robot. Please note that this transition model may not be correct. This will be the agents interpretation of the environment and itself. On top of that robot can benefit from this transition model to plan actions to achieve its goals. This way if the task is different the robot can use the previously learnt knowledge while performing another task, unlike transfer learning of Q-learning.

In order to be able to learn a transition model the affordance concept has been selected as starting point. The definition of the affordances can be found in Chapter 2

If the environment is changed in such a way that elements of the environment remain the same but the positions and orientations of the elements are changed, the affordance based transition model will be able to handle this change without further need of training whereas the transfer learning for Q-learning cannot. Furthermore if the capabilities of the robot are changed, with this transition model the robot will still be able to use some of the knowledge that are same for the current capabilities and the previous capabilities. However it will need some learning to get used to its differences with the previous capabilities.

Since the affordance concept is still very controversial and not very clear for mathematical usage a formalism has to be used. The formalism of Sahin et al. [3], which can be seen below, is a good starting point:

$$(effect, (entity, behavior))$$

This tuple consists of ‘entity’, ‘behaviour’ and ‘effect’. The ‘entity’ is the thing the agent is interacting with. The ‘behaviour’ is the action or series of actions the agent takes and the ‘effect’ is the result of these actions. The following are some examples to make the formalism better understandable:

$$\begin{aligned} &(lifted, (black - can, lift - with - right - hand)) \\ &(not - moved, (wall, push)) \\ &(dooropened, (button, push)) \end{aligned}$$

However it was seen that this formalism lacked some information in it, which is the state of the environment and the robot. For example in a scenario where a mobile robot is facing a door the following two scenarios can happen. First scenario: The door is slightly open, which does not allow the robot to pass-through; a visual of this scenario can be seen in the Figure 3-1. In the formalism of Sahin et al. it will look as follows:

$$(Remain stationary, (Door, Go Forward))$$

In the second scenario, the door is fully open, which does allow the robot to pass-through; a visual of this scenario can be seen in the Figure 3-2. In the formalism of Sahin et al. it will look as follows:

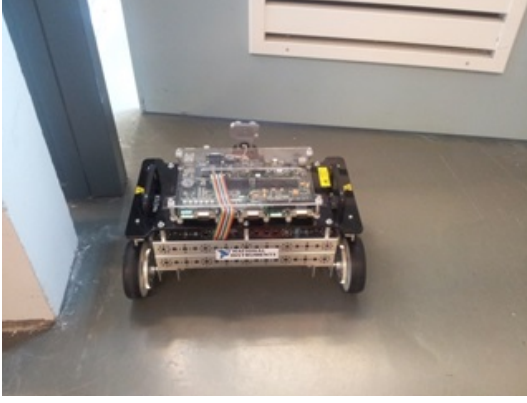


Figure 3-1: First scenario where the robot is not able to pass through the door



Figure 3-2: Second scenario where the robot is able to pass through the door

$$(Pass - through, (Door, Go Forward))$$

Because there is some information about environment is missing in the representation the knowledge will not be consistent. The next time the robot faces the door, it will have two experiences suggesting different results, even though when it takes the same action.

There is a reason why entities in the last example are selected as a door but not an obstacle, for the first scenario, and an empty space, for the second scenario. The reason is empty space, for instance, does not provide the information that state of the door might change. On the other hand, the entity door provides this information. Depending on the state of the door it can either behave like an empty space or an obstacle. Therefore, with this entity, when the agent faces a door and it is closed the agent will know that with a set of actions it can be opened.

In another scenario a mobile robot might have a robotic arm attached to it. Depending on the orientation of the end effector of the arm, the robot might get stuck going through a low passage or not. However with the formalism of Sahin et al. both of the tuples will look as follows:

$$\begin{aligned} &(Remain stationary, (Passage, Go Forward)) \\ &(Pass - through, (Passage, Go Forward)) \end{aligned}$$

Similar to the previous scenario there is some information is missing in the representation. Therefore the knowledge will not be consistent. This missing information is the states of the robot. This will be a problem when these tuples are used for completing tasks, such as in action planning, since the states that determine the effects are not considered. In order to tackle this problem a modified formalism has been used in this study which can be seen below:

$$(Entity, States, Action, Effect)$$

Where ‘Entity’ is the object, thing or entity the robot is interacting with. These entities should have labels which are generated by another mechanism which is not mentioned in this study. This mechanism should be able to group entities together that have very similar properties. For example, in the case of doors, the door of the bathroom, street door or a door of a hotel room behave the same, even though they might be in different sizes and properties, such as friction coefficients. In short robot should know that it is facing a door, in this example. This can be done via image processing or assigning bar-codes to the entities in an environment.

The ‘States’ describe the state the robot and the environment is in. States are represented as follows:

$$s(t) = [s_{env}(t), s_{rob}(t)]$$

Where:

$$s_{env}(t) = [s_{ent}(t), s_{other}(t)]$$

States should include as many observable properties of the environment as possible. And the agent should learn which states are affecting the outcome of actions without explicitly being told so. For example states might consist of these elements for an agent: Position of the agent, orientation of the agent, what is the openness of the door #1 and what is the openness of the door #2. Out of all these states the agent should figure out that when it is trying to go-through the door #1 the only important state to consider will be the openness of the door #1.

When the agent knows which states affect the outcome of its actions then it will have the transition model of the environment and itself. This will allow it to perform action planning to fulfil its tasks. There are two ways to show which states influence the effects. First way is replacing the state value with ‘Do not care’ symbol (#). An example can be seen below:

$$(Entity_a, (\#, \#, 5, \#), Action_b, Effect_c)$$

In the above example the agent is interacting with $Entity_a$, states are $(\#, \#, 5, \#)$, performs $Action_b$ and the result is $Effect_c$. In this example only the value of the third element of the states is important. Remaining the states are not relevant and they do not influence the effect.

Another way to show which states influence the effects is only showing the appropriate values of the influencing state. For example:

$$(Entity_a, State_3 = 5, Action_b, Effect_c)$$

In this study the first way will be used.

‘Action’ is the set of possible actions and behaviours, complex series of actions, the agent can perform. Behaviours can be programmed if the programmer knows some of the objects the agent will be interacting within an environment. For example, if a humanoid robot will be interacting with a fridge a series of actions to open the fridge door can be put in order as follows: Lift right hand, rotate the elbow 90°, move the hand forward until palm touches the handle of the door, grab the handle, pull. ‘Action’ can be represented as $a(t)$.

‘Effects’ are the differences in the states that are caused by the actions of the robot. The effect in the representation of Sahin et al. gets a label, such as Traversable, Lifiable, Pushable. These labels has to be extracted from the changes in the state space. In this study, however, this is not the case. Such labels are not needed to transfer knowledge between robots. ‘Effects’, in this study, are defined as follows:

$$e(t) : s(t) - s(t - 1)$$

Entity is actually one of the states of the robot, since it can be defined as the entity the robot is facing or interacting with. However it was defined separately since it is impossible to talk about affordance concept without an entity in question. This is the only reason why entity is written separately. Humans are aware of the object they are interacting with. For example: When they are lifting a can of soda, sitting on a chair or adjusting the tap of the sink. Robots should have a way to understand which entity they are interacting with.

The tuples of the first and second scenarios, which can be seen in the Figures 3-1 and 3-2, can be seen as follows:

$$\begin{aligned} &(\text{Door}, 5^\circ, \text{Go Forward}, \text{Remain Stationary}) \\ &(\text{Door}, 75^\circ, \text{Go Forward}, \text{Pass - through}) \end{aligned}$$

Second element of the tuples is the openness of the door, which belongs to the states of the entity. Including this, the robot will know that the determining factor that changes the effect is the state of the door. This will lead to more accurate predictions. This would not be the case with the representation of Sahin et al. since the robot would not be able to predict accurately without knowing the states.

In the passing through the door example, let us assume the minimum openness of the door for pass-throughability is 50°. Therefore two rules emerge from this situations:

$$\begin{aligned} &(\text{Door}, < 50^\circ, \text{Go Forward}, \text{Remain Stationary}) \\ &(\text{Door}, > 50^\circ, \text{Go Forward}, \text{Pass - through}) \end{aligned}$$

Note that in this example the states has been shrunk to only to the state of the door and everything else has been removed. This is what the idea affordance learning agent

should do. Removing the states that do not influence the effect and ending up with the full affordance knowledge.

Previous representations of the knowledge were given in such a way that they were specific to examples in question. However from these examples a more abstract representation should be acquired, which can be seen below:

$$(n, s, a, e) \quad (3-1)$$

When the robot is interacting with entity n ; when states are in the subset s , which is a subset of all the states, and agent performs the action a the resulting effect is e . Therefore the new state will be:

$$s_k = s_{k-1} + e_d$$

For a better visualisation of the representation a simple example will be given. Imagine a mobile robot that can move in one direction on a line. Its sensors can measure the position of this robot on this line and the battery level of the robot. Therefore the state will be represented as follows:

$$s_k = (Pos_k, Bat_k)$$

Where Pos_k is the position of the mobile robot on the line and Bat_k is the battery level of the robot at discrete instance k . The robot also has a sensor that can detect the entities that are in front of this robot and give labels to these entities. There are only 2 entities in this environment: Empty-space and a box. The robot can only use the “Go Forward” action. If there is an empty space in front of it, it can go forward and its battery levels will deplete by 2%. If there is a box in front of it, then the agent can only push it if its battery levels are above 30% and when it does its battery levels will deplete by 5%. This affordance knowledge can be represented as the following tuples:

$$\begin{aligned} & (Empty\ space, (\#, \#), Go\ forward, (+1, -2\%)) \\ & (Box, (\#, > 30\%), Go\ forward, (+1, -5\%)) \\ & (Box, (\#, < 30\%), Go\ forward, (0, -5\%)) \end{aligned}$$

The first, second and third tuples can be visualised as shown in the Figures 3-3, 3-5 and 3-4, respectively.

The effects of each actions has been shown in the Figures with a green arrow. These arrows show the direction of the effect as stated in the tuples that represent the affordance knowledge of this specific example. The advantage of showing the resulting effect and not the next state is that it is easier to generalize. This will be explained in an example; for this example entity will be removed from the representation. Therefore the representation will look like: state-action-effect. Imagine an agent that moves in an environment, where the state of the agent is represented as:

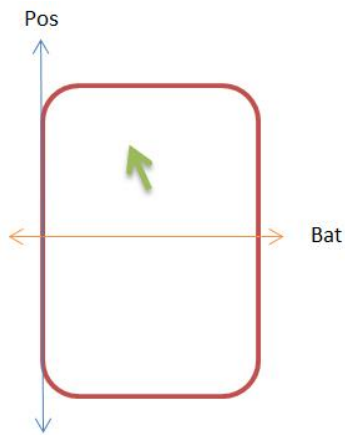


Figure 3-3: First tuple

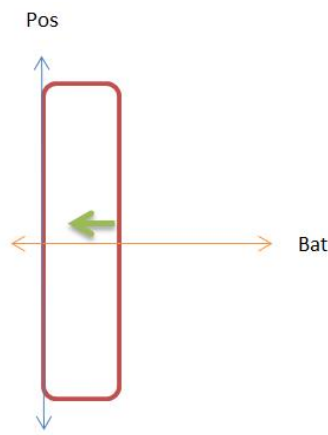


Figure 3-4: Second tuple

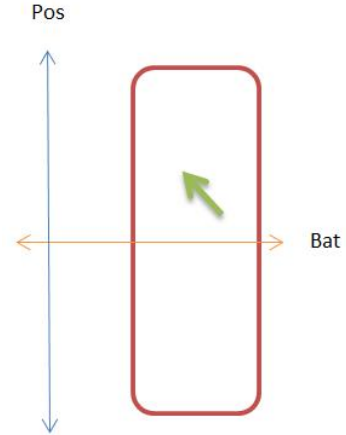


Figure 3-5: Third tuple

Figure 3-6: Visualisation of tuples

$$s_k = x$$

When this agent performs the action a_1 the effect will be $e_k = +1$. This effect will result in:

$$s_{k+1} = s_k + 1$$

So the rule will look like:

$$(\#, a_1, +1)$$

If this is represented as state-action-state it would be impossible to generalize. Because for every state there would be a need for a separate rule, such as:

$$(0, a_1, 1)$$

$$(1, a_1, 2)$$

$$(2, a_1, 3)$$

This is one disadvantage of using state-action-state representation. One might say the representation proposed in this study consists of entity-state-action-effect and not state-action-effect. This is correct. However it was mentioned before that technically the entity is one of the states of the robot since it is a feature extracted from sensory information of the robot. For example when the robot processes the image it will know that it is facing a button.

3-2 Action Planning

Action planning allows the agent to use its knowledge to plan sequence of actions in order to reach its desired state. For example in a navigation task the desired state can be the desired final position of the agent, whereas in object placement task the desired state can be such that all the objects are on top of each other. Another example is a motivation-driven agent that wants to satisfy its needs, which could be managing the battery or heat levels of the agent. However for all these separate applications one action planning mechanism can be used. Action planning in this study is done as follows:

- Agent senses the current state
- Agent predicts what will happen if it chooses to perform any action in its repertoire
- Agent anticipates the next state when it performs any of the actions in its repertoire
- Agent iterates until it reaches its destination

This action planning has been mentioned in the literature [9] as ‘Forward Directed Search’. Figure 3-7 makes it easier to understand the concept.

