

UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

TÍTULO: Design of a self-driving mini-robot for indoor navigation using evolutionary artificial intelligence algorithms

Trabajo de integración curricular presentado como requisito para la obtención del título de Tecnologías de Información

Autor:

Joseph Ricardo Gonzalez Nuñez

Tutor:

Ph.D. - Lorena de los Angeles Guachi Guachi

Urcuquí - July 13, 2020

SECRETARÍA GENERAL
(Vicerrectorado Académico/Cancillería)
ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES
CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN
ACTA DE DEFENSA No. UITEY-ITE-2020-00018-AD

A los 23 días del mes de marzo de 2020, a las 11:30 horas, de manera virtual mediante videoconferencia, y ante el Tribunal Calificador, integrado por los docentes:

Presidente Tribunal de Defensa	Dr. ARMAS ARCINIEGA, JULIO JOAQUIN , Ph.D.
Miembro No Tutor	Dr. CUENCA PAUTA, ERICK EDUARDO , Ph.D.
Tutor	Dra. GUACHI GUACHI, LORENA DE LOS ANGELES , Ph.D.

El(la) señor(ita) estudiante **GONZALEZ NUÑEZ, JOSEPH RICARDO**, con cédula de identidad No. **1718567983**, de la **ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES**, de la Carrera de **TECNOLOGÍAS DE LA INFORMACIÓN**, aprobada por el Consejo de Educación Superior (CES), mediante Resolución **RPC-SO-43-No.496-2014**, realiza a través de videoconferencia, la sustentación de su trabajo de titulación denominado: **Design of a self-driving mini-robot for indoor navigation using evolutionary artificial intelligence algorithms**, previa a la obtención del título de **INGENIERO/A EN TECNOLOGÍAS DE LA INFORMACIÓN**.

El citado trabajo de titulación, fue debidamente aprobado por el(los) docente(s):

Tutor	Dra. GUACHI GUACHI, LORENA DE LOS ANGELES , Ph.D.
--------------	---

Y recibió las observaciones de los otros miembros del Tribunal Calificador, las mismas que han sido incorporadas por el(la) estudiante.

Previamente cumplidos los requisitos legales y reglamentarios, el trabajo de titulación fue sustentado por el(la) estudiante y examinado por los miembros del Tribunal Calificador. Escuchada la sustentación del trabajo de titulación a través de videoconferencia, que integró la exposición de el(la) estudiante sobre el contenido de la misma y las preguntas formuladas por los miembros del Tribunal, se califica la sustentación del trabajo de titulación con las siguientes calificaciones:

Tipo	Docente	Calificación
Presidente Tribunal De Defensa	Dr. ARMAS ARCINIEGA, JULIO JOAQUIN , Ph.D.	9,5
Tutor	Dra. GUACHI GUACHI, LORENA DE LOS ANGELES , Ph.D.	10,0
Miembro Tribunal De Defensa	Dr. CUENCA PAUTA, ERICK EDUARDO , Ph.D.	9,0

Lo que da un promedio de: **9.5 (Nueve punto Cinco)**, sobre 10 (diez), equivalente a: **APROBADO**

Para constancia de lo actuado, firman los miembros del Tribunal Calificador, el/la estudiante y el/la secretario ad-hoc.

GONZALEZ NUÑEZ, JOSEPH RICARDO
Estudiante

Dr. ARMAS ARCINIEGA, JULIO JOAQUIN , Ph.D.
Presidente Tribunal de Defensa



Firmado electrónicamente por:
JULIO JOAQUIN
ARMAS
ARCINIEGA

Firmado Digitalmente por: LORENA DE LOS ANGELES GUACHI GUACHI
 Hora oficial Ecuador: 12/06/2020 16:43
 Dra. GUACHI GUACHI, LORENA DE LOS ANGELES , Ph.D.
Tutor

ERICK EDUARDO
CUENCA PAUTA

Signature numérique de ERICK
EDUARDO CUENCA PAUTA
Date : 2020.06.16 17:02:41 -05'00'

Dr. CUENCA PAUTA, ERICK EDUARDO , Ph.D.
Miembro No Tutor

TORRES MONTALVÁN, TATIANA BEATRIZ
Secretario Ad-hoc

Autoría

Yo, **Joseph Ricardo González Núñez**, con cédula de identidad 1718567983, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el/la autor(a) del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.
Urcuquí, Diciembre 2019.




