FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Rafiqul Talukder
Teemu Kettukangas

ROBOTIC SENSORIMOTOR INTERACTION STRATEGIES

Bachelor's Thesis
Degree Programme in Computer Science and Engineering
June 2023

Talukder R., Kettukangas T. (2023) Robotic Sensorimotor Interaction Strategies.
University of Oulu, Degree Programme in Computer Science and Engineering, 33 p.

ABSTRACT

In this thesis we investigate the mathematical modeling of cognition using sensorimotor transition systems. The focus of the thesis is enactivism, where an agent learns to think through actions. As a theoretical basis for our implementation, we discuss a mathematical model of enactivist cognition, sensorimotor interaction and how they can be used as algorithmic aides for studying theoretical problems in robotic systems.

In Chapter 3 of this thesis, we introduce a platform which was developed in the University of Oulu as a software project and explain how enactivism and sensorimotor interaction have been taken advantage of, in developing a 2D platform. This platform enables one to concretely implement and explore different interaction strategies that allow an agent to construct internal models of its surroundings. The agent in the platform is a multi-jointed robotic arm, which maneuvers through an obstacle-filled environment. The robotic arm tries to explore its environment with minimal sensory feedback, using algorithms created by the user of the platform.

Our main goal on this thesis is to implement new features to this platform. We implement a memory functionality which allows the robotic arm to store all its performed actions. The memory helps the agent infer to a greater extent its surroundings from a limited sequence of action-observation pairs, and helps it in getting a better grasp of the environment. In addition, we implement other methods and functionalities, such as an obstacle sensor, a graph visualization of the internal models, etc. to enhance the perceptual ability of the robotic arm.

In Section 5 , we develop an algorithm for a simple 2D environment with no obstacles. Here the robotic arm makes a 360-degree move in four steps to perceive its surroundings and generates a state machine graph to visualize its internal model of the environment. The goal of the algorithm is to build an accurate representation of the environment with the help of memory.

Through this algorithm we are able to evaluate the performance of the newly implemented features. We also test the platform through unit testing for finding and resolving bugs.

Keywords: Transition Systems, Robotics, Enactivism

Talukder R., Kettukangas T. (2023) Sensorimotoriset vuorovaikutusstrategiat robotiikassa. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 33 s.

TIIVISTELMÄ

Tässä tutkielmassa tutkimme kognition matemaattista mallintamista käyttämällä sensorimotorisia transitio-järjestelmiä[1]. Tutkielman keskiössä on enaktivismi, jossa agentti oppii ajattelemaan toiminnan kautta. Teoreettisena perustana toteutuksellemme käsittelemme matemaattista mallia enaktivistisesta kognitiosta, sensorimotorista vuorovaikutusta ja kuinka niitä voidaan käyttää algoritmien apuvälineinä teoreettisten ongelmien tutkimisessa robottiikkajärjestelmissä[2].

Tutkielman luvussa 3 esittelemme alustan, joka on kehitetty Oulun yliopistossa ohjelmistoprojektina, ja selittämme miten enaktivismia ja sensomotorista vuorovaikutusta on hyödynnetty 2D-alustan kehittämisessä. Alusta mahdollistaa erilaisten vuorovaikutusstrategioiden[3] konkreettisen toteuttamisen ja tutkimisen. Näiden avulla agentti rakentaa sisäisiä malleja ympäristöstään. Alustassa mallinnettu agentti on moninivelinen robottikäsi, joka liikkuu esteitä sisältävässä ympäristössä. Robottikäsi pyrkii tutkimaan ympäristöään minimaalisen sensoritiedon avulla käyttämällä alustan käyttäjän luomia algoritmeja.

Tutkielmamme päätavoite on kehittää uusia ominaisuuksia tälle alustalle. Toteutamme muistitoiminnallisuuden, jonka avulla robottikäsi tallentaa kaikki suoritetut toiminnot. Muisti auttaa agenttia päättelemään enemmän ympäristöstään rajoitettujen toiminta-havainto-parien avulla, ja auttaa sitä ympäristön hahmottamisessa. Lisäksi kehitämme muita menetelmiä ja toiminnallisuuksia kuten estesensorin ja sisäisten mallien graafisen visualisoinnin parantaaksemme robottikäden havainnointikykyä.

Tutkielman myöhemmässä osassa kehitämme algoritmin yksinkertaiselle 2D-ympäristölle ilman esteitä. Siinä robottikäsi tekee 360 asteen liikkeen neljässä vaiheessa havainnoidakseen ympäristönsä, ja luo tilasiirtymäkaavion visualisoidakseen sisäisen mallinsa ympäristöstä. Algoritmin tavoitteena on rakentaa tarkka malli ympäristöstä muistin avulla.

Tämän algoritmin avulla pystymme arvioimaan kehittämiemme uusien ominaisuuksien toimintaa. Testaamme alustaa myös yksikkötesteillä löytääksemme ja korjataksemme virheitä.

Avainsanat: Transitio-järjestelmät, Robotiikka, Enaktivismi

[1] transition systems
[2] robotic systems
[3] interaction strategies

TABLE OF CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS

PE	Perception Engineering
UBICOMP	Ubiquitous Computing
RSSTP	Robotic Sensorimotor System Testing Platform
FSM	Finite-State Machine
NP	Nondeterministic polynomial time

1. INTRODUCTION

Typically, industrial robotic arms are equipped with numerous sensors, and are pre-programmed for a specific environment. However, imagine we stripped away most of the sensors and allowed the robotic arm to learn about its surroundings through trial and error while using minimal sensorimotor feedback. Can an industrial arm learn and comprehend its environment with minimal sensorimotor input? One can approach this question by investigating basic sensorimotor integration in a biological system. From this perspective, we can also obtain insights into how machines might learn and adapt in a similar way. Numerous theories have been proposed over the years to explain why and how organisms think [1, 2, 3, 4, 5]. Enactivism is one such theory. It emphasizes the role of interaction and embodiment (the constrains caused by one's physical presence in an environment) in cognitive processes.

Minimal theoretical models of enactivist cognition are currently an active research topic in the Perception Engineering (PE) research group at the Center for Ubiquitous Computing (UBICOMP) in the University of Oulu. A 2D platform has been developed in a software project [6] to learn sensorimotor interaction techniques, allowing a multi-jointed robotic arm to manoeuvre through a space with various obstacles. To evaluate the performance and functionality of the platform, an algorithm has been developed in the University of Oulu.

The primary focus of our project was to improve the robotic arm's ability to build a better model of its interaction by implementing new features into the existing software. For testing and training the robotic arm in its environment we have developed a new algorithm that significantly improved the agent's ability to model its interaction with its environment. In addition, we have implemented unit testing to test each part of the program's functionality to ensure optimal performance of the software.

This thesis is divided into five sections. In Section 2, we present an introduction to cognition and discuss mathematical modeling of cognition. Additionally, we explore how sensorimotor interaction and enactivim can be applied in robotic systems. In Section 3 we provide background information about the existing software platform that serves as the starting point of our thesis, and explain what different parts of this software does and how these software parts work together. In Section 4 we implement new features to the existing platform and in Section 5 we test our platform through algorithm and unit testing. Finally, in Section 6 we discuss our results and possible further development of the platform.

2. COGNITIVE THEORY OF SENSORIMOTOR INTERACTIONS

2.1. Theories of Intelligence

Cognition refers to the essential ability of humans and animals to interact intelligently with their environment, and to understand their surrounding world. Cognition allows us to sense, process information, make judgments and form beliefs. It is hard to concretely define what cognition means.

In recent years it has become a growing opinion among cognitive scientists that cognitive science is in need of a new perspective [7, 8]. According to Thompson traditional cognitive science lacks the understanding of human cognition because it does not take into account emotion, affect, or motivation [7]. A complete theory of the mind must also take subjectivity and consciousness into consideration [7].

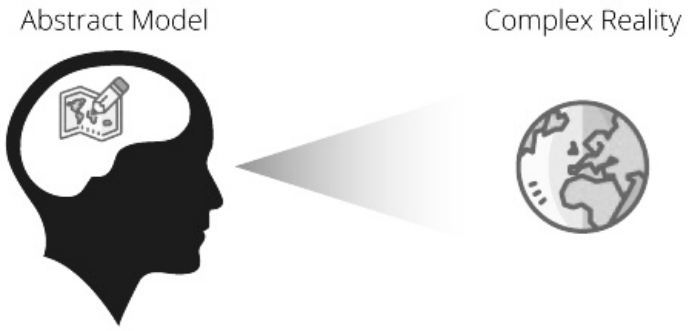

Figure 1. A representation of the outside world inside the human brain [9].

In cognitive science, there are three main approaches to studying the mind: cognitivism, connectionism, and embodied dynamicism. In cognitivism the mind is metaphorized as a digital computer, whereas in connectionism the mind is metaphorized as neural network. In embodied dynamicism, the mind is seen as an embodied dynamic system [7].