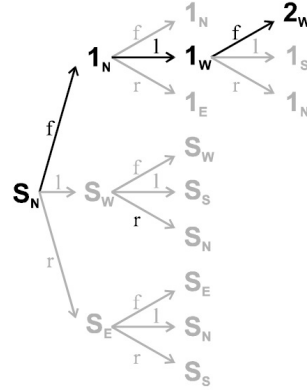


Figure 3-7: Forwards directed search method for action planning [9]

However in order to use the action planning the agent should have a transition model of the environment and itself. This transition model is the affordance knowledge of the environment. This transition model can be learned by experimenting with the environment and using these experiences to generate a model. How this is done will be described in section 3-6-3.

In order to use the action planning the agent should have an overview of the environment that it is acting in. Otherwise if the agent is put in a new environment and doesn't have an overview it will have to explore the environment to get an overview of it. With Reinforcement Learning (RL) if the overview of the environment is used

in the state-space then there will be an issue of curse of dimensionality. An example of an overview of an environment can be using Simultaneous localization and mapping (SLAM) algorithms to make a map of this environment using the camera images. Therefore the agent will be able to predict the state when it performs a set of actions. In different words, the sub-space of the state space should be observable by the agent in order to use action planning. The sub-space of the state space is the set of all possible states the agent will be in while performing a task.

3-3 Learning

There are three learning algorithms proposed in this section. Off-line learning has been tackled with first since it is less complex than on-line learning. On-line learning is more difficult since there is a coupling of action planning using the knowledge of the environment and generating this knowledge after performing actions which were determined in the action planning part. In off-line learning this problem is not present as the agent first gains experience then this experience is processed to create the knowledge that allows action planning to be used in the test stage. The idea to work on the hybrid learning idea came after observing the weak points of the off-line and on-line learning algorithms. However it was not thought of in the beginning of the study.

3-3-1 Off-line Learning

Below is the off-line learning algorithm in short and after that the detailed description follows.

Off-line Learning Algorithm:

- 1: Obtain tuples, such as 3-1 by doing exploratory movement and stack these tuples into the Experience Matrix (EM)
- 2: Remove duplicates in EM
- 3: Using the tuples in the EM generalize and build Population of Rules (PR)
- 4: Use the rules in the PR to

Obtain tuples

This phase can also be called as exploratory phase. In off-line learning the agent can act randomly and store the experience in the EM in order to save effort of the programmer.

Experience Matrix (EM): a matrix where the tuples are stored. This is used for generalizing the tuples and creating rules upon them.

The programmer can also guide the agent such that the agents experiences the important states rather than randomly experiencing the same task repeatedly. A more sophisticated exploration method can be used however these will not be investigated in this study.

Remove duplicates

It will be almost impossible not to encounter the same tuple more than once in the exploratory phase. This is not good since repeating tuples pollute the EM and do not bring anything new. This will slow the algorithm down since there will be a big data which is supposedly be smaller. This can be tackled in two ways. First way is as the agent performs an action and a new tuple is generated it can be checked to see if the same tuple has been generated before. If so this tuple will be neglected since it is already in the EM; if not this tuple can be stored in the EM.

Second way to do this is storing all the tuples in the EM and after the exploratory phase performing a duplicate removing. In this study this way has been selected. However there is no specific reason for this selection.

Generalization

The agent can use its previous experiences to perform the tasks in the environment it has already performed before. However if this same agent has to perform tasks in environments it has never seen before these experiences will not be enough for it or performing new tasks in the environments it has already has seen before. Therefore the agent has to have the ability to generalize the previous experiences to predict what will happen in states that it has never experienced before. After the generalization the agent should store these rules to use them again. In this methodology the storage for the rules is the PR.

Population of Rules (PR): is the set of all known rules

For example lets us consider the example of the mobile that can move only in one direction on a line. In a scenario where the agent starts at the states $s_0 = (1, 37\%)$ and performs 6 ‘Go forward’ actions consecutively. In this scenario if there was a box at the position 5, in other words the agent faced the box at position 4, the following tuples would be obtained:

$(Empty\ space, (1, 37\%), Go\ forward, (+1, -2\%))$
 $(Empty\ space, (2, 35\%), Go\ forward, (+1, -2\%))$
 $(Empty\ space, (3, 33\%), Go\ forward, (+1, -2\%))$
 $(Box, (4, 31\%), Go\ forward, (+1, -5\%))$
 $(Box, (5, 26\%), Go\ forward, (0, -5\%))$
 $(Box, (5, 21\%), Go\ forward, (0, -5\%))$

The ideal behaviour of the agent would be that the agent generates the rules, affordance of the empty space and the box, based on these tuples. The affordance of these entities have been determined before as follows:

$$\begin{aligned}
& (Empty\ space, (\#, \#), Go\ forward, (+1, -2\%)) \\
& (Box, (\#, > 30\%), Go\ forward, (+1, -5\%)) \\
& (Box, (\#, < 30\%), Go\ forward, (0, -5\%))
\end{aligned}$$

In other word this agent has to have the ability to build a transition model out of these previous experiences. However there is a chance that the set of rules, the affordance based transition model, generated may not be correct. For example if the agent never experienced pushing the box when its battery level below 30% then it would never know that the box would not be pushable when its battery level is low. Furthermore the agent would have generated a rule which is wrong such as:

$$(Box, (\#, \#), Go\ forward, (+1, -5\%))$$

This rule suggest the box is always possible independent of its position, which is correct, and its battery levels, which is not correct. However the generalization mechanism will think that this correct since the mechanism lacks the counter examples that contradict with this rule. With the off-line learning algorithm this will be an issue since the agent will not be learning on the run and constantly making mistakes due to the falsely learned rules, even it encounters the counter examples. This is a major disadvantage to the off-line learning algorithm. This can be solved by endowing the agent with the on-line learning ability. On-line learning ability will be described in the section 3-6-3.

The advantage of the off-line learning algorithm is that it only has to do generalization once whereas with the on-line learning the generalization should be constantly done. This might be problematic as the data in the EM gets large. Not only the amount of generalizations will be many but also the computational requirements will be high since the data involved in the generalization will be many.

3-3-2 On-line Learning

In dynamic environments, off-line learning will not work properly since it does not have the ability to learn on the run; for instance if a rule is not correct any more off-line learning algorithm would not able to correct it. The on-line learning algorithm, however, can.

The major difference between the on-line and off-line learning algorithms is that in the off-line learning algorithm the generalization procedure, which extracts affordance knowledge out of the previously experienced tuples, is done only once after the exploratory phase. In the on-line learning algorithm however the generalization procedure can be done after every step of the agent. In order to save time and computational power generalization can also be done once in every 5-10 steps, for instance.

In Figure 3-8 flow chart of on-line learning methodology. In this flow chart the agent first tries to create a path plan that will get it to its desired final state. If it can generate this path then it will follow it until it gets to the final state. If it cannot

generate this path then it will perform exploratory actions. After each action the agent will check whether the new obtained tuple was already experienced before. If yes then will continue with action planning for the next step; if not then the agent will perform generalization and generate rules.

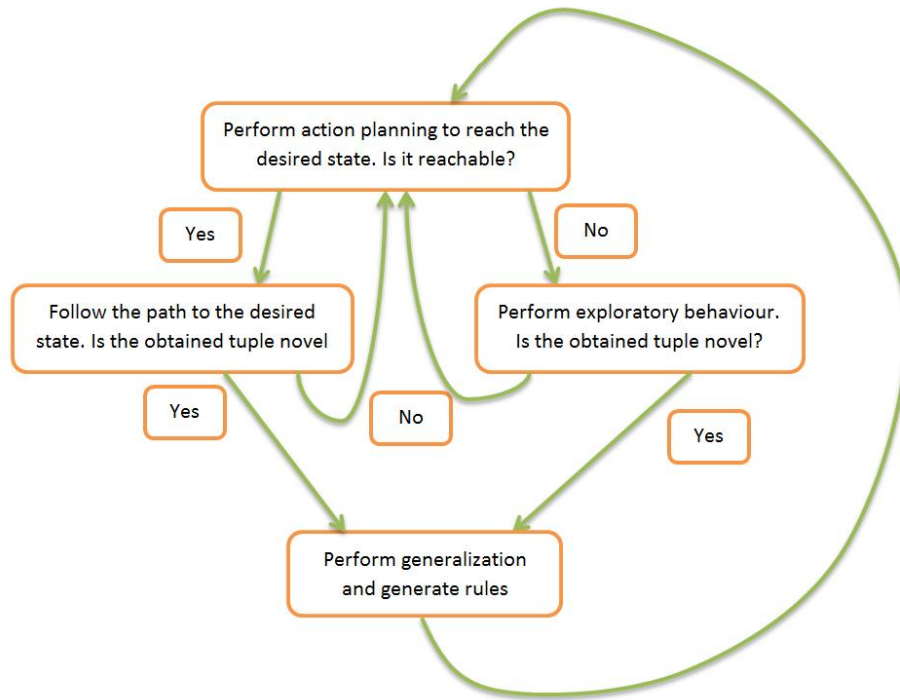


Figure 3-8: Flow chart of the on-line learning methodology.

Exploratory action

Similar to the off-line learning, on-line learning needs experience in order to learn. Therefore whenever the agent cannot find a path to its desired final state it will perform exploratory action. This exploratory behaviour can be random acting, guided by the programmer or something even smarter. However it will not be discussed which of these methods are the best in this study.

Generalization

Whenever the agent experiences a novel tuple, which it never experienced before, it initiates generalization and rule generation process. Generalization is same with the off-line learning except generalization in on-line learning is done more than once.

A big disadvantage to on-line learning algorithm is that generalization has to be performed constantly, which takes a lot of time and computational power. A disadvantage for the off-line learning is that it cannot correct the false rules on the run. It might be a good

idea to combine these two in order to workout their disadvantages. An alternative algorithm hybrid learning algorithm is mentioned in the section 3-3-3.

Check consistency of the rules

After obtaining a novel tuple the existing rules in the PR should be checked for consistency. The reason for this is that if the environment is dynamic some of the old rules will not be correct any more. If the rules are not consistent they are removed from the population. Assigning fitness values to the rules and decreasing these fitness values when the rules are not consistent is also a possibility. It might be more suitable for stochastic environments than immediately removing a rule from the population when they are wrong once.

As an improvement, if the rules are not consistent the future agents can be programmed to look for new information which might be affecting the effects and that are overlooked before. In other words, rules should be dynamic. For instance there might be a factor in the environment which does not appear in the rules but influences the effects. It is logical to add this determining factor into the future rules to get more accurate representation of the environment and the robot. However this topic will not be elaborated more since it is out of the scope of this study.

3-3-3 Hybrid Learning

The previous algorithms, off-line and on-line learning algorithms, have their disadvantages. In order to make a stronger point it was decided to combine these two and try to minimize their handicaps. Their handicaps were not being able to adapt on-line, for the off-line learning algorithm, and taking very long time to run calculations, for the on-line learning algorithm. These two were combined to create Hybrid Learning Algorithm. This algorithm works as follows:

- Off-line learning generates rule based on the experience it already has
- These rules are passed to on-line learning algorithm
- On-line learning algorithm tries to fulfil its task while learning on the of the rules passed from the off-line learning algorithm

The hybrid learning algorithm has the best of both worlds. It can be seen as the middle point of on-line and off-line learning algorithms. Hybrid learning has not been tested. However in a scenario where there are already rules and experiences generated by the off-line learning algorithm it will be logical to pass these rules and experiences to the on-line learning algorithm. Therefore the on-line learning algorithm will accept these rules just like it generated these rules and experiences. This will save time since in off-line learning algorithm generalization is run only once, whereas in on-line learning algorithm it is run at every step. Furthermore if there are rules which are not correct the on-line learning algorithm will be able to correct these on the run, unlike off-line learning algorithm.

3-3-4 Rule Generation

By using generalization the agent can plan its actions in environments which have not been experienced before. The idea is using the previous experiences to predict what will happen in new tasks and environments which have never been experienced before by the agent.

Earlier in this Chapter generalization mechanism has been mentioned. However, only the inputs and the desired outputs to this mechanism has been described. This is because there are many machine learning methods that can function as the generalization mechanism such as ‘Decision Trees’ and ‘Artificial Neural Networks’. However in this study a logic based learning is used. In literature this is also mentioned as ‘Inductive Reasoning’. Before going into this logic based learning the representation will be described.

As mentioned earlier, technically the entity is one of the states of the robot. Therefore the formalism used in this study resembles of Anticipatory Classifier Systems (ACS), which is:

$$State - Action - Effect$$

In this study only the representation of the ACS has been used since the Genetic Algorithm (GA) as a rule generation mechanism has been found to be not very suitable for this task. Therefore further information about ACS will not be given, for more details please refer to work of Stolzmann [9].

The reason to use these logical operations and not any other generalization algorithm is that it is more intuitive and easy to modify for the author. Eventually it turned out that it wasn’t the best generalization method out there. However the purpose of this study is to show that the knowledge transfer using affordance concept is possible and not finding the best algorithm for that.

Learning is slightly different for off-line, on-line and hybrid learning however the main idea is using the experiences to extract key and irrelevant sensory information that do/do not influence the future outcome. For instance in an environment the position of the agent has no direct impact on its action possibilities. It does not matter if the agent is here or there. What really matters is the entity the agent is interacting with. For example the docking station in front of the agent might allow ‘Charging of batteries’. This is because there is a docking station in front of the agent and not because the agent is at a specific location. If the docking station is moved elsewhere than the previous locations will not allow ‘Charging of batteries’ however the docking station will. Ideally the agent should learn this.