Joseph Ricardo González Núñez
CI: 1718567983

Autorización de publicación

Yo, **Joseph Ricardo González Núñez**, con cédula de identidad 1718567983, cedo a la Universidad de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Urququí, Diciembre 2019.



CI: 1718567983

Dedication

I wish to address my work to those who dedicate their lives to do science. I am sure that the world would not be an interesting place without the work of such brilliant minds.

Joseph Ricardo González Núñez

Acknowledgements

I wish to express my deepest gratitude to my family, who has supported me through all my life, to my friends who have been my second family, and to my teachers, who have inspired me. Honestly, there are so many people that I have to thank that a simple page will not be enough to write all their names

Joseph Ricardo González Núñez

Resumen

Los sistemas autónomos tienen varias aplicaciones como por ejemplo exploración espacial, tareas de ensamblaje, mantenimiento en el hogar, entre otras. Estos sistemas deben ser capaces de adaptarse a una gran variedad de tareas. Los enfoques de aprendizaje por refuerzo y enfoques evolutivos son los mayores campos con características adaptativas. El algoritmo NEAT es una combinación de estos dos enfoques y por esta razón será usado en este trabajo con propósitos de navegación autónoma, en conjunto con enfoques de reconocimiento de color. El algoritmo NEAT ha sido ampliamente usado en aplicaciones de videojuegos y ambientes simulados. El trabajo presente tiene el objetivo de extender el uso del algoritmo NEAT a aplicaciones de procesamiento de imágenes y aplicaciones robóticas; introduciendo un algoritmo NEAT adaptado llamado NEAT auto tripulado para el reconocimiento de colores o NEAT-SDCR por sus siglas en inglés. El objetivo principal del presente trabajo es descubrir si NEAT-SDCR se desempeña bien en entornos reales. Para un primer enfoque, la tarea asignada es simple; un robot será entrenado para seguir objetos de color verde. El entrenamiento será realizado en un ambiente simulado y luego será probado en un robot real. Las contribuciones de este trabajo son implementar el sistema en un robot real, diseñar una función de aptitud para el problema y mejorar en general el rendimiento del algoritmo NEAT mediante el uso de un nuevo método de reproducción y usando un enfoque incremental.

Palabras clave: Sistemas autonomos, Aprendizaje por refuerzo, Enfoques evolutivos, NEAT

Abstract

Autonomous systems have many applications as space exploration, assembling tasks, and household maintenance. These systems have to be able to adapt to a wide variety of tasks. Reinforcement learning approaches and evolutionary approaches are two major fields with adaptability characteristics. The Evolutionary artificial intelligence (NEAT) algorithm is a combination of these two approaches, and, for this reason, in this work, it is used for autonomous navigation purposes in conjunction with color recognition approaches. The NEAT algorithm has been widely used in video game applications and in simulated environments. The present work attempts to extend the uses of the NEAT by using it in an image processing and robotic application introducing an adapted NEAT algorithm, called NEAT for self-driving with color recognition (NEAT-SDCR). The main aim of the present work is to discover if NEAT-SDCR, performs well in a real indoor environment. For a first approach, the assigned task is simple; a robot is trained to follow green objects. The training is done in a simulated environment; then it is tested in a real robot. The contributions of this work are the implementation of this system in a real robot, the design of a fitness function for the problem, and the improvement in general of the accuracy of the NEAT algorithm by the use of a new reproduction method and by using an incremental approach.