In cognitivism and connectionism, the world and the mind are considered as distinct entities that exist independently of one another, by mirroring the outside world as a representational model inside the head (see Figure 2). Embodied dynamics denies these assumption and takes a different approach toward cognition [7, 3].

2.1.1. Enactivism and Embodied Dynamicism

Embodied dynamicism questions most of the assumptions about cognition which have been presented by cognitivism and connectionism, especially the idea that cognition is a disembodied and abstract mental representation. Embodied dynamicism, like connectionism, emphasizes self-organizing dynamic systems, but it also stipulates that cognitive processes emerge from ongoing sensorimotor interactions between the brain, body, and environment through nonlinear and circular causation [7].

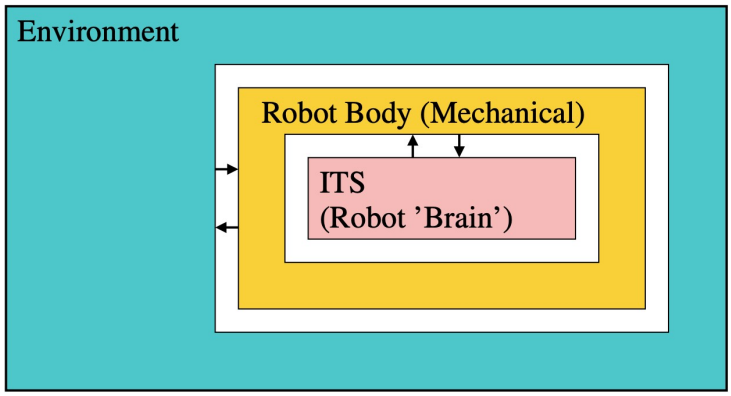

Figure 2. The environment, body, and nervous system (or brain) are modeled as inseparable parts of a coupled transition system [10].

Enactivism is often viewed as a subprinciple of embodied dynamicism. It shares many similarities with embodied dynamicism, and it stresses greatly the notion that cognition emerges from continuous interactions between an organism and its environment [7, 10]. Enactivism builds bridges between embodied dynamicist accounts of the mind and phenomenological accounts of human subjectivity and experience [7]. Together, these approaches offer a more comprehensive understanding of cognition.

The main underlying principles of enactivism consist of five basic enactivist tenets, formulated in [10] based on relevant literature. These tenets are as follows:

- (1) Embodiment. The environment, the body, and the nervous system are inseparable parts of the system and they cannot be meaningfully understood in isolation from each other [11, 12].
- (2) Groundedness. The organism's current interactive tendencies are explained by nothing other than its history of previous interactions. The brain does not "acquire" or "possess" contentful states, representations, or manipulate semantic information in any other way [12].
- (3) Emergence. The crucial properties of the brain-body-environment system from the point of view of cognition emerge when the agent's and the environment's prior structure come together to faciliate new structure which emerges through the sensorimotor engagement [13].
- (4) Attunement. Agents differ in their ways of attunement and they have different skills depending on their environments. A skill is a potential possibility to engage reliably in complex sensorimotor interactions with the environment [11].
- (5) Perception. Perception arises from skillful sensorimotor activity. Perceiving makes the agent better attuned to its environment [14, 13].

2.2. Sensorimotor Interactions

In philosophy of mind and cognitive science, sensorimotor interaction and enactivism are closely related. We discussed earlier how, from an enactivist perspective, cognition emerges from dynamic interactions between an agent and its environment. Sensory and

motor capacities play a critical role in determining how these interactions are shaped. Sensorimotor interaction approach to perception highlights the close relationship between perception and action [15]. It argues that perception is an active process of exploration and interaction with the environment rather than passive process of receiving and absorbing information from the environment. Sensorimotor interaction also suggests that action and precepting are deeply inbred with each other and they cannot be understood in separation.

2.2.1. Sensorimotor Interaction in Perception

In Sensorimotor Life [15], an example is given to help understand sensorimotor interaction in depth, and how it can be used to help teach robots about their environment. In [15], Ezequiel Di Paolo and Thomas Buhrmann explain sensorimotor perception through an imaginary setting. They provide the following example to illustrate, how an agent would be able to find an exit door when it is in a dark room full of curved walls (see Figure 3). The agent won't be able to walk through or jump over the obstacles because they are made of durable material, and they resemble tall brick walls. These walls are curved in the form of circles and spirals. As the agent walks around the room spectating curved walls the light goes out because of a power failure, making it impossible for the agent to see anything. The agent remembers that the exit door is situated close to a circular wall at the other end of the room. It decides to find the circular wall to get closer to the exit door. The agent then walks with its arms extended and feels the curved walls.

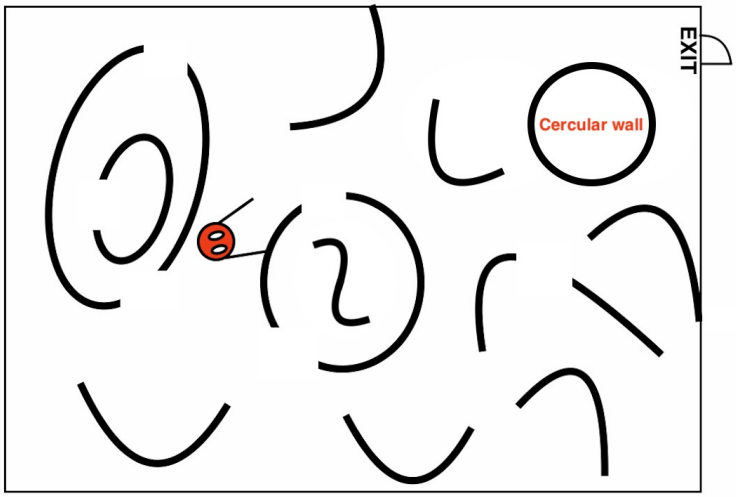

Figure 3. A stage for sensorimotor perception. Source: Regenerated image from Sensorimotor life [15]

Now the main question is how the agent would be able to tell the difference between cylinder wall from other curved walls? Only the contact with the wall is insufficient by itself to help it understand the shape of the wall, because the local curvature of the wall is not enough to tell the difference between cylinder wall and curved wall. But as the agent continues to walk while feeling the curvature of the wall, maybe it observes that

for a while it has turned in the same direction. In this case the agent has used bodily sensations such as kinesthetic and proprioceptive cues, generated by some regularities in its own wall-guided walking. The knowledge is obtained from a long pattern of sensorimotor activity. By bodily sensations it becomes certain that it had found the cylinder wall and guides itself to the exit door next to it.

There are different kinds of sensorimotor regularities. One kind of sensorimotor regularity is primarily influenced by the relationship between the agent's bodily feature and environment. Since the walls are fixed in the room, at different given spots, a displacement of one step produces a change in tactile sensation particular to that spot. This is one of the general sensorimotor regularities because it simply depends on the relationship between bodily and environmental structures. Sensorimotor activity operates in a circular manner, forming a closed loop. It is hard to tell which one occurs first, the movement that changes the experience, or the sensation that changes the action [15].

2.2.2. A Minimal Model

In Sensorimotor Life [15] Ezequiel Di Paolo and Thomas Buhrmann discuss the concept of tactile discrimination and other various notions that are related to it. They described tactile discrimination as a minimal model for understanding the deep relation between sensorimotor environment, habitat, coordination and scheme. Regardless of their close relations, these notions can also be used separately for different purposes. The authors explain how psychology, neuroscience, and robotics have already made use of these notions. They also draw attention to the potential of designing robots that can interact with their environment in an adaptive and intelligent manner.

Sensorimotor task scenarios have been presented, in which an agent needs to solve problems using only tactile perception. Such a task could involve finding a light switch from a set of switches on a wall or arranging coins blindfoldly according to their thickness order with the tip of only one finger. The authors argue that tactile discrimination can be challenging even for solving these simple tasks, and explain the necessity of an active categorical perception.

To demonstrate the active perception of categories, they proposed an agent-based model where the agent's movement is controlled by a simple neural system. The agent moves around in a one-dimensional sensorimotor environment that contains two bell-shaped objects of different diameters. The objective of the agent is to discriminate between the objects' shapes by using simple tactile perception while moving away from a wide bell-shaped object toward a narrow bell-shaped object or other way around. The agent senses the change of objects' shapes through input and feeds it to a small dynamical neural network which then continuously sends out motor commands for managing the agent's velocity. The agent learns to discriminate through trial and error, and it gets a score depending on the success rate of completing this objective.

2.3. Modeling Cognition

There are various different approaches when attempting to build a model for cognition. In classical cognitive theory the brain is often thought of as an information-processing device; the brain receives input from the external environment as stimuli and builds an internal representation of it. The brain can manipulate the internal model through computational procedures, and it can produce an output (action) [3]. This suggests that the truly cognitive processes happen during the computational processes in between perception and action systems [16].