3-4 How Does Algorithm Handle Knowledge Transfer

The proposed methodology can handle knowledge transfer when the initial & final states are changed and when the transition model is changed. Changes to the initial&

final states can be seen as, basically, the agent starting from a different state than it did when it was trained and at the same time trying to reach a different final state which it has never tried to reach. This is implicitly handled by the State-Action-Effect representation that is borrowed from ACS. This representation allows action planning, therefore a general knowledge is sufficient for the agent to get to different final states.

In RL it is not possible to benefit from transfer learning without further learning needed when the initial & final states are changed. When the final state is changed then the agent has to relearn the new rewarding mechanism which will direct it to its new final state.

Changes to the transition model is a bit more complex than changes to the initial & final states. The whole aim of this algorithm is to salvage as much of the previous knowledge and use it when the transition model changes. For instance if swimming capabilities of a robotic agent changes it shouldn't reset all its knowledge and start from beginning. Instead this agent should benefit from the still usable knowledge, for instance the knowledge that relates to locomotion on the ground of this agent.

Above mentioned knowledge transferring will be mentioned in more detailed in the following subsections.

3-4-1 Changes in the state space

Changes in the state space can be agent starting from a different initial state or trying to reach to a different final state in the same environment. Furthermore it might even be the case the state representation remain the same however the state space gets bigger. Moreover it might also mean that the environment changes in such a way that every entity behaves the same however these entities are placed somewhere else in the state space.

Changes to starting and final states

The algorithm proposed in this study can handle when the starting and final states and the size of the state space are changed. There are two reasons for this. First of all the rule representation is state-action-effect. As mentioned in the previous sections with this representation it is possible to generalize.

Second reason is that proposed algorithm uses action planning using the known rules. Action planning and generalization of the tuples allows the agent to get to parts of the state space that have not been explored before.

Changes to the environment

It might be the case that the entities in the environment are displaced. Then these entities will still behave as they used to however now they are in a different place in the state space.

Affordance concept allows the algorithm to handle this change. Since the agent can use the affordance based transition model in the new environment, as long as the environment properties remain the same. In other words if, for example, empty-spaces, boxes and walls behave the same the agent will be able to find a route to reach its final state. And this is the case as shown in the Chapter 4.

With transfer learning for Q-learning it will take time since the values, for example, on the Q-table are not correct for the new environment and the Q-learning Agent (QA) will require some learning time to get used to it. This is claim is supported by the results in the Chapter 4.

3-4-2 Changes to the transition model

Changes to the transition model can be caused by two reasons. One of them is the changes of the properties of the environment. For instance if there is an electrical outage then the docking stations will no longer charge the batteries of the agent. Second cause is the changes of the capabilities of the agent. For example, one of the wheels of the agent is broken and cannot navigate in the environment any-more. However it can perform other tasks that it can do, such as grabbing boxes with the robotic arm attached on top of it.

The idea of the knowledge transfer in this study is to use as much information from the previous learning as possible. The agent should not start another learning from scratch but it should salvage as much information as possible. Agent should only learn the differences between his new capabilities and the previous.

The proposed algorithm tackles this problem as follows: Whenever a rule is not consistent with the newest experience that rule is deleted from the population of rules. However the rest of the rules in the population are still usable by the action planning.

3-5 Suggestion for Knowledge Transfer Between Different Morphologies

Knowledge transfer between similar or same robots is relatively easy compared to knowledge transfer between different robotic morphologies. There are two reasons for that. First is that because of the sensory configuration of robotic agents there might be more than one state representation for the same situation.

For example a humanoid robot might have its location sensor on its head whereas the other one might have it in its chest. Therefore they will have different state values, even though they might be in the same situation. Because of this the affordance knowledge generated by one of the robots will not be consistent for the other. In order to overcome this problem there should be mapping between the sensors of two robots.

Obtaining a mapping between sensors of the robots might be another problem. It might be solved by having a training session for both of the robots in a common environment.

When they are in the same situation their sensory readings can be mapped together. This might be a solution.

The reason is that different morphologies will most likely have different set of actions and capabilities. This can be solved by labelling the actions therefore the robotic agents will know which actions are similar, such as kick, go forward, rotate the valve and so on. However labelling all the actions might be a difficult task. Another solution can be using action equivalences, which has been mentioned earlier in this study. Action equivalences can be considered as a mapping between the actions of two robotic agents.

Most likely this approach will not be sufficient or not very effective. However it is only mentioned to guide the initial search in the field. This topic will not be discussed further in this study since it is out of the scope of the research.

3-6 Algorithm Used for Testing

In order to test the capabilities of the proposed methodology a study case and working learning algorithms are needed. By working algorithm it is meant that since the methodology is described generically an algorithm that tackles this study case is needed. These are explained in this Section.

3-6-1 Task Description

In a Simulated Environment (SE), which will be mentioned in the section 3-6-2, the mobile agent should be able to reach its destination. In off-line learning agent performs lots of trials to gain experience with the environment. After gaining enough experience it uses its experience to generate rules which describe the working mechanism of the environment. Agent uses these rules to do action planning in the test stage.

In on-line learning the agent is not given any prior knowledge and it should be able to learn and reach its destination at the same time. Agent only has a repertoire of actions and can read the sensory information. If enough knowledge to reach its destination is known then the agent tries to reach its destination, if it cannot then the agent tries to gather some more knowledge.

In order to test to see if the proposed methodology can realize these demands first a SE is built, a grid-world. This is explained in the following Subsection. After that both the off-line and on-line learning algorithms are explained in detail.

3-6-2 Grid-world

In order to develop and test the methodology, a SE has been designed and programmed in MATLAB. An example of this environment can be seen in Figure 3-9. This kind of environments are called 'Grid-world' in the literature. A Simulated Agent (SA) interacts with the SE. The SA has an array of elements as sensory input and can

perform one action at each time step. In the Figure, the SA is at position (0.5, 0.5), facing North and is represented by a T; lower part of the T is the front of the agent. The blue lines indicate the walls whereas the red lines indicate the button on the wall and the green line is for the door.

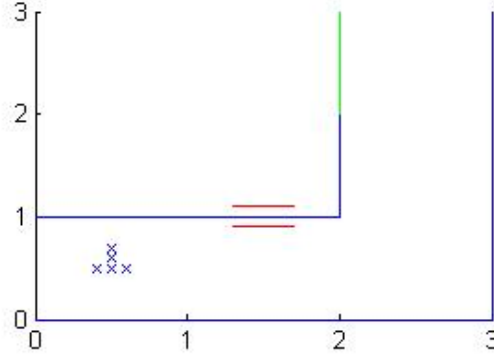


Figure 3-9: Example of a room with 3 grids at each axis

SE is chosen to be a discrete environment in order to be able to solve the learning problem easily; on the other end of the scale there are the challenges of building a continuous simulation, designing a learning algorithm which has continuous representation and more difficulties are present. In this environment there are 4 entities: Wall, door, button and empty space. The SA is not able to go through a wall or a closed door however it can go through an opened door and an empty space. Since the button is on the wall the button does not allow the agent to go through itself.

With just a couple of lines of programming the whole room can be changed completely. Since the novel room designs challenge the generalization ability this feature is important for this study.

Sensors and Actions of the Simulated Agent

The SA is able to perceive 6 features from the environment:

- S_a is the information of the object ahead of the SA. The wall is denoted as 1, door as 2, button as 3 and empty space as 4.
- S_b represents the state of the button. If the value is 1 if button is pushed and if it is 2 then it is pushed.
- S_d represents the state of the door. If the value is 1 if the door is opened and if it is 2 then it means it is closed.
- S_x , the position of the SA in X-axis. S_x takes values $x-0.5$ for $x=1, 2, 3...$
- S_y , the position of the SA in Y-axis. S_y takes values $y-0.5$ for $y=1, 2, 3...$

- S_o is the orientation of the SA. Value 1 means the SA is oriented towards the North, 2 East, 3 South and 4 West.

The agent is able to perform 4 actions: Turn left, go forward, turn right and push the button. Simulation mechanism allows the agent to rotate freely independent of what the SA is facing or where it is. The SA can only go through an empty space or an opened door. Push button action only has an effect when the agent is facing a button. The button works as a switch. If the button is pushed, $S_b = 1$, push button action releases the button, $S_b = 2$. If the button is not pushed, $S_b = 2$, push button action brings the S_b to 1.

3-6-3 Learning

As the SA gets more experience with the environment it should be able to generalize and predict what will be the consequences of its actions. In order to do this, a learning algorithm based on ACS representation and a logical operations to generate new rules are used. The reason to use these logical operations and not the genetic algorithm is during prior tests genetic algorithm often got into loop of creating too general rules and deleting these because they were mostly wrong.

This learning is slightly different for off-line and on-line learning however the main idea is using the experience to extract key and irrelevant sensory information that do/do not influence the future outcome. For instance in this SE the position of the SA has no direct impact on its action possibilities. It does not matter if the SA is at the starting position of (0.5, 0.5) or destination position (7.5, 9.5). These numbers do not mean anything special, they are given randomly in order to explain that the position has no influence on the action possibilities. What really matters is the entity the SA is facing. Ideally the agent should learn this.

The SA should have some experience in order to start generalization. The experience representation will be introduced in the subsection 3-6-3.

3-Tuple

The SA is able to store its experience with the environment in its memory. These stored experience instances are called tuples. These tuples are made of 3 parts: the state of the SA, the action performed by the SA, the effect caused by this action. The tuple is represented as follows:

$$Tuple : (State) - (Action) - (Effect)$$

Note that the entity is hidden in the states since it is technically one of the states as it was mentioned earlier. In this SE a tuple looks like this:

$$(S_a^k, S_b^k, S_d^k, S_x^k, S_y^k, S_o^k) - (u^k) - (E_a, E_b, E_d, E_x, E_y, E_o)$$

In this SE, the effect is the difference between the new state and the previous states. Effect is shown as follows:

$$(E_a, E_b, E_d, E_x, E_y, E_o) = (S_a^{k+1} - S_a^k, S_b^{k+1} - S_b^k, S_d^{k+1} - S_d^k, S_x^{k+1} - S_x^k, S_y^{k+1} - S_y^k, S_o^{k+1} - S_o^k)$$

An example of an experience instance, a tuple, is as follows:

$$(4, 1, 1, 0.5, 2.5, 1) - (2) - (0, 0, 0, 0, 1, 0)$$

This experience instance says that the SA was facing an empty space(4), the button was pushed(1), the door was opened(1), the position was (0.5, 2.5) and the SA was oriented toward North(1). After performing go forward action(2) the effect was a “+1” in the position on Y-axis, which was represented as the 5th element in the effects. Therefore the new state of the agent is:

$$(4, 1, 1, 0.5, 3.5, 1)$$

The tuple comes from the ACS. However ACS is not directly used in this study, but a variant of it, which is developed during this study, will be used. The learning algorithms will be described in the sections 3-6-3 and 3-6-3. For more information about ACS please refer to work of Stolzmann [9].

Off-line Learning

Off-line learning has been tackled with first since it is less complex than on-line learning. In on-line learning there is a coupling of action planning using the knowledge and generating this knowledge after performing actions which were determined in the action planning part. In off-line learning this problem is not present as the SA first gains experience then this experience is processed to create the knowledge that allows action planning in the test stage. Below is the off-line learning algorithm in short and after that the detailed description follows.

Offline Learning Algorithm:

- 1: Obtain tuples by doing exploratory movement and write these tuples into the EM
- 2: Remove duplicates in EM
- 3: Find stochastic effects
- 4: Remove stochastic effects
- 6: Generalize
- 7: Remove duplicates in PR

Obtain tuples In off-line learning the SA acted randomly and stored the experience in the EM in the SE in order to save effort of the programmer. A more sophisticated exploration method is suggested in the on-line learning in subsection 3-6-3.

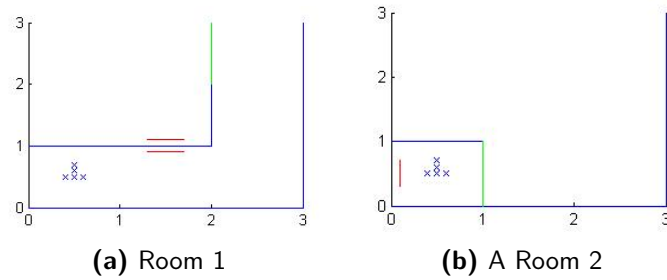


Figure 3-10: Two Rooms of same size, different configurations

Remove duplicate Because the SA acted randomly a lot of duplicates occurred. These duplicates should be removed which do not bring anything new except they pollute the EM.

Remove duplicate function is also used in the PR. This is a mechanism to check whether the newest rule created is in the PR. If so adding it to the PR will only pollute it.

Find stochastic effects After removing all duplicates from the EM the stochastic effects are removed from the effects. The stochastic effects are the remaining part of the effect when the deterministic part of the effect is removed. Stochastic part comes from the fact that the sensors of the SA is in such a configuration that the problem is not a Markov Decision Process (MDP).