Keywords: Autonomous system, Reinforcement learning, Evolutionary approach, NEAT

Contents

1	Introduction	11
1.1	Problem statement	11
1.2	Justification	11
1.3	Contribution	12
1.4	Thesis overview	12
2	Objectives	13
2.1	General objective	13
2.2	Specific objectives	13
3	Theoretical framework of evolutionary algorithms	14
3.1	Evolutionary approaches	14
3.2	Evolutionary artificial intelligence algorithm (NEAT)	15
3.2.1	Related work	15
3.2.2	NEAT algorithm	15
4	Methodology	23
4.1	Assumptions	23
4.2	Data collection (Images)	24
4.3	Design and implementation of the proposed NEAT-SDCR algorithm	25
4.3.1	Design of the fitness function applied to color recognition	26
4.3.2	Novel changes explored on the original NEAT algorithm	29
4.4	Design and assembly of the proposed mini-robot system	30
4.4.1	Hardware design	32
4.4.2	Sending signals to gears motors to control movement	32
4.5	Implementation of the mini-robot and computer communication	35
4.5.1	Sending images from the camera to the arduino	36
4.5.2	Sending image data and receiving commands (Mini-robot)	36
4.5.3	Receiving image data and sending commands (Computer)	36
5	Experimental setup	37
5.1	Parameter settings	37
5.1.1	Hyper-parameters for mutations	37
5.1.2	General hyper-parameters	38
5.1.3	Hyper-parameters for the car	38
5.2	Experiments	39
5.2.1	Sexual reproduction vs. asexual reproduction experiments	39
5.2.2	Variation in the image size experiments	39
5.2.3	Variation in the population experiments	40
6	Results	41
6.1	Results of sexual vs. asexual reproduction	42
6.1.1	Experiment 1: Sexual reproduction vs. asexual reproduction compared in terms of generations	42

6.1.2	Experiment 2: Sexual reproduction vs. asexual reproduction compared in terms of total time	43
6.1.3	Experiment 3: Sexual reproduction vs. asexual reproduction compared in terms of time per generation	43
6.1.4	Experiment 4: Sexual reproduction vs. asexual reproduction compared in terms of size of the networks	43
6.2	Results of variation in the image size	45
6.2.1	Experiment 5: Variation in the image size tested in terms of accuracy	45
6.2.2	Experiment 6: Variation in the image size tested in terms of time per generation	45
6.3	Results of change in population	46
6.3.1	Experiment 7: Variation in the population tested in terms of accuracy	46
6.3.2	Experiment 8: Variation in the population tested in terms of time per generation	46
7	Conclusions and future work	48
	References	49
	Appendices	52
.1	Appendix 1.	53

Chapter 1

Introduction

An autonomous navigation system is able to make accurate decisions about navigation in order to reach its target. In this sense, there are many works related to autonomous system as robot localization [1], control of biped robots [2], [3], [4], control of quadrupedal robots [5], and control of underwater vehicles [6]. It is also remarkable by its adaptability the algorithm NEAT proposed in [7], this algorithm has been mostly used for games or robotics in simulated environments [8], [9], [10], [11]. The problem dealt with in this project is the application of NEAT in a real mini-robot in indoor environments for autonomous navigation purposes in conjunction with color recognition approaches. To the best of our knowledge, this is the first time that NEAT is applied in such a scenario. The NEAT algorithm cannot be applied directly over this problem because of problems as a lack of accuracy. For this reason, the present work proposes new modifications on NEAT to adapt it to the problem of robotics and to improve its accuracy in general; the adapted new algorithm is called NEAT-SDCR (NEAT for self-driving and color recognition). The dataset used in this work is generated using an own algorithm, and the performance of the algorithm is measured considering metrics like accuracy, time, and complexity (size) of the evolved network.

1.1 Problem statement

Cameras are becoming standard and affordable equipment for the robotic field. In particular, small and light on-board cameras are often used as the primary sensors for information gathering about the robot environment. Many Automatic visual navigation robots are based on image-navigation. These methods allow acquiring information-rich image description for subsequent detection of regions of interest such as moving objects, people, vehicles, geometric figures, color regions, etc.

Although NEAT algorithm has been applied to some problems, such as game playing and control systems in virtual environments (controlling robots and vehicles), the application to real scenarios for autonomous navigation has not been explored [7].