A common model is the emulation theory of representation [1], which emphasizes the computational processes in the brain that produce and consume representations [15]. It is based on the concept of *efference copy* in the context of motor control, introduced in the early 19th century [2]. The emulation theory of representation is a framework that can synthesize a wide variety of representational functions of the brain. It is based on constructs from control theory and signal processing. According to the emulation theory of representation, the brain constructs neural circuits that act as models of the body and environment. During sensorimotor engagement, these models are driven by efference copies in parallel with the body and environment. The models try to predict the sensory feedback of an action, and the prediction error is used to enhance sensory information [1].

The enactive approach disagrees with the idea presented in traditional cognitive theories that perception consists in mostly passive transduction of external stimuli into neuronal representations. Instead, in enactivism it is argued that perceptual processing should be conceptualized as an interactive framework, in which sensory and motor processes work in a closed action-perception loop [3]. Enactivism also criticizes the notion of serial information processing, according to which perception, decision making, and action planning happen in seperate stages. This does not comply with the enactive idea of closed action-perception loop [3]. According to the enactive view of cognition, the primary role of the brain is guiding interaction with the environment, rather than representing or understanding the world [3, 5, 4]. It emphasizes closed perception-action loops and a tight coupling between agents and environments [3].

2.3.1. The Symbol Grounding Problem

For creating autonomous and intelligent agents that can act in the real world, it is important to establish and maintain a connection between what the agent reasons about and what it can sense in the real world [17]. This can be considered as an aspect of the symbol grounding problem [17]. The symbol grounding problem as defined in [18] asks: how to ground the meanings of symbols as anything different than other symbols. In [18] a "Chinese to Chinese dictionary" example is given to help understand the problem. Imagine one had to learn Chinese as a second language using only a Chinese to Chinese dictionary. The symbols in this example are the words of the dictionary. A word in the dictionary is explained using other words, and one would pass endlessly from one word to another, never understanding what any of it meant. In this case a word would not have any intrinsic meaning.

This is relevant when thinking about how to represent symbols in robotics and intelligent systems, for example an automated vacuum or facial recognition programs. However, there is no need for the symbols to have intrinsic meaning, if an intelligent system is able to perform tasks [18]. Regardless, endeavouring to ground symbol systems could lead to performance gains in intelligent systems, because our own symbols do have intrinsic meanings while those of computers do not, and we can perform tasks which the computer so far cannot [18].

Physical symbol grounding is a subtopic of the symbol grounding problem. It is defined in [19] as the grounding of symbols to real world objects by a physical agent interacting in the real world [18]. One of the basic challenges in physical symbol grounding is to ground symbols to sensory data where the symbols denote categorial concepts such as color, shape and spatial features [18]. There are multiple different apporaches to this challenge [18, 20, 21, 22, 23]. For example, in [21] basic words are grounded into an agent's own categorial representations via sensorimotor experience. In [22, 23] an agent ties symbols to their sensorimotor instantiations and at the same time unties sensorimotor representations from their physical specificities, correlating them to abstract symbolic structures.

Symbol grounding is important when integrating robots and distributed systems in dynamic and unstructured environments. Such environments require the agent an understanding of the connection of symbolic and sensory information to be able to successfully operate [18]. Steps towards the solution of the symbol grounding problem will be key to creating robotic systems that are capable of high level reasoning [18].

2.3.2. Mathematical Models of Enactivist Cognition

Transition systems are very general mathematical structures that can be used as a framework for mathematical models of enactivist models of cognition. They can be used to describe an agent's potential behavior. A transition system consists of states and the transitions between the states (see Definition 1).

Definition 1 *A transition system is a triple (X, U, T) where X is the state space (mathematically it is just a set), U is the set of names for outgoing transitions (another set), and* $T \subseteq X \times U \times X$ *is a ternary relation. [10]*

In [24], transition systems appear in the form of information spaces that contain a set of all possible transformations that could be applied to the agent [24]. Information spaces can be used in solving tasks in robotics by manipulating or simplifying the information space [24]. In the thesis, we focus on solving robotic tasks with a minimal information approach, where the necessary brain structures emerge from sensorimotor interaction and do not necessarily have predetermined components. Without predetermined components, the robot does not initially have information on its current state. The robot can get information regarding its state from sensor observations, and from the actions that the robot has performed [24].

Transition systems can also be used to model the robot-environment interaction [25]. In [25], an example is given on how this interaction can be established using two transition systems. An external system is used to model the physical world, which

matches the environment and the robot body in it. An internal system is used to model the robot's brain. It observes the external system through a sensor mapping and interacts with it through a selection of actions. These two systems form a coupled dynamical system, where the external system produces a sensory observation, and the internal system processes the sensory observation in order to select a new action. Then the external system processes the action and gives a new sensory observation. The two systems form a loop as seen in Figure 4. The key in this theory is to focus on necessary and sufficient conditions that a robot's brain must maintain to solve the required task [25]. By minimal sufficient conditions we mean the minimal amount of sensing, actuation and computing needed for the task. If anything is removed from a minimal sufficient system, the task will become unsolvable [25].

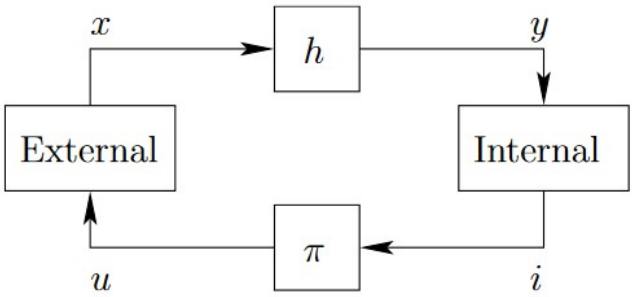

Figure 4. The internal and external systems form a loop, in which the internal system uses the external system's output as an input, and the external system uses the internal system's output as an input. Source: [10].

The mathematical framework presented in [10] as a model of cognition is intended to be in line with the five enactivist tenets presented in Section 2.1.1. For constructing the framework transition systems, more specifically sensorimotor systems, have been utilized. A sensorimotor system is a transition system which includes a sensory set and a motor set (see Definition 2).

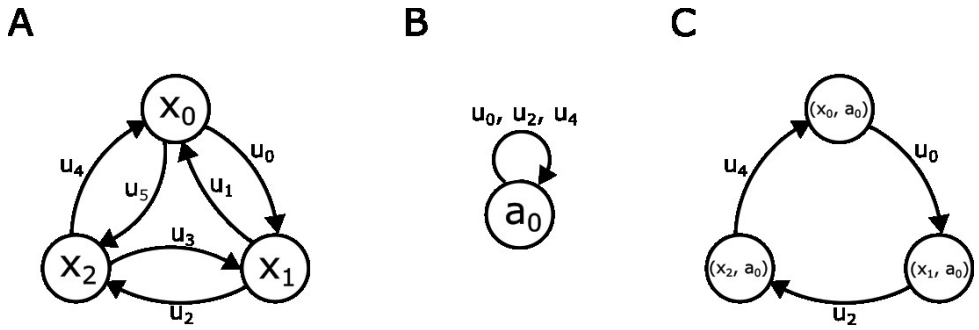

Figure 5. **(A)** Transition system X. **(B)** Transition system A. **(C)** The coupled system $X * A$.

Definition 2 *A sensorimotor system (or SM-system) is a transition system (X, U, T) where $U = S \times M$ for some sets S and M, which we call in this context the sensory set and the motor set, respectively. [10]*

Sensorimotor systems can be connected together to form coupled systems in order to model any part of the environment-body-brain coupling. Numerous simulation studies have shown that coupling an agent with the environment or another agent can produce an interesting pattern of behaviour, which can solve tasks [3, 26, 27, 28, 29]. For example, if one transition system models the environment, and another transition system models the agent, then the coupling of these two transition systems tells us about all the possible ways in which the agent can engage with the environment [10]. The coupled system contains information on what would happen in any given environment state while the agent is in any of its states. See Figure 5 for a demonstration on how two transition systems can be coupled.

2.4. Classical Automata Learning

An automaton is an abstract self-acting computing device which follows a predetermined sequence of operations automatically. An automaton with a finite number of states is called a Finite-State Machine (FSM). An FSM consists of states and transitions as seen in Figure 6. The automaton makes transitions between states according to its transition function, which takes the previous state and current input symbol as its arguments.

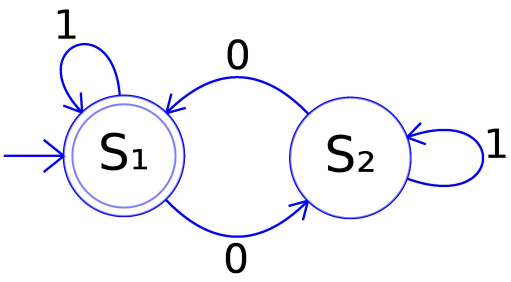

Figure 6. A finite-state machine with two states (S_1 and S_2) and two types of inputs (0 and 1) [30].