As an example, in the Figures 3-10a and 3-10b two rooms of the same size are seen. However the configurations of the doors, buttons and walls are different. For both of the rooms, at this position and orientation, sensory input to the SA is:

$$\text{States in each room} : (1, 1, 1, 0.5, 0.5, 1)$$

If the SA chooses to turn right, the next state of the agent in room 1 will be:

$$\text{Next state in room 1} : (4, 1, 1, 0.5, 0.5, 2)$$

For the room 2 the next state will be:

$$\text{Next state in room 2} : (2, 1, 1, 0.5, 0.5, 2)$$

These two states in separate rooms and performed action would be recorded in the EM as the following two tuples:

$$\text{Tuple 1} : (1, 1, 1, 0.5, 0.5, 1) - (3) - (3, 0, 0, 0, 0, 1)$$

$$\text{Tuple 2} : (1, 1, 1, 0.5, 0.5, 1) - (3) - (1, 0, 0, 0, 0, 1)$$

It is visible that at the next state the first element of the sensory input, which is the object in front of the SA, is different for different rooms. However the rest of the states are same. This happens because of the sensory configuration of the SA. If the SA had a sensor on his right side this would not happen for ‘Turn right’ action, since the initial states would not be equal. Therefore it can be concluded that the first element of the states are stochastic for different rooms. However the rest are deterministic.

Replace stochastic elements After finding stochastic element, one of the tuples are removed and the effect of the other is modified as follows:

$$Tuple\ 1 : (1, 1, 1, 0.5, 0.5, 1) - (3) - (\#, 0, 0, 0, 0, 1)$$

Notice that a ‘Do not care’ symbol ($\#$) has been put at the first element of the effects, meaning that the change of this state does not concerns us anymore. This solution may not always work in different applications. However in this environment removal of the stochastic effects is needed in order to have a MDP. After finding a stochastic effect, the whole population is checked and if tuples have same effects(including stochastic and deterministic effects), the stochastic parts are replaced by the ‘Do not care’ symbol.

Generalization The algorithm starts from the first element of the EM and picks a tuple at a time to generate rules upon. Lets call this tuple, Tuples to Generate Rules Upon (TTGRU).

For each TTGRU, a matrix is created from very similar state-action pairs; this matrix can be called Similarity Matrix (SM). By similar it is meant that state-action parts of every tuple in this matrix is same as state-action part of TTGRU except for one element. For instance if the TTGRU is:

$$Tuple\ 1 : (1, 1, 1, 0.5, 0.5, 1) - (3) - (\#, 0, 0, 0, 0, 1)$$

Then the SM can look like this:

$$Tuple\ 5 : (1, 1, 1, 3.5, 0.5, 1) - (3) - (\#, 0, 0, 0, 0, 1)$$

$$Tuple\ 13 : (1, 1, 1, 2.5, 0.5, 1) - (3) - (\#, 0, 0, 0, 0, 1)$$

$$Tuple\ 17 : (1, 1, 1, 0.5, 1.5, 1) - (3) - (\#, 0, 0, 0, 0, 1)$$

$$Tuple\ 45 : (1, 1, 1, 0.5, 0.5, 1) - (1) - (\#, 0, 0, 0, 0, 3)$$

$$Tuple\ 57 : (4, 1, 1, 0.5, 0.5, 1) - (3) - (\#, 0, 0, 0, 0, 1)$$

$$Tuple\ 61 : (1, 1, 1, 0.5, 0.5, 1) - (2) - (\#, 0, 0, 0, 0, 0)$$

Different elements in state-action parts of each tuple are written in *italic*. The tuples in the SM which have the same effect as the TTGRU are compared and different elements

of the state and actions are determined to be not influencing the effect. For this example after comparing the tuples 5, 13, 17 and 57 with TTGRU it can be concluded the 1st, 4th and 5th elements do not influence the effect.

However when tuples in SM which have different effects (45 and 61) are compared with the TTGRU, it is revealed that the 7th element, which is the action, influences the effect. It can be concluded that the action is a key factor determining the effect. It can be argued that, it is obvious that the action will affect the effect, however in certain situations several actions might result in same effects. Furthermore the task of the agent is to learn a model based representation of the environment, without any prior information given to it.

After figuring out which elements do and do not influence the effects, it is a question of how to create rules. There are two ways considered in this study: Aggressive and conservative. Aggressive way generates rules which have ‘Do not care’ symbol at every element except for the elements which have an influence on the effect. Conservative way only puts ‘Do not care’ symbol at the elements which were found to be not affecting.

The rule that would be created by aggressive manner:

$$\text{Rule 1 : } (\#, \#, \#, \#, \#, \#) - (3) - (\#, 0, 0, 0, 0, 1)$$

The rule which would be created by the conservative approach:

$$\text{Rule 2 : } (\#, 1, 1, \#, \#, 1) - (3) - (\#, 0, 0, 0, 0, 1)$$

The difference is in the states of the rule. Depending on the application any of two could prove to be better. By better it is meant finding rules that are correct faster. However since the rules have to be consistent with the EM, there is a small chance of creating a wrong rule for both approaches. In off-line learning this could be a problem when SA is executing a task and experiences a new situation because the SA will not be able to correct the rules on the run. However this is not a problem for the on-line learning. The on-line learning algorithm checks the rules whenever a new experience is gained therefore ensures the correctness of the rules. On-line learning will be described in more detail in subsection 3-6-3.

Created rules will cover situations which were not experienced before because of the ‘Do not care’ symbol. For instance the rule 2 will be able to cover these situations:

$$\text{Sensory input 1 : } (3, 1, 1, 10.5, 20.5, 1)$$

$$\text{Sensory input 2 : } (2, 1, 1, 1.5, 95.5, 1)$$

$$\text{Sensory input 3 : } (4, 1, 1, -10.5, 1.5, 1)$$

It is easy to see that the 1st, 4th and 5th elements of these sensory input, states, are covered by the rule 2. These elements are written in *italic*.

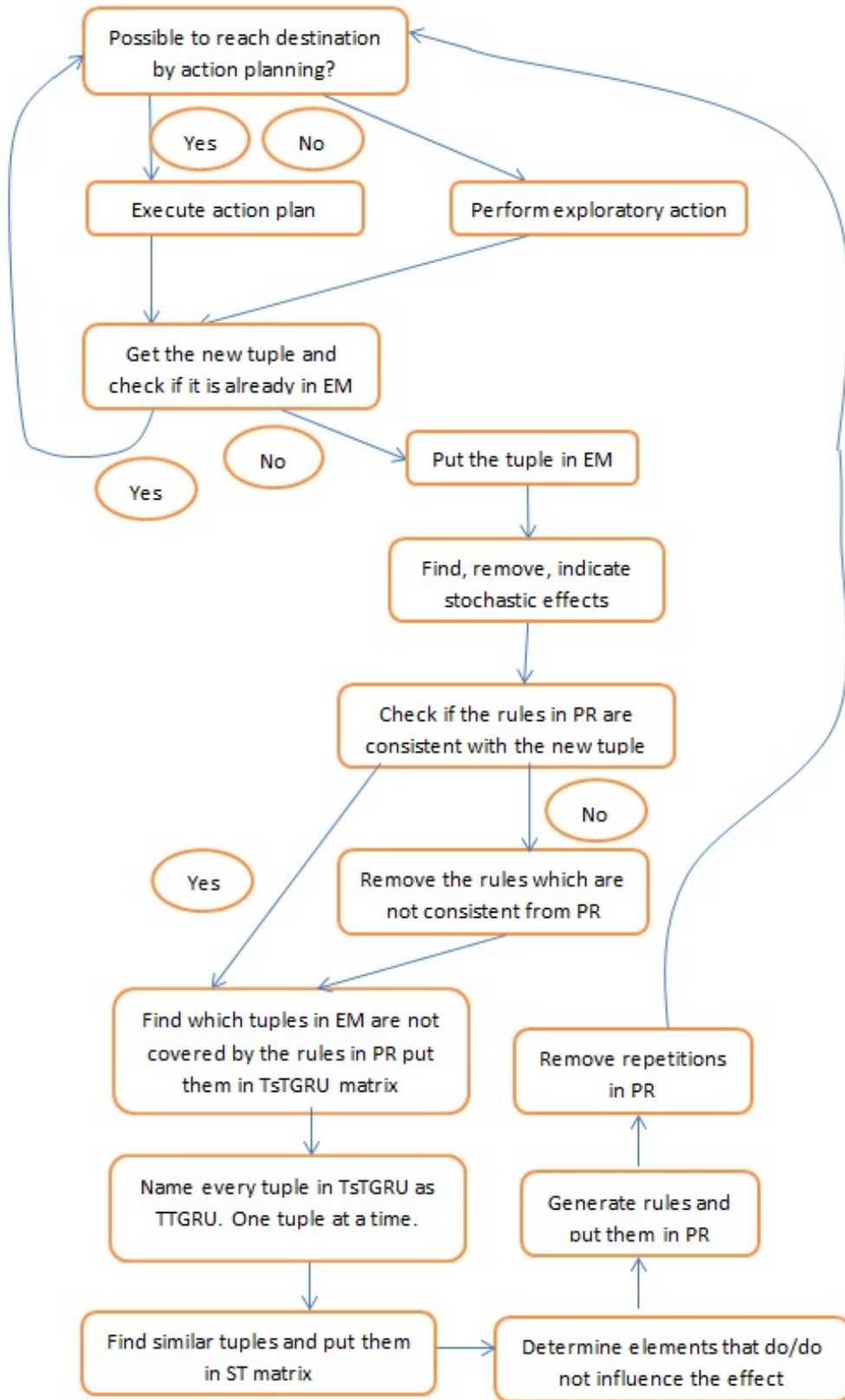


Figure 3-11: Block diagram of online learning

It is possible to give prior knowledge to the robot by writing rules and adding them to the population of rules after off-line training. This is valuable if the programmer does not want the robot to try dangerous actions which could be costly. This can be done both in off-line and on-line learnings.

On-line Learning

The problem with the off-line learning is that in order to work in a static environment it has to explore and learn sufficient amount of knowledge. Furthermore in dynamic environments, off-line learning will not work properly since it does not have the ability to learn on the run; for instance if a rule is not correct any-more off-line learning is not able to correct it. On the other hand on-line learning allows learning on the run and does not have the disadvantages of the off-line learning.

Exploratory action Similar to the off-line learning, on-line learning needs experience in order to learn. Therefore in the beginning exploratory behaviour is selected automatically since the action planning most likely will not be able to find a path to the destination with very limited knowledge.

Stochastic effects In on-line learning stochastic parts are not removed completely as they were in off-line learning. However they are used to extract deterministic parts of the new experience, as well as old experiences. Stochastic parts of the effects are indicated with the 'Do not care' symbol (#).

Check consistency of the rules At every step of the on-line learning & action planning algorithm, the new coming tuples are compared with the already existing rules, to see if these rules are consistent. If the rules are not consistent they are removed from the population. This is beneficial in terms of recognizing a dynamic entity in the environment.

As an improvement, if the rules are not consistent the future agents can be programmed to look for new information which might be affecting the effects and overlooked before. In other words, rules should be dynamic. For instance there might be a factor in the environment which does not appear in the rules and influence the results of actions. It is logical to add this factor into the rules.

Find tuples which are not covered by the rules After checking to see if the rules are consistent or not the SA runs a search in its whole EM to see which tuples, experience instances, are not covered in the population of rules.

Then the algorithm uses these not covered tuples to generate new rules. Rule generation process is same as the one in off-line learning. It is not controlled if the newly generated rules subsume the already existing rules for a reason: If a general rule covers specific rules, specific rules disappear from the population and if the general rule is proven to

be not correct later then it will be deleted. As this general rule is deleted the specific rule will also disappear with it.

Subsuming means that when a rule is more general than another rule and both of them suggest the same action, the more general rule is enough therefore the more specific rule is removed from the population of rules. For instance, below there are two rules where rule 3 subsumes rule 4; because rule 3 has ‘Do no care’ symbol at elements which rule 4 also has ‘Do no care’ symbol however rule 3 has extra ‘Do no care’ symbols meaning that it covers more situations than rule 4:

$$\text{Rule 3 : } (1, \#, \#, \#, \#, 1) - (3) - (\#, 0, 0, 0, 0, 1)$$

$$\text{Rule 4 : } (1, 2, 2, \#, \#, 1) - (3) - (\#, 0, 0, 0, 0, 1)$$

As an improvement a mechanism to bring back specific rules in case of a general rule not being correct should be added. This was a problem with the Extended Learning classifier System (XCS), the algorithm always subsumed specific rules aggressively and general rules always tend to fail at some point, which resulted in losing information. This process has repeated several times resulted in a loop. For more information about the XCS please refer to work of Butz and Wilson [19]

After generating new rules and removing duplicates in the PR, the simulation advances to the next time step. And the whole process is repeated until the SA reaches its destination.

3-7 Summary

This Chapter first explains the important concepts for this study. An affordance formalism from the literature has been explained then the modified version of this formalism has been mentioned. The modified formalism has several advantages which are also explained in this Chapter. Then the action planning used in this methodology has been described.

After the important concept the methodology has been explained, all three of the learning manners. After this how the methodology handles changes in tasks have been discussed and suggestions for knowledge transfer between different robotic morphologies has been made. Lastly the algorithms used for testing the methodology have been presented in detail.

Results and Comparison

The capabilities of the proposed methodology have been presented here. The capabilities are being able to transfer knowledge, that was obtained in source tasks, to target tasks which have different state spaces, initial & final states and/or transition models. These elements of differences have been described in Chapter 2.

First the off-line learning algorithm has been tested in various settings. Then experiments for transfer learning for Q-learning have been executed and compared with the off-line learning algorithm. After that experiments regarding to the on-line learning algorithm have been conducted. Finally the Chapter is concluded with a short summary.

4-1 Off-line Learning

In order to test the off-line learning algorithm, it should get some experience with the environment first. For that the Simulated Agent (SA) was trained in 5 different rooms designs, all of these can be seen in the Figure 4-1. In this image the walls are shown as blue lines, the door is shown as a green line and the buttons are shown as red lines.