1.2 Justification

Self-driving robots, also often called autonomous robots, are desirable in fields as underwater exploration, operation in urban environments, household maintenance, wastewater treatment, delivering goods and services, among others, where image analysis has gained an important role with the emerging technologies and digital devices which provide a useful way to acquire and use high quality and economical video cameras for understand and detect a wide range of environments and objects such as people, vehicles, road signs on video/images sequences automatically.

In recent years, the use of autonomous robots in real scenarios has received considerable attention. Therefore, this work proposed a modified version of NEAT algorithm, called NEAT Applied to Self-driving Navigation (NEAT-SDCR), to create a mini-robot with autonomous visual navigation in indoor environments. To achieve this goal, NEAT-SDCR explores reproduction, population, and input rules to make navigation decisions based

on color identification. In this sense, this work provides a chance for further explorations of evolutionary algorithms or even industrial fields.

1.3 Contribution

This work intends to apply the NEAT algorithm to the autonomous visual navigation of a mini-robot. To achieve this, the original NEAT algorithm is explored. Then, the Fitness function is adjusted to the color recognition problem. Furthermore, reproduction, population, and input rules are explored and adapted to reach high performance. In addition, for experiments in realistic scenarios, a mini-robot system is built. It is compounded by an Arduino (for navigation purposes), a camera (to capture images), and a laptop (To execute NEAT-SDCR, process images, and send commands to Arduino). Obtained results allow determining whether NEAT-SDCR meets its efficiency expectations.

1.4 Thesis overview

This work is organized as follows. Section 2 describes the objectives of this work. Section 3 explains in detail the original NEAT algorithm. Section 4 detailed the proposed modifications, and the hardware design of the robot. Section 5 details the conditions under which the experiments were performed and the parameter settings. Section 6 shows the results as a comparison of the original NEAT algorithm and the proposed modifications. Section 7 explains the conclusions of the work and the future work derived from this.

Chapter 2

Objectives

2.1 General objective

To design and build an autonomous mini-robot system able to learn and react in presence of colored objects in indoor environments by the use of evolutionary approaches.

2.2 Specific objectives

- To adapt the most outstanding evolutionary algorithm to make navigation decisions based on color recognition. For this work purposes, only green color objects will be followed by the mini-robot; other colors will be ignored.
- To design and build a mini-robot system capable of communicating with the computer to send images and receive commands that allow it to navigate in an indoor environment.
- To evaluate the ability of the adapted evolutionary approach to make navigation decisions based on color recognition in a real-scenario in terms of maximum direction.

Chapter 3

Theoretical framework of evolutionary algorithms

The present work is closely related to computer vision, autonomous system, and robotics. Artificial intelligence approaches are mostly used in these fields; however, this work explores the ability of evolutionary algorithms to work for Autonomous Systems (ASs). The behavior of an AS can be represented as a function in which inputs are obtained from sensors of the robot (stimulus), and outputs are the actions that this AS tacked as reactions to this stimulus.

An AS is characterized by the ability to perceive, learn, and adapt to respond to the world around them; in everyday situations, humans take for granted, is still a challenge. In this area of research, ASs-based navigation involves observing the world around them by using computer vision algorithms, then processing the collected data, and generating adequate responses. In recent years, there has been an increasing interest in Artificial Intelligence (AI) approaches. These algorithms have been widely applied to many fields. For example, robotic applications as controlling four-legged robots [8], robot localization [1], control of drones [12], soccer player robots [13], controlling biped robots [2], [3], and cooperative robotics [11]. Video games applications as real-time strategy [9], [10], and simulations [14], [15]. Computer vision applications as pedestrian detection [16], recognition of passengers on motorcycles [17], hand-gestures detection [18], image segmentation [19], [20], and background extraction [21], [22]. AI algorithms are characterized by their ability to generalize knowledge based on information extracted from training data.