Generally, deterministic automata are easier to learn than nondeterministic automata [31]. This is because in deterministic automaton, a state can only have a single arrow of the same label, whereas in nondeterministic automata, a state can have multiple arrows of the same label, which can lead to the same input giving varying results. Therefore inferring a nondeterministic automaton is more complicated, than inferring deterministic automaton.

In computational complexity theory, decision problems are categorized into complexity classes. The complexity classes describe the time it takes for a computer to solve the problem. The complexity class P consists of problems that are solvable deterministically in polynomial time. Complexity class NP (nondeterministic polynomial time) is the set of decision problems that can be solved in polynomial time by a nondeterministic Turing machine, or the set of problems whose solution can be verified in polynomial time by a deterministic Turing machine [32]. The hardest problems in the NP class are called NP-complete problems. An algorithm capable of

solving an NP-complete problem in polynomial time would also be able to solve any other NP problem in polynomial time. The existence of such an algorithm would prove that the sets P and NP are in fact the same. This P versus NP problem is a well-known unsolved problem in computer science. It is widely believed that such algorithms do not exist [33].

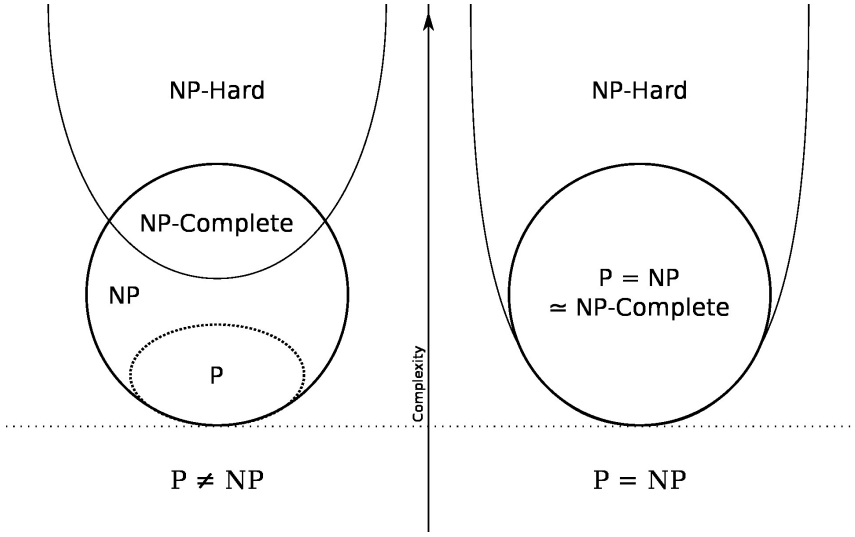

Figure 7. Euler diagram of P, NP, NP-complete and NP-hard set of problems [34].

An NP-hard problem is a problem that is at least as hard as the hardest problems in NP, but it might not be in NP. For example, the subset sum problem is NP-hard [32]. The task in the subset sum problem is to check, for a given set of non-negative integers and a value sum, whether there exists a subset of the given set whose sum is equal to the value sum.

The task of finding the smallest automaton consistent with a given sample of input/output pairs is investigated in several papers and it is shown to be NP-complete [31, 35, 36]. For example, in [36] E. Mark Gold calls this task the minimum automaton identification problem, and he defines it as the constructing of a finite automaton with the minimum number of states which agrees with the given data. Gold proves that minimum automaton identification problem is NP-hard and NP-complete [36]. Our robotic arm does minimum automaton identification, when it builds a representation of the environment. It uses input/output pairs (sensory feedback/action pairs) as data, and uses it to build an automaton consistent with the data. This suggests that our problem could also be NP-hard.

The NP-Hardness results depend on a specific restriction placed on the learning algorithm, requiring the learning algorithm to use a particular representation of the automaton [31].

Kearn and Valiant explore the limitations of learning Boolean formulae and finite automata [37]. Boolean formulae are constructed from variables, Boolean operators and parenthesis [38]. In this paper they discuss the representation-free problem [37]. A representation-free hardness result means that learning is challenging regardless of how a learning algorithm represents its hypothesis, as long as the hypothesis can be evaluated in polynomial time [37].

The main goal of the representation-free problem is to investigate how difficult it is to predict the output of an unknown automaton, without any previous knowledge of the automaton's internal representation or structure. Traditional automata learning determines the output of underlying automaton based on collection of observed input and output samples. In our project, the agent attempts to learn a minimal model of an unknown environment from finite action-observation sequences.

3. ROBOTIC SENSORIMOTOR SYSTEM TESTING PLATFORM

Our project is based on the software project Robotic Sensorimotor System Testing Platform (RSSTP) which is a platform for testing algorithms. It is inspired by the OpenAI Gym which is an open source Python library for developing and comparing reinforcement learning algorithms. It provides a standard API to communicate between learning algorithms and environments. The RSSTP can be used by software developers to develop and test new algorithms, which can help a robot to learn about its environment with minimal sensor feedback. In this section we describe the functionality of the testing platform prior to the implementation of the additional features that we designed in the project.

The RSSTP simulates a multi-jointed robotic arm which moves in a two-dimensional space. The goal is for the arm to build a model of its environment by moving around in the space. The robotic arm does not get any sensory feedback, except in a single point in the space. We call this the sensory feedback point. This point is predetermined by the user, and could for example be the arm's starting point. With the help of the sensory feedback point, the arm can try to deduce whether there are obstacles in the space or not, and it will remember the obstacles' precise locations.

For example, imagine a scenario where a robotic arm with a single joint starts at the sensory feedback point. The arm then moves to the right three times. Then the arm starts moving back to the left, but only after two movements the arm gets sensory feedback, meaning that it is back at the starting position. How is this possible? There must be an obstacle to the right of the starting point, and the arm hit the obstacle after two movements to the right. When the arm tried to move to the right for the third time, the obstacle prevented it from doing so. But because the arm does not have any other sensors, it does not know that it hit a wall until it returns to the starting point, and notices the inconsistency between the series of actions it took.

3.1. Software Design

The software is split into two parts - the internal and the external - and there are three classes: the *internal*, the *external*, and the *msrgym* class. The *internal* and the *external* classes of the software work independently from each other. The *msrgym* works as an interface for the user and it combines the behaviour of both the *internal* and the *external*.

3.1.1. The Internal System

The internal part of the software is the *brain* of the robotic arm. You can think of it as a *brain in a jar*, which does not know anything about the external world.

The internal system builds a model of the environment by using a transition matrix. The transition matrix is a type of a transition system (see Definition 1), and it consists of states, which represent positions in the external world (see Figure 8). The size of the matrix will be $n \times n$, where n is the number of states. The transition matrix

contains information about the actions that allow the robot to move between states. These actions can be for example *"move joint 2 to the left"*.

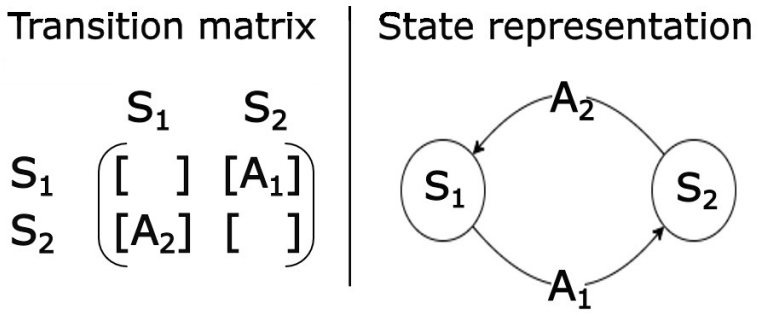

Figure 8. Comparison between the transition matrix and the state representation.

As the robotic arm gains more information of the environment, the internal system manipulates the transition matrix accordingly. There are four methods which can be used to modify the transition matrix. For example, transitions can be added or deleted between nodes (nodes = internal states), a node can be split into two nodes, or two nodes can be merged into one.

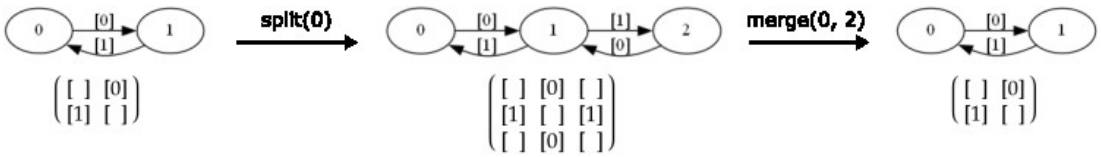

Figure 9. A demonstration of the split and merge methods. First, node 0 is split, and after that nodes 0 and 2 are merged.