Initial results of the off-line learning showed that the SA was not able to generalize (extract the affordance knowledge of) the entities door and button because it didn't have enough experience with them. The SA didn't have difficulties generalizing the walls and the empty spaces since there are many of these entities in an average room design hence learning these entities was easy.

Not being able to learn the properties of doors and buttons problem occurs because there is only 1 door and two buttons in one unique room and only 5 different room designs. However the rule generating mechanism requires more than this much. Since it would take time to design different rooms, so that the SA can have more experience with the buttons and doors, two rooms with lots of buttons and doors in them are designed. This is purely to save time. One of the rooms can be seen in the Figure 4-2.

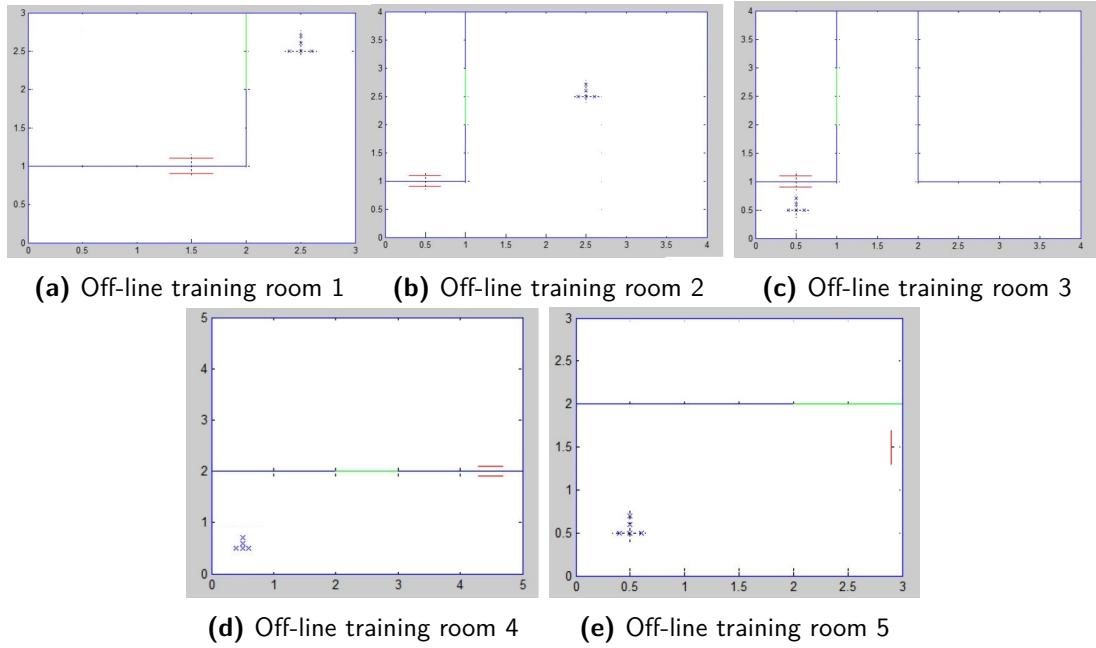


Figure 4-1: Initial training rooms for the off-line algorithm

If one button in this room was pushed state of every button will be changed and the state of the every door will be changed as well.

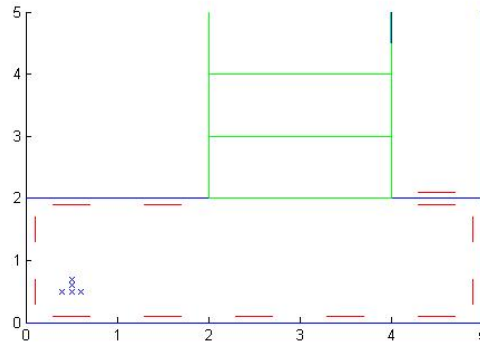


Figure 4-2: A room with lots of doors and buttons

In total the SA experienced 13 different locations for the door and 18 different locations for the buttons. This was sufficient for the algorithm to generalize based on the experience. The same results would have been attained if 13 rooms existed with different button and door locations.

In total SA took 18300 random steps in these 7 rooms. Because of the random steps there are repetitions of some of the tuples. Therefore repeating tuples are deleted. The remaining 1404 are all unique tuples. With unique tuples the SA has been trained and it generated rules that allow action planning as it is described in Chapter 3.

It was observed that after this training the SA was able to extract the affordance

knowledge of the doors and buttons. The SA could reach its destination by using the action planning method mentioned in section 3-2. In the Figure 4-3 test environment can be seen which was never experienced before by the SA. It can also be seen that the wall designs are different than the rooms where the SA was trained.

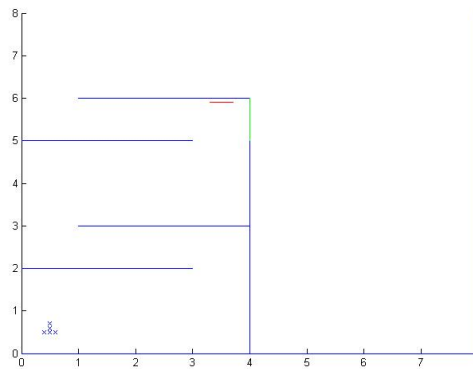


Figure 4-3: Test environment for the offline learning and action planning

The SA was able to reach to points (5.5, 3.5), (7.5, 2.5) and (3.5, 7.5) from the starting point of (0.5, 0.5) in minimum possible steps required, 27, 30 and 28 steps, respectively. In order to reach all destinations SA pushed the button and passed through the door. If SA did not use the door it would have taken it 36, 39 and 31 steps, respectively. This proves that the SA was able to generalize the knowledge of the door, button, walls and empty spaces since SA has not experienced these entities in such a combination like this room before. This test environment requires the agent to interact with these entities outside of the X and Y-axis range where SA was trained before. The door is only interactable at points (3.5, 5.5) and (4.5, 5.5) and the button is only intractable at point (3.5, 5.5) in this test room. However the biggest room design the SA experienced during training was 5 by 5.

These experiments support the claim that the proposed algorithm can transfer knowledge between tasks when the initial state, final state and the environment is different. The initial state differs since the SA was trained in 7 other rooms where it started in different locations. The final state differs since in this experiment the SA had three different final states. Furthermore while training the SA performed latent learning. In other words it didn't have any goal/desired state during training. The environment differs such that the properties of the objects in the environment remain the same however the placement of them are different.

Please note that transfer measures mentioned in Chapter 2 have not been used here since the SA performed optimally after the knowledge transfer without needing any further learning.

4-2 Transfer Learning in Reinforcement Learning (RL)

To see how capable the off-line learning algorithm is transfer learnings experiments for Q-learning have been performed for several tasks. First three Q-learning tasks have been performed without any transfer learning in order to observe how difficult it would be to scale up without transfer learning. Then transfer learning for Q-learning tasks have been performed in order to compare with the off-line learning algorithm, proposed here. Two stages of transfer learning have been performed. In the first stage only the final state has been changed. In the second stage of the transfer learning initial & final states and the room design have been changed.

4-2-1 Q-learning Agent (QA)

The states and the actions of the QA have been selected identical to the SA. In a 8x8 room the Q-table would be in the size of $4 \times 2 \times 2 \times 8 \times 8 \times 4 \times 4 = 16,384$ (4 entities, 2 states for the button, 2 states for the door, 8 possible positions on X-axis, 8 possible positions on Y-axis, 4 possible orientation for the agent, 4 possible actions).

It might look like this selection is not optimal for QA since there are many states and this might cause curse of dimensionality. For instance if only the state of the button (which is equal to the state of the door), positions on X and Y-axes, orientation of the agent and the possible actions were used as states then the size of the state space would be $2 \times 8 \times 8 \times 4 \times 4 = 2048$. However with the former selection of states the agent will only encounter a sub-space of the whole state space which is as big as the latter selection. The reason is the placement of entities in the rooms are static for a single room, which means when the agent is at a specific location and orientation the entity in front of it will only be one of the 4 for that room. Furthermore since the states of the button and door are equal to each other there will not be a case where the button is pushed (1) and the door is closed (0); or vice versa.

In the beginning of the first learning session Q-tables were only consist of values zeros.

4-2-2 Without any Transfer Learning When Final States Change

Without transfer learning the Q-learning is not scalable. The results in this subsection support this claim since for every new final state the Q-learning should be trained from scratch and this requires a lot of trials and time.

Three QAs have given similar navigation tasks. All QAs started from the same point with SA, point (0.5, 0.5) and oriented towards north. Destination of QA 1 was point (5.5, 3.5), QA 2 was point (7.5, 2.5) and destination of QA 3 was point (3.5, 7.5). The Q-learning values of QA 1 were:

$$\alpha = 0.5, \gamma = 0.95 \text{ and } \epsilon = 10\%$$

For QA 2 were:

$$\alpha = 0.5, \gamma = 0.95 \text{ and } \epsilon = 40\%$$

For QA 3 were:

$$\alpha = 0.5, \gamma = 0.95$$

Exploration ratio of QA 3 was decreased after some amount of trials:

$$\epsilon = 70\% \text{ for 10 trials, } 30\% \text{ for 25 trials, } 20\% \text{ for 15 trials,}$$

The reward function has been defined such that, the QAs get 100 points if they reach their destinations. In order to accelerate the learning of the QAs a punishment of 1000 points given if QAs do a move which result in no change in the effects. Examples for such behaviour can be when a QA tries to go forward when it is facing the wall or tries to use the ‘push button’ when it is not facing the button.

Amount of steps required for agents to reach their destinations are shown in the Figure 4-4

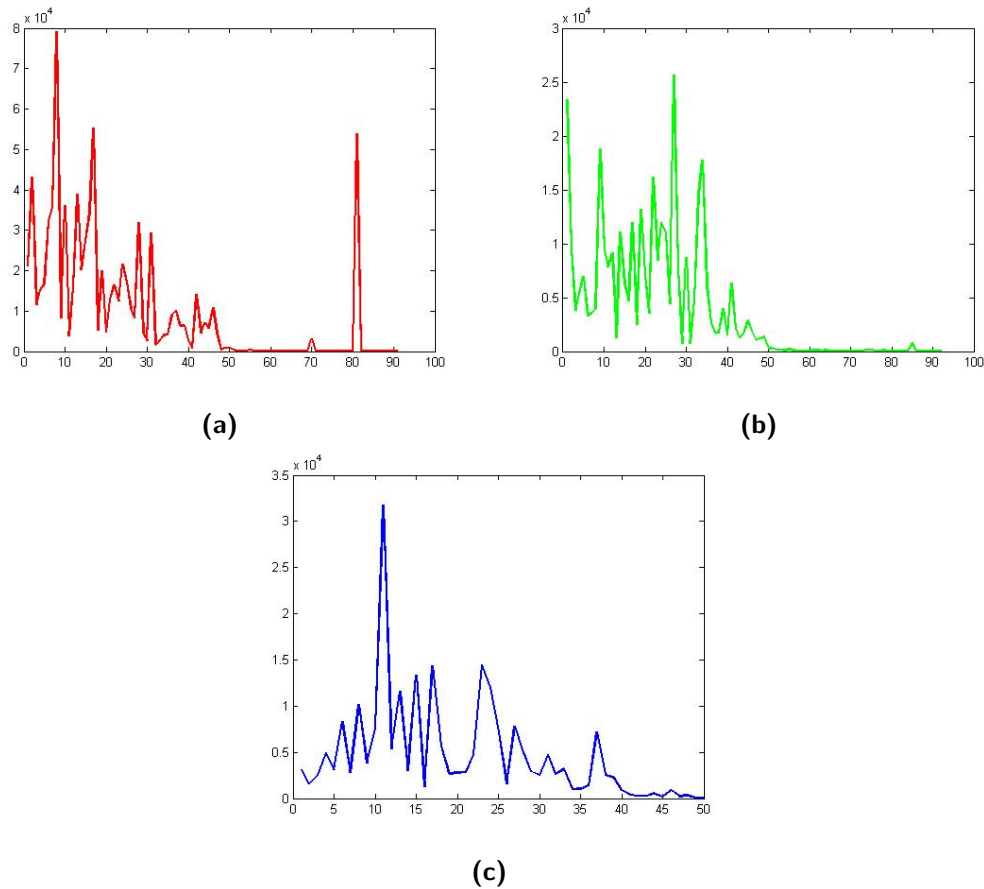


Figure 4-4: Steps required for all Q-learning agents to reach their destinations for every trial

Trials with full exploitory behaviour were performed after it was thought that the agents learned enough. However it turned out they didn't learn enough to perform optimally, which is reaching the final state in minimum amount of trials. The result for the exploitory behaviour for QA 1 is 37 steps to reach the destination. Minimum possible steps required to reach this destination is 27. This is because the agent learned to turn left 3 times rather than turning right once at 5 different locations in the room in order to reach the final state. This agent took 851,181 steps while learning and still the policy is not optimal.

The QA 2 reached its destination in 31 steps whereas the minimum required steps were 30. While navigating toward the destination QA 2 goes forward twice, turns right, goes forward 4 times, turns left and goes forward twice. The minimum amount of steps can be achieved if the agent performs these actions: go forward 4 times, turn right, go forward 4 times. This agent took 342,131 steps while learning however it reaches its destination in almost minimum amount of steps required.

The QA 3 reached its destination in 30 steps while the minimum amount of steps required are 28. This is because the agent turns left 3 times instead of turning right once in the beginning. This agent took 231,067 steps to learn to perform almost optimally.