In this chapter, evolutionary algorithms for AS-based navigation are presented. Starting with some relevant applications, challenges, and the main workflow of this kind of algorithms. Finally, some relevant applications and theoretical foundations of the NeuroEvolution of Augmenting Topologies (NEAT) algorithm are described. Evolutionary algorithms optimize its accuracy iteratively based on ideas of natural evolution; it means that these algorithms need a way to know if the output is accurate or inaccurate in each iteration. In other words, they need feedback to improve their performance.

3.1 Evolutionary approaches

Evolutionary approaches take part in the bigger field of bio-inspired algorithms. These methods use ideas from nature to solve engineering problems, for example, ant colony algorithms [23], which are inspired by the organization of real ants. Neural networks [24] that are inspired in the functioning of animal brains. Swarm algorithms [25] based on the emergency properties of groups of animals as flocks of birds or shoal of fishes. Researchers watch nature in order to find creative solutions and create new algorithms based on natural systems. Evolutionary approaches are inspired by the evolution of species, which was proposed by Charles Darwin.

In the evolution of species, living beings have to be adapted to their environment in order to survive and pass their genes to other generations. These living beings are different from each other because of mutations. The environment selects the individuals with better characteristics, and consequently, these features become

more frequent in future generations; in this way, species change over time. This process can be seen as an optimization of the species to its environment.

Evolution is the force that is responsible for the variety of living beings existing today. Because of its undisputed achievements in nature, researchers, as Stanley [7] and Siebel [26], have proposed to use concepts as reproduction, mutation, and selection to create algorithms. These new algorithms are classified as evolutionary approaches. One of the most promising evolutionary approaches is NEAT.

3.2 Evolutionary artificial intelligence algorithm (NEAT)

NEAT stands for Neuro Evolution of Augmented Topologies, it is a new kind of neural network algorithm proposed by Stanley et al. [7]. This algorithm is used to evolve synaptic weights along with the structure of a neural network.

3.2.1 Related work

Xu et al. [8] made a comparison between two evolutionary algorithms called CMA-NeuroES and NEAT. They tested those two algorithms in a simulation. The assigned task was increasing the speed of a four-legged robot. This automaton has four joints, one in each leg. The inputs of the neural networks are the angles and the speeds of every leg. The outputs are the desired angle. The fitness function is the displacement walked by the robot. The work purpose was to discover if CMA-NeuroES and NEAT can evolve adequate topologies and weights to increase the gait speed of the four-legged robot.

Olesen et al. [9] used the NEAT algorithm in the problem of Real-Time Strategy (RTS) in games. Authors created an AS that is able to play against a human in the game Globulation 2(G2). G2 is a strategy game whose objective is to manage an empire by controlling the gathering of resources, the creation of buildings, and the training of troops. Additionally, G2 is focused on macro-management rather than micro-management, not considering micro-management makes possible a better performance in NEAT because of the dimensionality reduction.

Lowell et al. [10] compared two evolution methods NEAT and HyperNEAT. These algorithms were tested in the keep-away soccer game. In this problem, keepers agents have to keep the ball while avoiding taker agents to take the ball. The keeper agents have to decide when to hold the ball or pass it to another keeper. Keepaway soccer game is a fractured problem; it means that the optimal decision is changing fastly as the state change.

Nitschke et al. [11] compared two evolution methods Collective Neuro-Evolution 2(CONE2) and NEAT in performing a cooperative task. In this task, simulated robots have to move and join blocks of different types. These blocks have to form a line on a specific order; the order rules are set up by the user at defining the environment. The purpose of Nitschke is to identify if CONE2 and NEAT are able to evolve behavioral heterogeneity; this is, robots specialized in moving one type of block. Nitschke concludes that CONE2 outperforms NEAT in evolving heterogeneity behavior.

3.2.2 NEAT algorithm

This section describes the original NEAT algorithm introduced by Stanley et al. in [7]. NEAT implements three innovations, historical markings, speciation, and evolving from simpler. The first innovation is genetic encoding through Tracking Genes; it is called *Historical Markings*; this makes it possible that crossover between two networks parents produces offspring that preserves parent performance skills. The second innovation is *speciation* to protect structural innovations from disappearing before being optimized; it is doing so by allowing individuals to compete only inside their species. The third innovation is *evolving a complex neural network from the simpler possible* to avoid over-complexity in networks. The simpler possible structure is input neurons connected directly to output neurons. NEAT has been widely applied in the field of ASs in tasks as robotics [8], real-time strategy games [9].