Figure 9 shows a demonstration of splitting and merging a node. Splitting a node creates a new node, which retains all the attributes of the original node, such as the transitions to other nodes. Merge is the opposite of split. It takes two nodes and turns them into one node. The resulting node will have the combined transitions of the two original nodes.

3.1.2. The External System

The external part of the software sets up the environment for the robotic arm. The environment is a two-dimensional plane that can contain circular obstacles, sensory feedback point, and the multi-jointed robotic arm. The environment and the movements of the arm are visualized to the user as seen in Figure 10.

The external part also contains logic, which defines all the possible movements for the arm in a certain environment. It ensures that the movements of the robotic arm are allowed, meaning that the arm cannot go through obstacles, and it cannot intersect with itself.

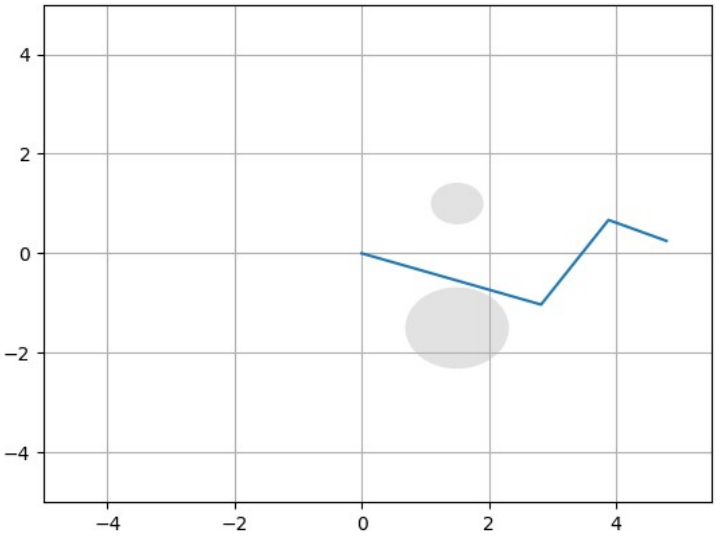

Figure 10. The visualization of the robotic arm and its environment. The blue line represents the multi-jointed robotic arm. They grey circles represent obstacles in the space.

3.2. Aims of the Project

We are looking to explore different ways to improve the performance of the platform. One of the challenges which was encountered during testing was that the algorithms become slow, making it difficult for the arm to make progress. The robotic arm did not have memory, so optimizing the algorithm was difficult. The platform attempts to maintain a record of every state of the transition matrix, and when the robotic arm has more than one joint, it becomes increasingly complex. Quickly the transition matrix becomes too large to keep track of and it slows down the algorithm. In this work we develop new functionalities to the platform that help the robotic arm to make progress faster. These functionalities can be used by algorithm developers and other users as tools to help them build and evaluate new algorithms.

4. IMPLEMENTATION

In this section we will take an in-depth look at all the new functionalities we have implemented and the reasoning behind them.

4.1. Implementing Memory

For the robotic arm to be able to learn through actions, it first needs to be able to remember the actions performed in the past. For that reason, implementing memory was necessary. With the help of the memory, the robotic arm (and the user) can review the actions performed to move from one state to another and attempt to replicate them. If the replication fails, then the robotic arm can deduce that its understanding of the environment is not complete. The memory also helps the user to understand the decision making of the robotic arm, because the user can go through the memory to analyze the behavior of the arm.

We implemented the memory of the robotic arm as a nested list in Python. The memory consists of steps, and every time the robotic arm moves, a new step will be added to the memory. The memory saves three different values: the previous action, the internal state, and the sensory feedback. The previous action is the most recently performed action by the arm. The internal state is the state of the transition matrix, in which the arm is in at that particular moment. The sensory feedback is either true or false, depending on whether the arm is in its sensory feedback point or not.

4.2. Comparing Submemories

We implemented a compare method that compares two submemories to each other. This method can be used to determine whether the memory is consistent or not. For example, the same set of actions from the same starting point should always lead to the same outcome. If the outcome is different, that means that the memory is inconsistent, and the transition matrix needs to be redefined.

The memory consists of steps, and each step contains information about the robotic arm at that moment, including sensory feedback and previous action. These steps are indexed by non-negative integers. For example, for a memory that is six steps long, we can choose to compare the submemories of the memory from indices two and zero. Figure 11 shows an example of using the compare method. The submemory with the higher starting index determines the length of the submemories. In this case, we take the memory from index two onwards until the most recent step, indexed at five. We get two submemories with the length of four steps each. First we compare the starting points. Both of these submemories start from a state where they do not get sensory feedback. They both then use the same action, which leads them to a state where they do get sensory feedback. Again, they both use the same action, but we notice that they end up in different states, because one of them (submemory 1) gets sensory feedback, and the other one (submemory 2) does not. We can then deduce that the model is not ready and it needs further refinement.

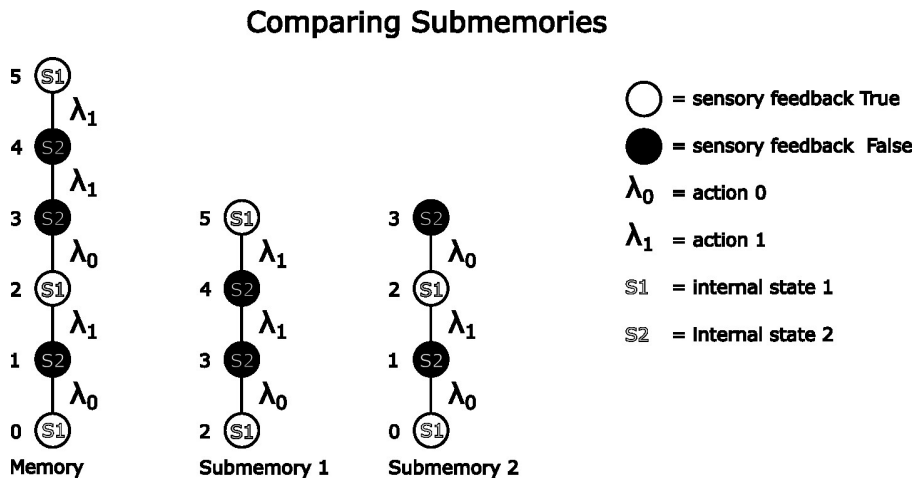

Figure 11. An example of using the compare method. Here we compare two submemories which start from indices 2 and 0.

Note that even if the two submemories are fully consistent, this does not mean that the model is ready. This method does not prove whether a model is ready, but it is rather used for proving that the model is not ready, and more adjustments needs to be made.

4.3. Checking Determinism

We have implemented a function that determines whether the transition matrix of the robotic arm defines a deterministic transition system. Our primary objective is to ensure that the transition system is deterministic. If the transition system is nondeterministic, it is possible that the action taken by the robotic arm could lead it to multiple different states. We do not want this to occur, but rather we want a deterministic transition system so that regardless of the circumstances the robotic arm should always switch to the same state by the same action.

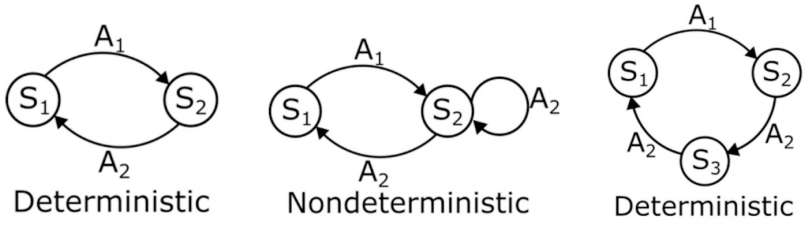

Figure 12. Examples of deterministic and nondeterministic transition systems.

If the transition system is nondeterministic, it indicates that the robotic arm does not have enough states or it did not perceive the environment sufficiently. In this case it needs to create more states until the transition system becomes deterministic.

4.4. Visualizing the Transition Matrix

In the original code, the transition matrix was represented as a list of lists (Figure 13). Each row represents an internal state and its actions for transitioning to other states. For example in Figure 13, action 1 needs to occur to get from state 0 to state 1 (represented in row 1 column 2 in Figure 13). These types of representations of small transition matrices can be easily understood, but for bigger matrices it can get more troublesome. We decided to add a new visualization feature of the transition matrix by using visualizing graphs since the original visualization method was challenging to read.

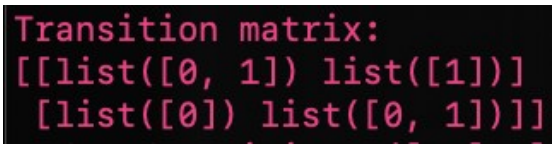

Figure 13. Transition matrix visualized as a print in command prompt.