All three QAs took 1,424,379 steps in total to be able to reach their destinations in a sub-optimal manner. In contrast the SA needed 18,300 steps to learn to reach its 3 different destinations in optimal number of steps. SA required much less steps to learn to get to a single destination, let alone being able to reach other destinations with the same knowledge gained from the training phase. If we assume other QAs learns at the rate of the QA 3, which is the fastest, they need in total of 14,788,288 trials to be able to get to all possible destinations in the test room without transfer learning.

It is easy to see that if the destination is changed Q-learning without transfer learning will take much longer than the Off-line Learning Agent.

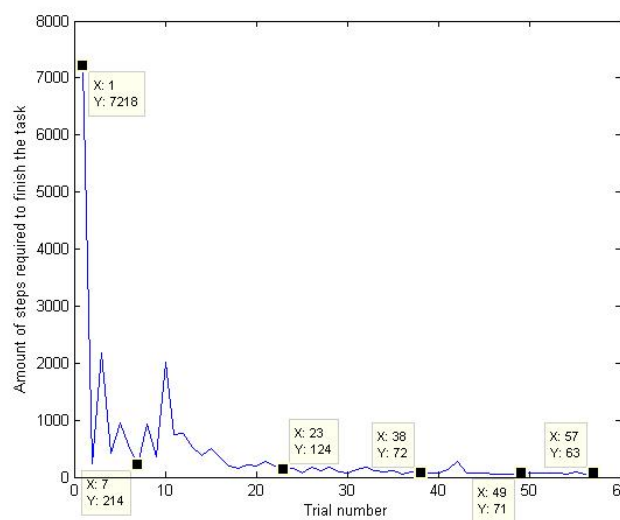


Figure 4-5: How many steps the QA2 took to reach its new destination in consecutive trials

4-2-3 With Transfer Learning When Final State is Changed

In the previous experiment it was shown that individual task learning for Q-learning requires a lot of trials compared to the off-line learning algorithm proposed in this study. The questions are with transfer learning how will Q-learning improve its learning speed and would this be as efficient as the proposed methodology.

QA2 has been used in this experiment, which is described in subsection 4-2-2. QA2 has the same parameters as before and used its previous Q-table. In other words, the starting Q-table will lead the QA2 to its previous desired final state which was (7.5, 2.5). The new desired final state is chosen as (3.5, 7.5).

The required amount of steps for the QA2 to reach its new destination can be seen in the Figure 4-5.

After this training session the agent performed this task with fully exploitory behaviour. The agent reached its new destination in 29 steps, whereas the optimum behaviour can be achieved with 28 steps. In total the QA2 took 22,912 steps during training. This is much better than learning from scratch.

Transfer learning proved useful in this match between the source and the target tasks. The reason for that is that, even though the final state was changed, a lot of the useful information was benefited in the new task. For example, trying to use the ‘Go forward’ action or ‘Push button’ action when facing the walls on the sides. In the first experiment with QAs they were trying these actions against the walls, without knowing that they will receive punishment.

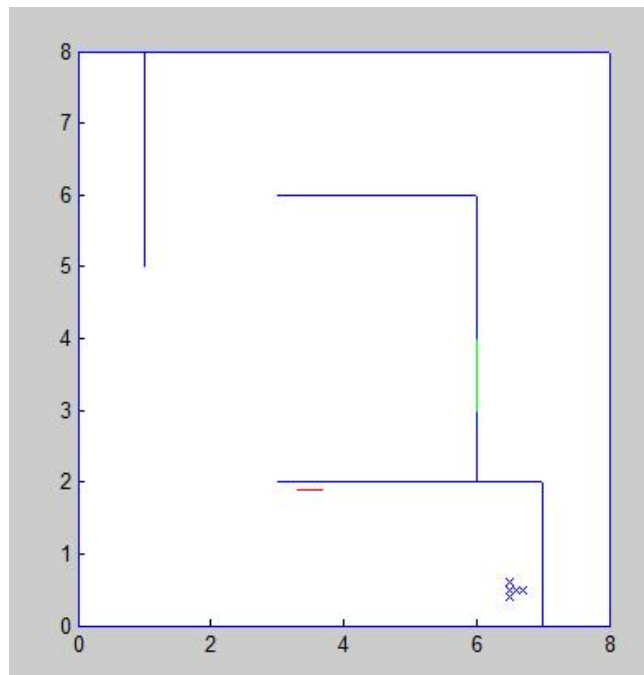
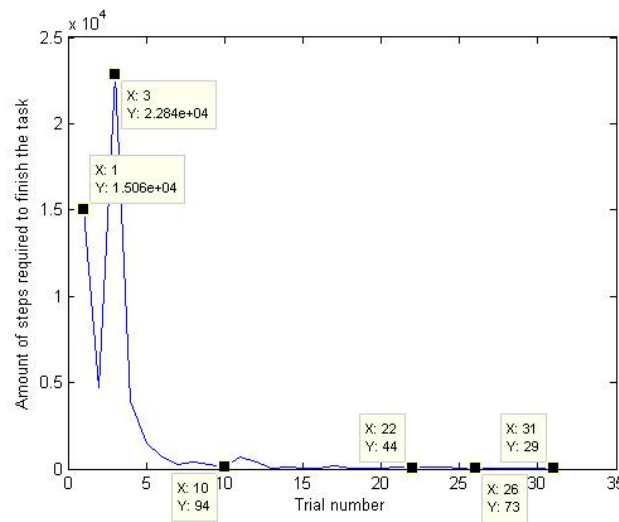


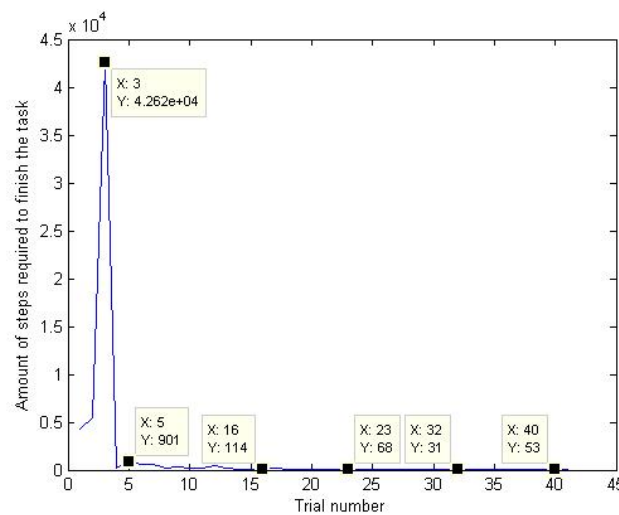
Figure 4-6: The test room used for testing transfer learning for Q-learning for a different room design

4-2-4 With Transfer Learning When the Room Design is Changed

In the previous experiment it has been observed that transfer learning for Q-learning increased speed of learning compared to Q-learning without transfer learning. In the previous experiment the source and target tasks shared the same room design however only the final state has been changed. This is relatively easy compared to a source and target task match where the room design is changed. This will be the case in this experiment. Furthermore in order to increase the difficulty the initial and final state has also been changed for this experiment.



(a) Different room design, different starting point, same vicinity of final state



(b) Different room design, different starting point, different final state

Figure 4-7: Transfer learning behaviour of QA2 where the room design, initial state and final state has been changed

The QA2 from the first experiment, which was described in subsection 4-2-2, has been used. Therefore the parameters and the Q-table have been imported to this experiment directly.

The room that was used for this task can be seen in the Figure 4-6. In this room the walls are shown as blue, door is shown as green and the button is shown as red. Two experiments have been conducted in this room. For the first one initial position was (5.5, 1.5) and the destination was (7.5, 6.5). Please note that this destination is in the same vicinity as the first experiment where QA2 learned from scratch, (7.5, 2.5). The agent learned how to get there, in optimal behaviour, in total of 51,983 steps. The optimal behaviour is taking 15 steps to reach this final state. The learning process can be seen in the Figure 4-7a.

For the second experiment the vicinities of initial and final positions have been swapped. The agent started the task at (7.5, 0.5) and the destination was set to (6.5, 0.5). The agent took in total of 58,075 steps to find a policy to get to its destination in an optimal behaviour, in 24 steps. Learning behaviour can be seen in the Figure 4-7b

In this experiment the QA2 required more steps to learn to get to the final states; 51,983 and 58,075 compared to 22,912 steps of the previous experiment. The reason for that is these source and target task matches were more different than of the previous experiment since the room designs are different.

It was expected that in this experiment QA2 would have taken longer to find an optimum policy to arrive its final destination. However this was not the case. The reason for this might be that the minimum amount of steps required to get from the initial points to the final points were less than the previous experiments.

In a task where the minimum amount of steps required to get to the final state is 5 steps and in which the agent can only perform 4 actions there is a $1/4^5$ chance that the agent will arrive its destination in the first trial while performing random exploratory behaviour. If the minimum amount of steps to get to the final state was 8 this meant the chances of doing so would be $1/4^8$. This is 64 times less likely to happen. This might be one of the contributing factors that enables the QA2 to find the optimal policy to reach the final state this quickly.

Even though the performance of the transfer learning is better than expected it is still worse than the off-line learning algorithm. Furthermore if the size of the test room was to be increased the transfer learning for Q-learning would have required more learning since the Q-table would have to be extended together with the room. The extended part of the Q-table would be zero since these states would have never been experienced by the QA2 before. In this scenario the transfer learning would only be beneficial in the tasks where QA2 only acts in the 8x8 part of the room where it used to act before but not in the newly added part.

4-3 Comparison of Off-line Learning Algorithm and Transfer Learning for Q-learning

The representation used in this methodology allows efficient knowledge transfer since it is not designed to direct the agent to specific states. The tuples and the rules only tell the consequences of the actions. Q-learning, on the other hand, does not tell the consequences but tells what reward the agent will get. This limits the amount of knowledge the agent can have, since changes of states are not preserved in the representation of the reward function. Because the reward function varies from task to task and agent to agent it is less efficient to transfer knowledge between tasks and robots with the representation of the Q-learning compared to the proposed methodology. It is not impossible though as it was shown in the previous experiments. Transfer learning for Q-learning made the QA2 perform better than it did when it had to learn from scratch.

In Q-learning there are 3 parameters, which makes it difficult to find the optimum set of parameters for learning. For example QA1 took 851,181 steps to reach its destination in 37 steps, which is 10 steps more than the optimal policy, but QA2 took only 342,131 steps to reach its destination in 31 steps, which is 1 step more than the optimal policy. It can be concluded that, getting the parameters right is important for Q-learning, whereas in the proposed algorithm suggested here this is not the case.

The memory requirement for the suggested methodology for this example was 63 kbs for experience instances and 1 kb for rules whereas the Q-tables for QA 1, QA 2 and QA 3 required 4, 4 and 7 kbs respectively for the first experiment. For few destinations this can be considered as disadvantage for the methodology. However in a scenario where the agent has to move to lots of destinations this will be an advantage since for 64 different destination, Q-learning agents will require at least $64 \times 4 = 256$ kbs of memory. It is assumed that each policy will require the smallest amount of memory that was found in this study.

The only problem with the off-line learning algorithm is that the agent will not be able to reach its destination if it learned something wrong or if the properties of the environment and the agent have changed. For example in a situation where the doors do not allow the agent to pass even when they are open. Q-learning algorithm is able to learn this and adapt however this is not the case for the off-line learning algorithm. A work around for this can be stopping the agent and then training it from the beginning using all the previous experience instances and the new ones. However rather doing so it is more clever to use an on-line learning algorithm.

The last experiment with transfer learning for Q-learning has been repeated again, however, this time the initial state has been changed to (0.5, 7.5). Since the QA2 didn't have a lot of experience with that part of the room it started to do random actions and wasn't able to find its way to the destination immediately. This experiment was done to show that how difficult it is to perform transfer learning with Q-learning. For every initial and final states agent has to go through a learning session whereas this is not the case for the proposed methodology.

4-4 On-line Learning

In order to test capabilities and observe the learning process better three test settings are used. The initial test is to see the increased speed of task completion in a basic scenario, the advanced test is in a same set-up as the off-line learning algorithm and the third where the SA has to complete a task when its capabilities are changed.

4-4-1 Initial Test

In order to test capabilities of the online learning, stochastic effects were given to the SA. The reason for this is, in order for SA to determine stochastic effects by itself, it should experience different room designs to figure out which effects are stochastic and which are deterministic. However there is only one test environment and SA would not have the chance to figure out the stochastic effects. This stochasticity occurs due to sensory configuration of the SA. Giving the stochastic effects to the SA beforehand makes the test setting an MDP.

In a room with 30 grids at each axis, the SA started from the point (0.5, 0.5) and the destination was set as (3.5, 3.5) at first. After reaching its destination the next destination was set to a further point which was 4 grids further from the previous destination on each axis: (7.5, 7.5), (11.5, 11.5), (15.5, 15.5) and so on. In the Figure 4-8 the amount of steps required by the agent to reach the destination can be seen. First destination which was at the point (3.5, 3.5) was reached in 112 steps, second destination which was at the point (7.5, 7.5) was reached in 12 steps and so on.

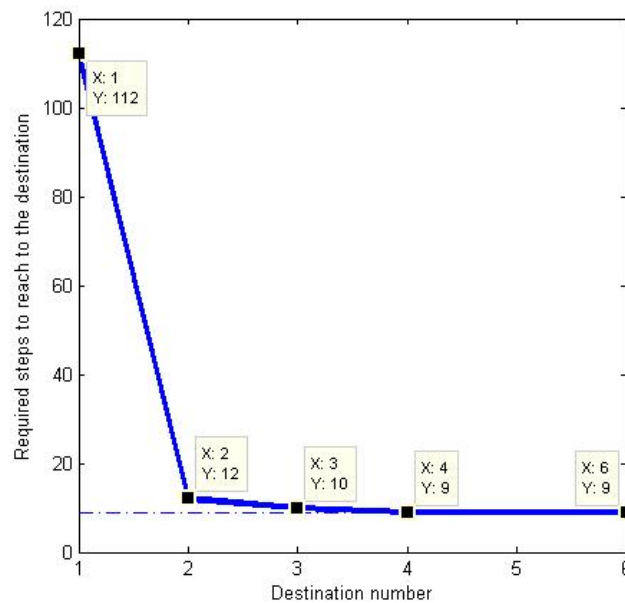


Figure 4-8: How many steps were required to for SA to reach its destination

In the beginning the SA explores most of the time; therefore it takes a lot of steps to reach its destination. However towards the end the steps required to reach the destination converges to the minimum possible steps, which is 9 including 4 steps North, 4 steps East and 1 step of turning right or left according to its initial position.

For this case only the time to threshold measure is applicable among other transfer learning measure, which are mentioned in the Chapter 2. As a performance measure a binary function is selected. This binary function gives the value 1 if the agent reaches its destination and the value 0 if the agent has not reached its destination yet. In the Figure 4-9 the performances values of the first 4 tasks can be seen.

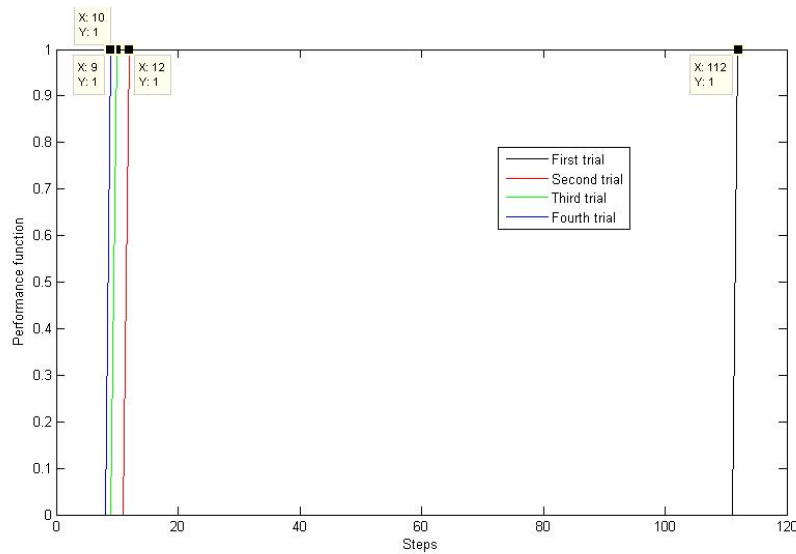


Figure 4-9: Performance function for each trial (First trial is represented as black, second as red, third as green, fourth as blue)

The second tasks inherits the rules and experiences learned in the first task. Likewise the third tasks inherits the rules and experiences from the first and the second tasks. With this knowledge transfer the SA is able to fulfil its tasks better each time until it reaches to the optimum behaviour.

It can be seen from the Figure 4-9 the time to threshold between first and fourth tasks is 103 steps, this concept has been explained in Chapter 2. Whereas the total amount of steps required for the fourth task is only 9. This shows the benefits of the on-line learning algorithm.

At the end there were 25 rules in the population. Some of which were not general since the subsuming of very specific rules has not been implemented. Subsuming has been mentioned in subsection 3-6-3 The rules can be seen in the table 4-1.

Since the rule subsumption is not implemented, specific rules which are being covered by the general rules are still in the population. For example rule 1, 4, 5, 8, 16, 18 and 19 are covered by the rule 18. Rule 18 says that if button is not pushed and the door is closed, the button and the door was not in the test environment therefore default inputs

Table 4-1: The rules in the population after the SA reached its final destination

Rule Number	State						Action	Effect					
1	4	2	2	1.5	1.5	#	4	0	0	0	0	0	0
2	4	2	2	0.5	#	1	2	#	0	0	0	1	0
3	1	2	2	0.5	#	4	3	#	0	0	0	0	-3
4	4	2	2	0.5	#	1	4	0	0	0	0	0	0
5	4	2	2	#	#	#	4	0	0	0	0	0	0
6	4	2	2	0.5	#	2	1	#	0	0	0	0	-1
7	4	2	2	#	#	1	2	#	0	0	0	1	0
8	4	2	2	0.5	#	#	4	0	0	0	0	0	0
9	4	2	2	1.5	#	3	3	#	0	0	0	0	1
10	4	2	2	0.5	#	3	3	#	0	0	0	0	1
11	1	2	2	0.5	#	4	1	#	0	0	0	0	-1
12	4	2	2	0.5	#	3	2	#	0	0	0	-1	0
13	4	2	2	0.5	#	2	2	#	0	0	1	0	0
14	4	2	2	#	#	2	1	#	0	0	0	0	-1
15	4	2	2	0.5	#	1	3	#	0	0	0	0	1
16	#	2	2	#	6.5	#	4	0	0	0	0	0	0
17	#	2	2	#	#	#	4	0	0	0	0	0	0
18	#	2	2	0.5	#	#	4	0	0	0	0	0	0
19	#	2	2	1.5	1.5	#	4	0	0	0	0	0	0
20	4	2	2	#	6.5	2	2	#	0	0	1	0	0
21	4	2	2	#	6.5	1	3	#	0	0	0	0	1
22	4	2	2	#	#	3	2	#	0	0	0	-1	0
23	4	2	2	#	#	3	3	#	0	0	0	0	1
24	4	2	2	#	#	1	3	#	0	0	0	0	1
25	4	2	2	#	#	2	2	#	0	0	1	0	0

Table 4-2: The most general rules in the population after the SA reached its final destination

Rule Number	State						Action	Effect					
3	1	2	2	0.5	#	4	3	#	0	0	0	0	-3
7	4	2	2	#	#	1	2	#	0	0	0	1	0
11	1	2	2	0.5	#	4	1	#	0	0	0	0	-1
14	4	2	2	#	#	2	1	#	0	0	0	0	-1
17	#	2	2	#	#	#	4	0	0	0	0	0	0
22	4	2	2	#	#	3	2	#	0	0	0	-1	0
23	4	2	2	#	#	3	3	#	0	0	0	0	1
24	4	2	2	#	#	1	3	#	0	0	0	0	1
25	4	2	2	#	#	2	2	#	0	0	1	0	0

are 2, and SA performs "push the button" action anywhere in the room regardless of entity it is facing the effects will be zero. This is only true in this room since the SA did not experience any door and buttons in this room, the algorithm hasn't seen any experience which is a counter example therefore it concludes so.

If subsuming has been performed to the population of rules, there would only be 9 rules and not 25. These rules be seen in the table 4-2.

These rules are only sufficient for this task. If the destination was changed to a lower value in both X and Y-axis the SA would try to learn how to generalize some more rules. For instance, a rule which will have an effect of a decrease in X-axis. This rule is not in the current population yet since it wasn't needed to fulfil this task. Furthermore if the algorithm can plan a path that leads to the desired final state then it tries to go there and not to perform exploratory actions.

It can be conclude that this method of gaining experience of the environment and generalizing these experiences to extract affordance knowledge to use in action planning allows navigation in states which haven't been experienced before. This allows the SA to transfer knowledge to a different task and fulfil it in an optimal manner where the initial and final states are changed. With optimal manner, for this example, it is meant reaching to the desired state in the least amount of steps.

4-4-2 Advanced Test

This on-line learning experiment has been done while the SA tried to complete tasks in multiple rooms. These rooms can be seen in the Figure 4-10. The initial and final positions varied through each task, however the SA always started headed to North and the initial position of the button was the off position. Just like with the initial tests the SA started experiment number 1 without any prior knowledge or experience. However after each tasks the experiences and rules have been used in the following tasks.

The amount of time and steps required for SA to reach its destination can be seen in the table 4-3 with other details such as, initial position, destination and the room the agent performed in.

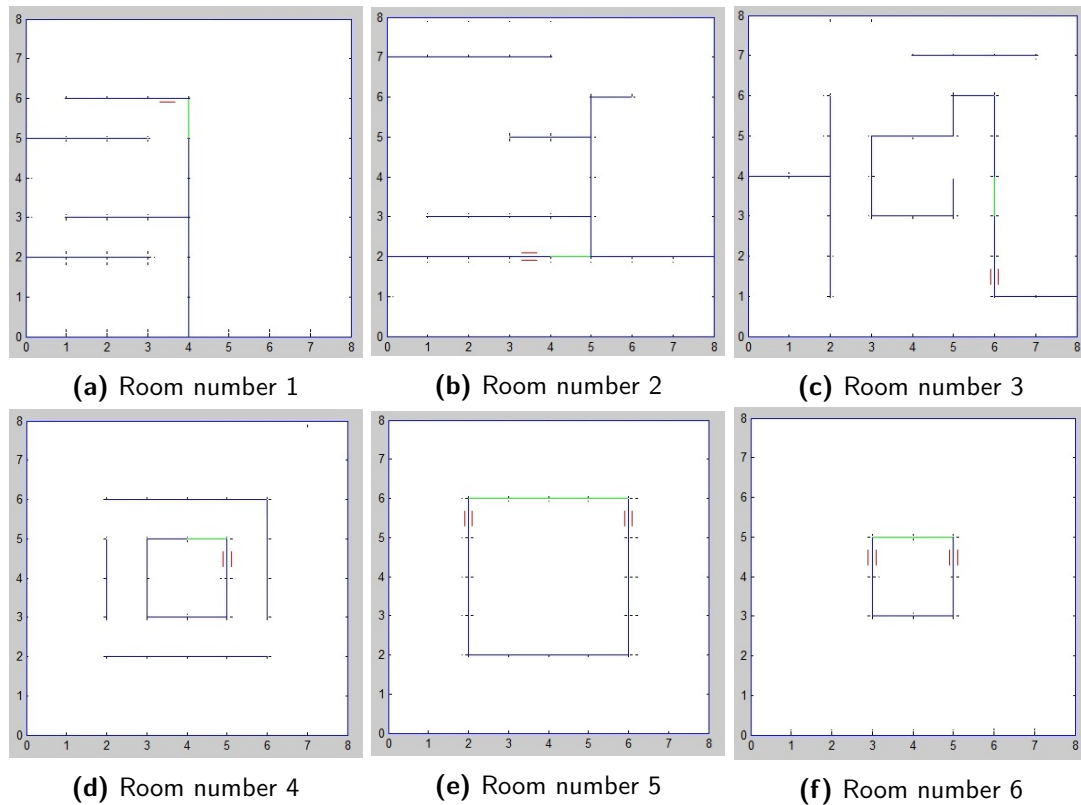


Figure 4-10: Task rooms for On-line Learning

After 20 experiments the SA has taken 18552 steps and all of the experiments took in total approximately 47 hours. Towards the end, the steps where the SA faced new tuples took more than two minutes. This shows the slowness of the algorithm when faced with new experience. The algorithm compares new tuples with the previously experienced tuples and starts the rule generation procedure which takes a while. However main purpose here is not showing that this algorithm is better than any other but to show that the suggested methodology also works on-line.

In the beginning the SA didn't have any experience or rules known to it. It started learning as it started the task. The experiment number 7 shows that the agent learned how to interact with empty-spaces efficiently since it reached its destination in the minimum possible steps using the empty-spaces, and not the door and the button. If the agent used the button and the door it would have reached its destination faster however the agent didn't have enough experience with the doors and buttons to generalize a rule. Therefore it wasn't able to plan actions using these entities. This phenomenon was faced with the Off-line Learning Algorithm as well. The Off-line Learning Algorithm learned to use these entities after being introduced to more experience with these entities.

In experiment 8, the SA was expected to reach its destination in small amounts of steps after observing its performance in experiment number 7. However this wasn't the case since the agent wasn't able to generalize enough to reach that destination. Experiments number 7 and 8 required different rules to be activated in order to do action planning.

Table 4-3: Task performance of On-line Learning Algorithm

Trial Number	Room Id.	Starting Point	Destination	Amount of Steps	Amount of Time(s)
1	1	(3.5, 3.5)	(3.5, 7.5)	3587	980
2	1	(7.5, 0.5)	(4.5, 6.5)	88	196
3	1	(4.5, 0.5)	(7.5, 7.5)	155	307
4	2	(4.5, 0.5)	(0.5, 2.5)	1565	2931
5	2	(5.5, 5.5)	(4.5, 4.5)	1242	4791
6	2	(0.5, 7.5)	(0.5, 6.5)	1709	3197
7	3	(4.5, 1.5)	(6.5, 4.5)	17	196
8	3	(7.5, 1.5)	(0.5, 4.5)	2253	22529
9	3	(7.5, 0.5)	(0.5, 3.5)	127	849
10	4	(4.5, 3.5)	(4.5, 2.5)	76	1944
11	4	(5.5, 5.5)	(6.5, 6.5)	12	74
12	4	(5.5, 5.5)	(6.5, 6.5)	12	3
13	4	(7.5, 1.5)	(4.5, 3.5)	6170	73574
14	5	(2.5, 2.5)	(4.5, 7.5)	500	15977
15	5	(0.5, 5.5)	(3.5, 3.5)	540	2618
16	5	(7.5, 0.5)	(0.5, 7.5)	71	19
17	5	(5.5, 5.5)	(7.5, 0.5)	166	4757
18	6	(3.5, 3.5)	(7.5, 0.5)	89	3213
19	6	(3.5, 3.5)	(7.5, 0.5)	74	27625
20	6	(3.5, 3.5)	(7.5, 0.5)	99	1381

This explains why it took a lot more to complete the experiment number 8.