The visualization of the state machine allows the user to better understand the relationship between states and actions. It also facilitates the analysis of error cases, such as a transition leading to an unexpected state. When an error occurs we will be able to pinpoint the error and get to the core to find its cause easily by checking the step-by-step progression of state transitions. In case of small 2×2 or 3×3 transition matrices, the usefulness of the graph visualization functionality is limited. But when the size of the transition matrix exceeds 3×3, it will take longer to figure out the connections between different states and actions.

4.4.1. The First Approach: NetworkX

For graph visualization in our initial approach, we chose a Python-based visualization tool called `NetworkX`. On the graph each node stands for a distinct state, and each edge represents an action that leads to another state (Figure 14). At first, we thought using `NetworkX` was an ideal choice. However, soon we discovered that the graph had significant flaws.

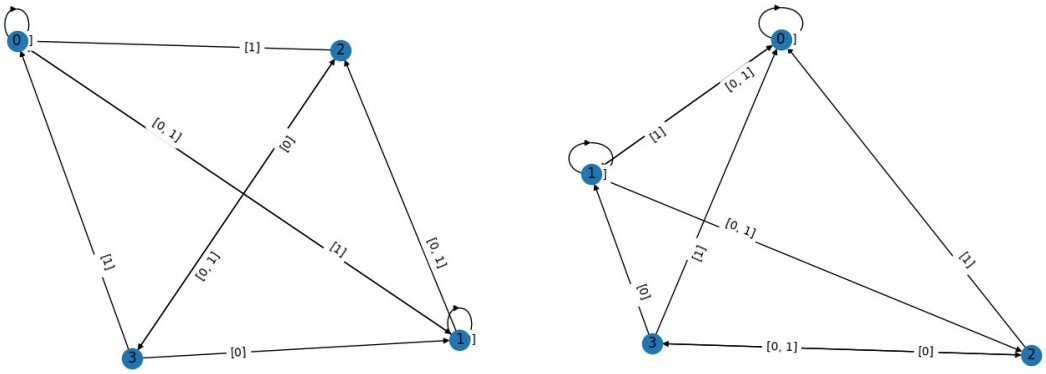

Figure 14. A transition matrix visualized twice using NetworkX.

Key flaws that appeared in our `NetworkX` generated graphs:

1. Overlapping occurred between directional arrowheads and vertex labels. Occasionally labels got partially covered by vertices, which caused difficulties to distinguish which label was assigned to which vertex. In some cases, two directional edges were totally unrecognizable due to the label overlapping issue.

2. Each newly generated graph appeared to be drawn into a totally different coordinates of the plot, even though the transition matrix was identical to the previous one. This inconsistency of visual representation made it challenging to find similarities between matrices, especially when attempting to compare many iterations of the same matrix.

4.4.2. The Second Approach: Graphiz

Because of `NetworkX's` disadvantages, we started looking for alternatives graph visualization tool to replace it. After long consideration and reading through different graph visualisation tools' documentations we came to the conclusion that `Graphviz` would be the solution to our problem. The implementation of `Graphviz` was easy and straightforward. The graph generated by `Graphviz` exceeded our expectations. High-quality graphs were generated in real-time while simultaneously creating image files of the graphs to our directory. Newly generated graphs appeared much cleaner and intuitive, resolving all the issues and imperfections we had encountered with `NetworkX`.

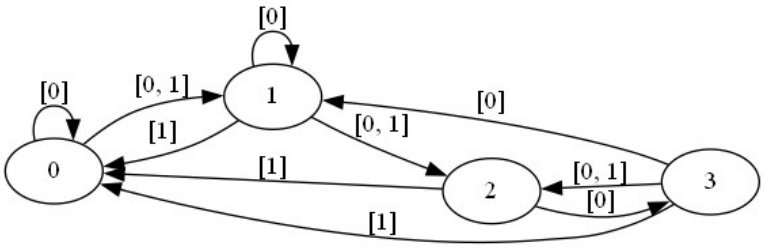

Figure 15. A transition matrix visualized using Graphviz.

$$\begin{bmatrix} [0] & [0,1] & [\,] & [\,] \\ [0] & [0] & [0,1] & [\,] \\ [1] & [\,] & [\,] & [0] \\ [1] & [0] & [0,1] & [\,] \end{bmatrix}$$

Figure 16. A transition matrix equivalent to the graph in 15

4.5. Implementing the Obstacle Sensor

To enhance the robotic arm's understanding of its environment, we implemented an obstacle sensor functionality into the software. This functionality not only increases the robotic arm's awareness of its surroundings, but also gives the user more understanding of how decisions have been made by the robotic arm.

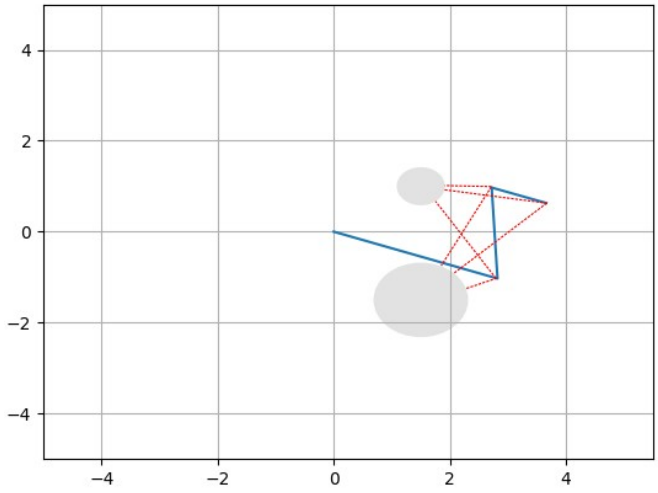

Figure 17. The obstacle sensor calculates the lengths of the red dotted lines.

Our obstacle sensor calculates the distance between an arm tip and all obstacles within the environment, for each of the joints of the arm (see Figure 17). To get a precise distance calculation we obtained the coordinates of the arm tip and measured the distance from the center point of each obstacle, while subtracting the obstacle radius from the calculation. The real time distance measurement is available to the user after every movement step that has been taken by the robotic arm.

4.6. Option to Change the Output of Sensory Feedback to Float

Previously, the sensory feedback provided a sensation that was either true or false, based on whether the robotic arm was at its designated sensory feedback point or not. While we are mainly interested in developing algorithms with minimal sensory feedback, such as a boolean value, we decided to add an option to change the sensory feedback to a floating-point number. This value can be used to help the robotic arm progress faster in its tasks. The float value describes how close the robotic arm is to its sensory feedback point on a scale from zero to one. Zero means that the arm is as far away from the sensory feedback point as possible, and one means that the arm is in the sensory feedback point. The user of the platform has the option to choose between a boolean value or a float value, depending on their preferences.

5. EVALUATION

5.1. Evaluation Plan

To test the software, we use unit tests individually for every method that we have implemented, as well as for the memory of the robotic arm. We create the unit tests manually and feed the tests different kinds of inputs, and make sure that the outputs are correct. We focus on not only the output, but also how the output is achieved.

We also develop an algorithm that uses the platform. This allows us to ensure that the software components work together as intended. The goal for the algorithm is to build an accurate representation of the environment using a transition matrix. The algorithm will use the methods and the memory that we have implemented.

5.2. Unit Tests

For evaluating our software we used unit testing. These test routines allowed us to confirm that each piece of the software works, when used independently from the rest of the system.

We made a total of 18 test functions which test eight of our implemented methods. We used different inputs for the test functions and made sure that the outputs were as expected. For example, when testing the compare method (see **??**), we used memories consisting of different sequences of actions and sensations as an input, and checked that the results of the submemory comparisons were correct. Unit testing helped us to find errors in our code, which we were able to fix after further debugging. Ultimately, we ensured that everything works as expected.

5.3. Algorithm

We developed an algorithm to test software functionalities with a transition matrix and limited number of states. In developing of this algorithm, we prioritized efficiency and quick computation. The algorithm has enabled us to evaluate the performance of the implemented functionalities and whether all the functions work in harmony. This has helped the robotic arm to make progress in building comprehensive transition matrix by using memory and gave it better understanding of its environment. It also helped us to evaluate the transition matrix and graph visualizations.

The algorithm we built works in a very simple environment. We chose to use a simple environment, because our main focus was to test the functionality of the platform, and we did not want it to be too complicated. The algorithm works in a simple setting, where the environment has no obstacles, the arm rotates 90 degrees at a time, and the arm only has one action; move to the right. That means, that there are only four possible positions for the arm.

The goal for the algorithm is to build a transition matrix, that accurately represents the environment. We accomplish this with the help of the (binary) sensory feedback and the memory. We also utilize the compare method which we implemented. Ideally the resulting transition matrix would look like the one in Figure 18. It would have

a total of four internal states, where each state represents a position of the arm. The transitions between the states would be deterministic, and the states would construct a loop, as seen in Figure 18.