Something similar to experiment 8 happened in experiment number 16. The agent wasn't able to do action planning until it moved randomly and reach the point (3.5, 0.5). After this point it was able to create an action plan which led the SA to its destination. This is because the SA didn't have enough experience to generalize and to find an action plan from its current state to final state therefore it wandered around for some time until it could find a path that lead to its final state. Therefore it was only able to sub-generalize the rules at certain areas in the state-space. However after certain amount of experience general enough rules will be present in the population of rules. This, indeed, was the case as after experiment number 20, it was observed that the SA had some rules which were as general as possible and correct at the same time.

The initial position and the destination for the experiments number 11 and 12 were the same. The agent took the same amount of steps to reach its destination. However there is a difference between the amount of time it took. This is due to the fact that there was one tuple on the way to the destination which was not experienced before. Therefore the agent took the time to run the rule generation mechanism on the 11th experiment. This shows that the SA was able to plan its path without actually experiencing specific states on the way before. This shows the generalization property of the algorithm. In the experiment number 12 the agent took less than 3 seconds to reach its destination, where it did not have to run the rule generation mechanism.

The experiments performed in the room number 6 started from the same point and

ended at the same point for all three experiments. However the results are different. This is due to the fact that the SA is not able to action plan from the starting area, area around the starting point. The starting area is inside the small room and the SA has to interact with a button and a door in order to reach its destination. However since it didn't have enough experience with buttons and doors it is not able to generate rules accordingly. Therefore it is not able to do action planning as long as it is inside the small room. However after leaving the room it still struggled with the action planning since it doesn't have a lot of experience to deal with states where the button is pushed. In other words the agent knows how to navigate well when the button is not pushed. However when the button is pushed it is a different subspace of the state space for the SA. Therefore the rules which apply to the subspace where the button is not pushed do not apply to the subspace where the door is open and the button is pushed. This because of the generalization method not being very good for the task. However it was chosen because it is intuitive and easy to use. Furthermore the aim of this work is not to get the best results but to show that the methodology works, even though it is slow.

In the table 4-3 it can be seen that there is a trend where the required amount of steps to reach the destination is decreasing. Please note that this is done at the same time the initial and final states and the room designs are being changed.

4-4-3 Test Where the Capabilities of the Agent Have Been Changed

The transition models of the environment and robot change when the properties of the entities in the environment and/or the designs of the environment are changed. Another reason for changes in transition models is the changes to the capabilities of the agent. For example, if the motors of a mobile robot are replaced with weaker ones then this mobile robot will not be able to push some of the objects it used to push.

For this experiment the ability of SA passing through a door was disabled. In a imaginary scenario this can be perceived as the agent carrying a load on top. Therefore does not fit through the door any-more. Another example can be transferring the knowledge of the SA to a SA2, which is much wider and therefore does not fit through the door.

The experiment has been performed in the same room as SA was tested before, which can be seen in the figure 4-3. The path SA took after its capabilities have been changed can be seen in the Figure 4-11. Its final state is (5.5, 3.5). SA reached the final state in 44 steps.

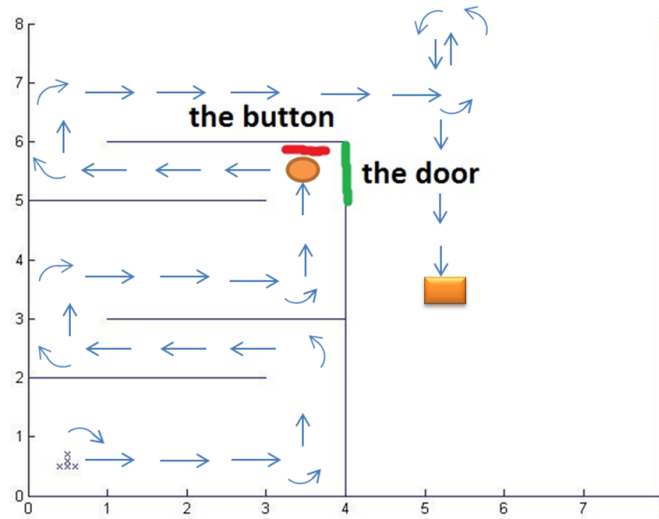


Figure 4-11: The test room that the knowledge transfer has been tested

The SA started from the initial point of (0.5, 0.5) and reached to the button as it would normally do. SAs position was at (3.5, 5.5) at this instance and this can also be seen as the orange circle in the Figure. It pressed the button turned right to face the door. Tried to move forward but was not able to. Then it learned that it is not able to do that any more. Turned left to face the button. Pressed it. Turned left and continued its path.

SA did a detour almost at the end since it wasn't able to generalize enough about knowledge of the environment. In a scenario where the decision making mechanism is much better than the one used in this study will not have this problem. The decision making mechanism is mentioned in Chapter 3.

The SA was able to learn that it was not able to pass through the door any more on the run. This happened as such:

- SA tries to pass through the door while following the path that was generated by the action planning
- SA is not able to pass through the door
- This new situation is recorded
- All of the rules are checked to see whether they are still consistent or not
- The rules that are not consistent, in this case the rule that says the SA can pass through the doors, are removed
- The SA does action planning again with the new set of rules, which doesn't have the rule that suggested that the SA can pass through the door
- The SA follows the new path that SA generated with action plan to reach its goal

This result supports the claim using the proposed algorithm an agent is capable of exploiting the transferred knowledge even though some of it is not correct any more. The agent gets rid of the wrong knowledge and continues to perform its task while using the rest of the transferred knowledge.

4-5 Summary

In this section results of the proposed algorithm have been exhibited and compared with results of transfer learning between Q-learning tasks. Detailed comparison between sophisticated Transfer Learning algorithms has not been done since the author has become aware of this field very close to the end of this study.

The proposed algorithm, compared with transfer learning for Q-learning, requires much less additional learning in order to fulfil new tasks. This also holds true even in these new tasks the initial and final states, designs of the environments and capabilities of the agent are changed.

It was realized during the experiments for on-line learning algorithm that the chosen generalization method, inductive learning method used in the experiments, is not very good since it took very long to execute learning sessions at the later stage of the experiment. The performance would be much better if better generalization methods were used such as decision trees.

During the experiments the agent was still learning and haven't mastered the environment it was in yet. If the agent was to obtain most accurate and general rules of the environment it would have performed optimally without requiring any learning in new tasks, provided that the properties of the environment and the robot remained the same. This would be the ideal scenario.

Conclusion and Future Work

5-1 Conclusion

In this work it has been found that concept of affordances is a good starting point to create a knowledge transfer technique between robotic tasks. The reason for this is that affordances maintain some sort of transition model of the environment and the robot acting in it. Though this transition model is not guaranteed to be correct.

The transition model in question provides enough insight to robotic agents such that they can plan and perform new tasks in new environments. Furthermore other robotic agents can also make use of this knowledge. In different words, other robotic agents can learn how to interact with entities and/or fulfil tasks in an environment without actually conducting a learning session in that environment with those entities. However this will only hold when two assumptions mentioned in this study are true.

To conclude the whole thesis a novel method of combining affordances with knowledge transfer has been developed. This methodology performed better than the existing transfer learning for Q-learning in a test environment. The methodology was better in the sense that when the agent was put in a new environment, which was not experienced before, it required much less trials to learn how to get to the final state compared to the transfer learning for Q-learning.

The generalization method chosen for the tests were not very suitable for the task since it took long time to generalize and generalizations were not very accurate. However the purpose was not finding the parts for the algorithm but to show that the knowledge transfer between tasks using affordance concept requires no or minimal amount of learning when a new task is presented, compared to transfer learning for Q-learning. By learning it is meant learning time and trials. And this was shown with the test results.

The proposed idea of combining affordances with knowledge transfer is new and should be improved. For example, the study case the algorithm developed and tested is a

discrete, deterministic and static environment. If this methodology will be used in robotics applications then it should be able to handle continuous, stochastic and dynamic environments.

One of the research questions was not address in detailed during this thesis work, which is transferring knowledge between different robotic morphologies. This is because there wasn't much time left to do so. All in all there are suggestions to create a methodology to perform such thing. These suggestions should be tested and improved if necessary.

To sum up, this thesis work can be a start of a methodology which enables efficient knowledge transfer between real world tasks, environments and robots.

5-2 Future Work

As mentioned couple of times the algorithm was tested in a discrete, deterministic and static environment. The next steps to prove the methodology should be developing it in such a way that it will be able to cope with continuous, stochastic and dynamic environments. This can be done by using a continuous representation for the tuples and rules, assigning fitness values to each rule and benefiting from a better generalization method.

Suggestions have been made to enable knowledge transfer between different robotic morphologies in the previous chapters. In short it was suggested to conduct training sessions for both robots in a common environment. Therefore these agents will have experienced same situations and based on this they can build a mapping between their sensors. In other words the agents will know whenever in a certain situation what will the other agent will sense. This way the robotic agents can abstract the affordance knowledge from their morphology to universal level.

Whenever a robotic agent wants to use an affordance knowledge it will then use the mapping between its sensors and the robot that discovered aforesaid affordances. However these are just suggestions and it might turn out that these suggestions are not applicable or not good enough. Therefore these should be tested and possibly further improved.

In an ideal scenario a robotic agent, mobile or humanoid, should be able to perform novel tasks in novel environments. Furthermore another robotic agent should be able to ask for relevant knowledge about the tasks or the environment in order to fulfil its tasks with minimum additional training. In short further improvements should be making the algorithm capable of coping with difficult environments -continuous, stochastic and dynamic- and mastering the knowledge transfer between different robot morphologies.

Bibliography

- [1] M. E. Taylor and P. Stone, “Transfer learning for reinforcement learning domains: A survey,” *The Journal of Machine Learning Research*, vol. 10, pp. 1633–1685, 2009.
- [2] J. J. Gibson, *The ecological approach to visual perception*. Psychology Press, 1986.
- [3] E. Sahin, M. Çakmak, M. R. Dogar, E. Ugur, and G. Ucoluk, “To afford or not to afford: A new formalization of affordances toward affordance-based robot control,” *Adaptive Behavior*, vol. 15, no. 4, pp. 447–472, 2007.
- [4] D. A. Norman, *The design of everyday things*. Basic Books (AZ), 2002.
- [5] D. N. Perkins and G. Salomon, “Transfer of learning,” *International encyclopedia of education*, vol. 2, 1992.
- [6] J. McGrenere and W. Ho, “Affordances: Clarifying and evolving a concept,” in *Graphics Interface*, pp. 179–186, 2000.
- [7] G. Humphreys, “Objects, affordances action!,” *The Psychologist*, 2001.
- [8] L. Bull and J. Hurst, “A neural learning classifier system with self-adaptive constructivism,” in *Evolutionary Computation, 2003. CEC’03. The 2003 Congress on*, vol. 2, pp. 991–997, IEEE, 2003.
- [9] W. Stolzmann, “An introduction to anticipatory classifier systems,” in *Learning Classifier Systems*, pp. 175–194, Springer, 2000.
- [10] R. J. Urbanowicz and J. H. Moore, “Learning classifier systems: a complete introduction, review, and roadmap,” *Journal of Artificial Evolution and Applications*, vol. 2009, p. 1, 2009.

- [11] I. Grondman, M. Vaandrager, L. Busoniu, R. Babuska, and E. Schuitema, "Efficient model learning methods for actor-critic control," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 42, no. 3, pp. 591–602, 2012.
- [12] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
- [13] J. H. Holmes, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, "Learning classifier systems: New models, successful applications," *Information Processing Letters*, vol. 82, no. 1, pp. 23–30, 2002.
- [14] W. Stolzmann and M. Butz, "Latent learning and action planning in robots with anticipatory classifier systems," in *Learning Classifier Systems*, pp. 301–317, Springer, 2000.
- [15] J. P. Seward, "An experimental analysis of latent learning.," *Journal of Experimental Psychology*, vol. 39, no. 2, p. 177, 1949.
- [16] E. C. Tolman, *Purposive behavior in animals and men*. Univ of California Press, 1951.
- [17] J. Hoffmann, *Vorhersage und Erkenntnis: die Funktion von Antizipationen in der menschlichen Verhaltenssteuerung und Wahrnehmung*. Hogrefe Verlag für Psychologie, 1993.
- [18] L. Torrey and J. Shavlik, "Transfer learning," *Handbook of Research on Machine Learning Applications. IGI Global*, vol. 3, pp. 17–35, 2009.
- [19] M. V. Butz and S. W. Wilson, "An algorithmic description of xcs," in *Advances in Learning Classifier Systems*, pp. 253–272, Springer, 2001.

Glossary

List of Acronyms

LCS	Learning Classifier System
XCS	Extended Learning classifier System
ACS	Anticipatory Classifier Systems
GA	Genetic Algorithm
RL	Reinforcement Learning
MDP	Markov Decision Process
HCI	Human-Computer Interface
PR	Popoulation of Rules
SA	Simulated Agent
SE	Simulated Environment
EM	Experience Matrix
SM	Similarity Matrix
TTGRU	Tuples to Generate Rules Upon
SLAM	Simultaneous localization and mapping
QA	Q-learning Agent