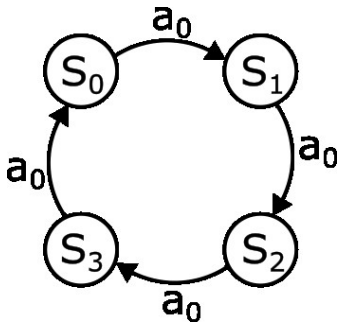

Figure 18. The ideal transition matrix for the simple setting that we described.

Initially, the transition matrix only has one internal state (see Figure 19). The robotic arm starts moving, and after every move, it uses the compare method to check for any inconsistencies in its memory. When the arm reaches the sensory feedback point, it will notice a change in sensory feedback. The arm can then deduce that it must be in a new position, and therefore a new internal state must be created. The arm splits the initial internal state into two states and manipulates the transitions between the two states accordingly. If the arm performed multiple moves before reaching the sensory feedback point, it can then deduce that there must be more than two states. For example, if in the arm's memory there are multiple instances of internal state 0, and from one instance of state 0 it takes only one move to reach state 1, but from another instance of state 0 it takes 2 moves to reach state 1. That means that the two instances of state 0 must actually be two different positions. Therefore the arm splits the state 0 for a second time creating a third state. The arm repeats calling the compare method and adding new states, until no more inconsistencies are found in the memory. After the arm is done checking the memory, it can keep moving.

The implementation of the algorithm has been successful. When we tested our platform through the algorithm, we did not encounter any crashes or software glitches. Every function worked as we expected. In addition, no mistakes were found in transition matrix and in graph visualization.

However, despite successful testing, we have identified a new problem with the algorithm. It does not have the ability to recognize unique positions of the robotic arm while perceiving the environment. As a result, it continues to generate new states even when the entire environment had been perceived, and it fails to recognize if it has previously created an identical state. Therefore, the algorithm keeps on adding new states until the user-defined movement number has been reached.

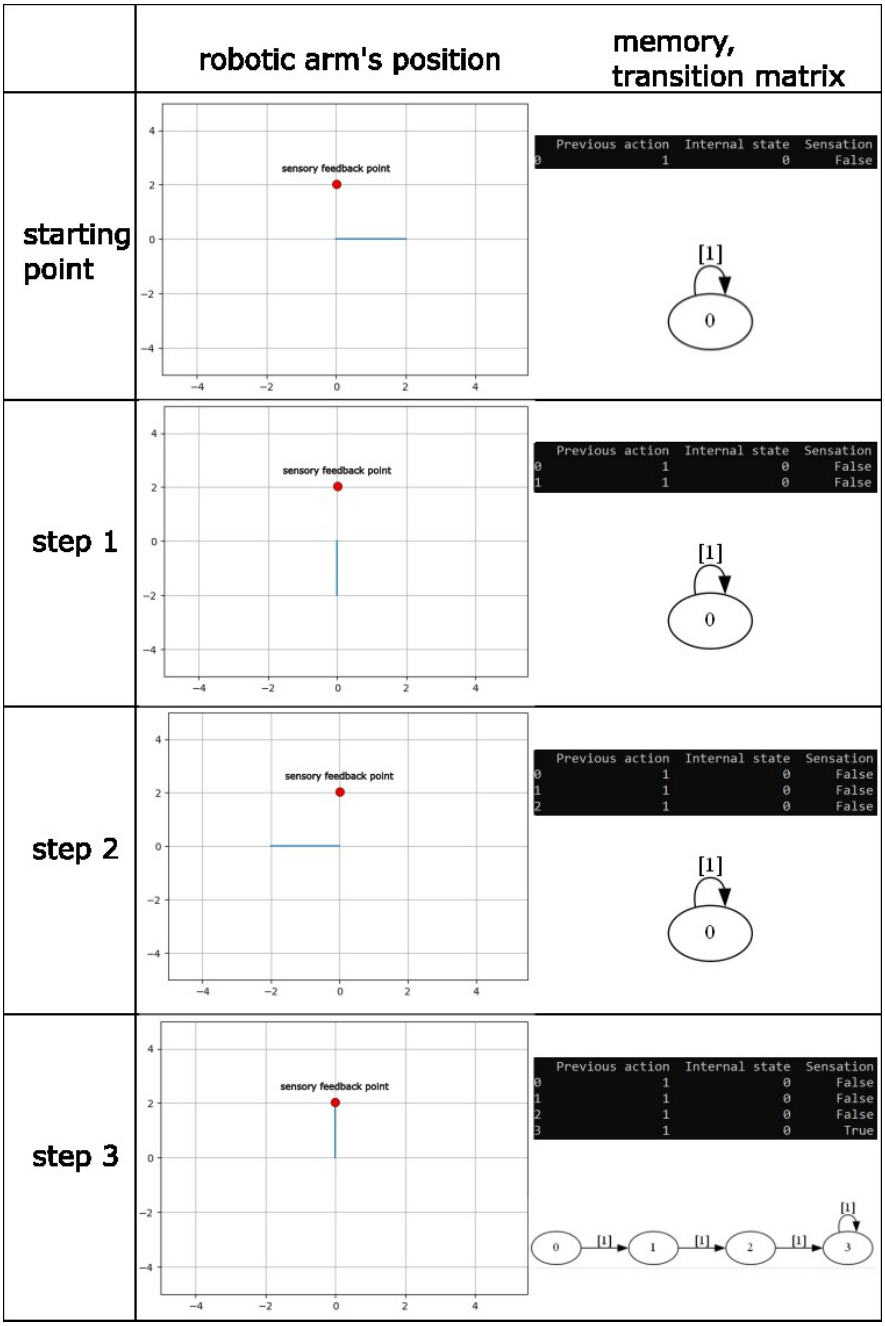

Figure 19. Demonstration of the algorithm's progression.

6. DISCUSSION

In this project we enhanced the funcionality and usability of the RSSTP. By using this platform, we demonstrated that learning is possible with minimal sensory feedback, where an agent builds an internal model of the external environment with the help of memory and a sensory feedback point. The new implementations made it noticeably easier for the agent to make progress in its quest to explore the environment compared to the previous iteration of the RSSTP where the agent had no memory. However, by adding new funcionalities such as an obstacle sensor, we have drifted away from the minimal information approach. Nevertheless, the RSSTP can be used for exploring and finding the minimal sufficient requirements for solving a task, but it can also be used in more broad sense where the agent uses "extra" or possibly "unnecessary" information such as the obstacle sensor.

There are several other platforms created for testing algorithms in robotics. For example, OpenAI Gym is a standard application programming interface (API) for reinforcement learning. OpenAI Gym works as a general framework in which any type of reinforcement learning task can be defined [39]. Usually it is modeled with two components: the agent and the environment, and the goal is to discover which actions best achieve the task. The process of evaluating an action is done using reward signal as an indicator to determine if the action taken gets the agent closer or farther away from the goal of the task [39].

Another example is FRobs_RL [39], which is a Python library that aims to faciliate the implementation, testing and deployment of reinforcement learning algorithms in intelligent robotic applications using robot operating system Gazebo and OpenAI Gym. Its main goal is to reduce the time it takes to transfer reinforcement algorithms from a simulation into intelligent robotics applications and hardware. Using FRobs_RL one can transfer a model which is trained through simulation directly into an actual robot with minimal effort [39].

RSSTP focuses more on the theoretical side of learning problems in robotics. To our knowledge RSSTP is the first algorithm testing platform in which the agent follows the enactive view of cognition where sensorimotor interaction with the environment is key in bringing intelligence to the agent. RSSTP uses transition systems to model the brain of the agent, and instead of using reward signals, it uses minimal sensory feedback.

In the future of the RSSTP, the robotic arm can be improved to fit various environments. We are interested in what would happen, if the environment is more complex than our testing environment. Would the arm still be able to construct an accurate transition matrix? For example, if the environment would have multiple obstacles, and the robotic arm would have multiple joints and move in smaller steps (degrees) resulting in a bigger transition matrix.

In the future new functionalities could be added to the RSSTP, for example constructing a *minimal model* of the environment as opposed to our ever growing transition matrix. For example, the agent could notice whether it is going in a loop through pattern recognition, and then minimalize the transition matrix accordingly. This could be implemented as a functionality in the RSSTP, or it can be left for the algorithm developers to include this logic in their algorithms.

Further on, the RSSTP and the idea of learning with minimal sensory feedback can be expanded from the robotic arm into other agents and problems, for example, a roaming robot exploring its environment.

6.1. Conclusion

In sum, we discussed different philosophical views to cognition in cognitive sciences. We focused more closely on the enactive perspective of cognition, and how it can be modeled mathematically using transition systems. We discussed how this philosophical point of view can be brought into robotics, and what the benefits of doing so are. We then introduced the RSSTP along with our newly implemented functionalities, and explained the reasoning of the implementations. We evaluated our implementations through unit testing and with an algorithm, which demonstrates the use of the platform. With the algorithm we also demonstrate how an agent can learn about its environment with minimal sensory feedback. We reached the initial target setting of making the RSSTP more useful with more functionalities, where it is easier for the robotic arm to make progress. While the RSSTP is one of many algorithm learning platforms, it is unique and has its own place in the field. In the future the RSSTP can be developed further by adding new functionalities, and it can be used to explore theoretic philosophical questions regarding sensorimotor interaction, cognition of an agent, and problem solving in robotics.

7. REFERENCES

[1] Grush R. (2004) The emulation theory of representation: Motor control, imagery, and perception. Behavioral and Brain Sciences 27, pp. 377–396. URL: https://doi.org/10.1017/s0140525x04000093.

[2] Steinbuch J.G. (1811) Beytrag Zur Physiologie Der Sinne. Schrag, Nürnberg, Germany.

[3] Pezzulo G., Donnarumma F., Iodice P., Maisto D. & Stoianov I. (2017) Model-based approaches to active perception and control. Entropy 19, p. 266. URL: https://doi.org/10.3390/e19060266.

[4] Pezzulo G. & Cisek P. (2016) Navigating the affordance landscape: Feedback control as a process model of behavior and cognition. Trends in Cognitive Sciences 20, pp. 414–424. URL: https://doi.org/10.1016/j.tics.2016.03.013.

[5] Engel A.K., Maye A., Kurthen M. & König P. (2013) Where's the action? the pragmatic turn in cognitive science. Trends in Cognitive Sciences 17, pp. 202–209. URL: https://doi.org/10.1016/j.tics.2013.03.006.

[6] Georgiev F., Chakal K., Boulfrad M., Fernando R. & Hasan S. (2022) Robotic sensorimotor system testing platform (RSSTP). Tech. rep., Oulun Yliopisto, Oulu, Finland.

[7] Thompson E. (2007) Mind in Life. The Belknap Press Of Harvard University Press, London, England.

[8] LeDoux J. (2002) The emotional brain revisited. In The synaptic self. NY: Penguin Group, New York, 200-234 p.

[9] A visual guide to react mental models. https://obedparla.com/code/a-visual-guide-to-react-mental-models/. Accessed: 2023-05-18.

[10] Weinstein V., Sakcak B. & LaValle S.M. (2022) An enactivist-inspired mathematical model of cognition. Frontiers in Neurorobotics 16. URL: https://doi.org/10.3389/fnbot.2022.846982.

[11] Gallagher S. (2017) Enactivist Interventions. Oxford University Press. URL: https://doi.org/10.1093/oso/9780198794325.001.0001.

[12] Hutto D.D. & Myin E. (2012) Radicalizing Enactivism. The MIT Press. URL: https://doi.org/10.7551/mitpress/9780262018548.001.0001.

[13] Varela F., Rosch E. & Thompson E. (1992) The Embodied Mind: Cognitive Science and Human Experience. The MIT Press, Cambridge, Massachusetts.

[14] O'Regan J.K. & Noë A. (2001) A sensorimotor account of vision and visual consciousness. Behavioral and Brain Sciences 24, pp. 939–973. URL: https://doi.org/10.1017/s0140525x01000115.

[15] Paolo E.D., Buhrmann T. & Barandiaran X. (2017) Sensorimotor Life. Oxford University Press. URL: https://doi.org/10.1093/acprof:oso/9780198786849.001.0001.

[16] Hurley S. (2008) The shared circuits model (SCM): How control, mirroring, and simulation can enable imitation, deliberation, and mindreading. Behavioral and Brain Sciences 31, pp. 1–22. URL: https://doi.org/10.1017/s0140525x07003123.

[17] Coradeschi S., Loutfi A. & Wrede B. (2013) A short review of symbol grounding in robotic and intelligent systems. KI - Künstliche Intelligenz 27, pp. 129–136. URL: https://doi.org/10.1007/s13218-013-0247-2.

[18] Harnad S. (1990) The symbol grounding problem. Physica D: Nonlinear Phenomena 42, pp. 335–346. URL: https://doi.org/10.1016/0167-2789(90)90087-6.

[19] Vogt P. (2002) The physical symbol grounding problem. Cognitive Systems Research 3, pp. 429–457. URL: https://doi.org/10.1016/s1389-0417(02)00051-7.

[20] Vavrečka M., Farkaš I. & Lhotská L. (2011) Bio-inspired model of spatial cognition. In: Neural Information Processing, Springer Berlin Heidelberg, pp. 443–450. URL: https://doi.org/10.1007/978-3-642-24955-6_53.

[21] Cangelosi A. (2005) Symbol grounding in connectionist and adaptive agent models. In: New Computational Paradigms, Springer Berlin Heidelberg, pp. 69–74. URL: https://doi.org/10.1007/11494645_10.

[22] Pastra K. (2004) Viewing vision-language integration as a double-grounding case. In: AAAI Technical Report.

[23] Pastra K. (2005) Vision -language integration: a double-grounding case URL: http://rgdoi.net/10.13140/RG.2.2.35311.76965.

[24] LaValle S.M. (2006) Planning Algorithms. Cambridge University Press. URL: https://doi.org/10.1017/cbo9780511546877.

[25] Sakcak B., Weinstein V. & LaValle S.M. (2022), The limits of learning and planning: Minimal sufficient information transition systems. URL: https://arxiv.org/abs/2212.00523.

[26] Beer R.D. (1997) The dynamics of adaptive behavior: A research program. Robotics and Autonomous Systems 20, pp. 257–289. URL: https://doi.org/10.1016/s0921-8890(96)00063-2.

[27] Hope T., Stoianov I. & Zorzi M. (2010) Through neural stimulation to behavior manipulation: A novel method for analyzing dynamical cognitive models. Cognitive Science 34, pp. 406–433. URL: https://doi.org/10.1111/j.1551-6709.2009.01079.x.

[28] Nolfi S. (2011) Behavior and cognition as a complex adaptive system: Insights from robotic experiments. In: Philosophy of Complex Systems, Elsevier, pp. 443–463. URL: https://doi.org/10.1016/b978-0-444-52076-0.50016-x.

[29] Nolfi S. & Floreano D. (2004) Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines. MIT Press, Cambridge, MA, USA.

[30] Finite-state machine. https://en.wikipedia.org/wiki/Automata_theory#/media/File:DFAexample.svg. Accessed: 2023-05-22.

[31] Rivest R.L. & Schapire R.E. (1989) Inference of finite automata using homing sequences. In: Proceedings of the twenty-first annual ACM symposium on Theory of computing - STOC '89, ACM Press. URL: https://doi.org/10.1145/73007.73047.

[32] Kleinberg J. & Tardos E. (2005) Algorithm Design. Pearson, Upper Saddle River, NJ: Pearson, Cornell University.

[33] Gasarch W. (2002) The p=?np poll. SIGACT News 33, pp. 34–47.

[34] Complexity classes diagram. https://en.wikipedia.org/wiki/NP_(complexity)#/media/File:P_np_np-complete_np-hard.svg. Accessed: 2023-05-22.

[35] Angluin D. (1978) On the complexity of minimum inference of regular sets. Information and Control 39, pp. 337–350. URL: https://doi.org/10.1016/s0019-9958(78)90683-6.

[36] Gold E.M. (1978) Complexity of automaton identification from given data. Information and Control 37, pp. 302–320. URL: https://doi.org/10.1016/s0019-9958(78)90562-4.

[37] Kearns M. & Valiant L. (1994) Cryptographic limitations on learning boolean formulae and finite automata. Journal of the ACM 41, pp. 67–95. URL: https://doi.org/10.1145/174644.174647.

[38] Bonet M.L. & Buss S.R. (1994) Size-depth tradeoffs for boolean formulae. Information Processing Letters 49, pp. 151–155. URL: https://doi.org/10.1016/0020-0190(94)90093-0.

[39] Fajardo J.M., Roldan F.G., Realpe S., Hernández J.D., Ji Z. & Cardenas P.F. (2022) Frobs rl: A flexible robotics reinforcement learning library. IEEE 18th International Conference on Automation Science and Engineering (CASE) , pp. 1104–1109.

8. GROUP MEMBERS' CONTRIBUTIONS

Rafiqul Talukder:

- Main author of **Abstract**
- Main author of chapter 1
- Main author of sections 2.1, 2.2, 2.2, 4.3, 4.4, 4.5

Teemu Kettukangas:

- Main author of chapters 3, 6.
- Main author of sections 2.3, 4.1, 4.2, 4.6, 5.1, 5.2.

Each participant's contributions were distributed fairly evenly throughout the thesis. During software development and planning, both of the team members put in a fair amount of work, while working together, demonstrating their dedication and commitment to the thesis. Sections not mentioned above (2.1.1, 2.4 5.3 and **Tiivistelmä**) were written together. Finally, the full thesis was reviewed together and every flaw of the text was corrected. The team members' collaborative effort ensured the overall quality of the thesis.