NEAT is composed of four steps: genetic representation and initialization, fitness evaluation, and selection of the best individuals, sexual reproduction, mutations, and speciation, as is shown in Figure 3.1. The NEAT algorithm is bio-inspired, which means that many ideas that NEAT uses are based on nature. For example, natural selection, sexual reproduction, mutations, and combinations of genes are ideas from evolution theory.

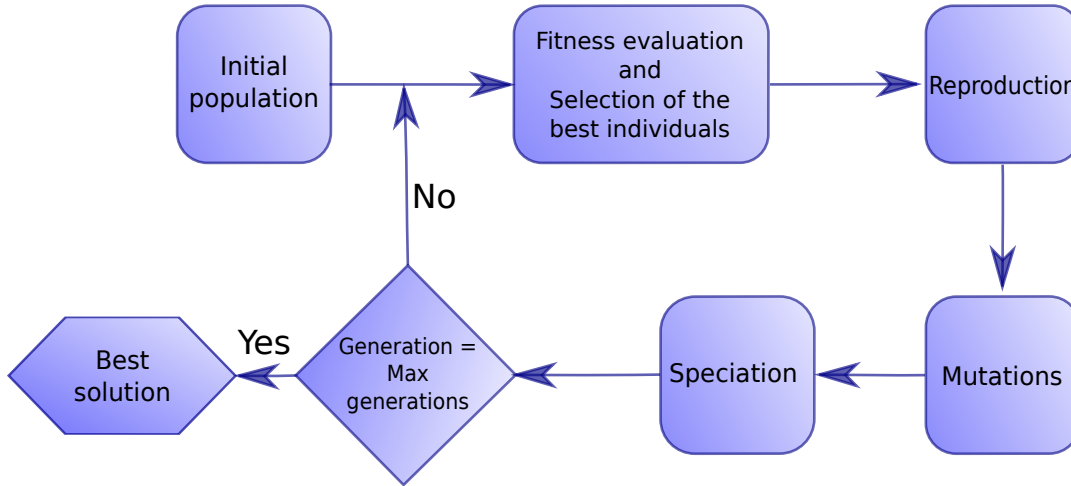


Figure 3.1: Stages of NEAT, this diagram is based on [7]

3.2.2.1 Genetic representation and initial population

Every neural network is represented using its genome g , which is a group of genes gen as can be seen in Figure 3.2

$$g = \{gen_1, gen_2, gen_3, \dots, gen_s\}$$

Where s is the number of genes, and therefore the size of the genome. Every gene represents a connection between two neurons in the neural network, and it is composed of five subfields.

$$gen = \{input, output, weight, enable, innovation\}$$

- **input:** is the identifier of the incoming neuron in the connection.
- **output:** is the identifier of the outgoing neuron in the connection.
- **weight:** is a float value representing the strength of the connection
- **enable:** is a boolean value that indicates if the connection is activated or not in the neural network. Mutations could deactivate or activate a connection; this will be explained later.
- **innovation:** it is a global unique identifier for each connection.

The information of connections is explicitly written in the genomes. For this reason, this algorithm uses a *direct encoding*. Algorithms with *indirect encoding* use a function to store the information of connections. Genomes g are grouped in species sp , the criteria used to classify a genome in one of the species is explained later. Parameter np is the number of genomes that are a member of that species.

$$sp = \{g_1, g_2, g_3, \dots, g_{np}\}$$

All the different species together form the entire population pop of individuals (genomes). The parameter ns is the number of living species in the population.

$$pop = \{sp_1, sp_2, sp_3, \dots, sp_{ns}\}$$

In order to start the algorithm, it is necessary to create the first population; this generation, unlike the others, will not have parents. Stanley et al. proposed creating the minimal possible genomes, which means creating all the possible connections between all the inputs and all the outputs.

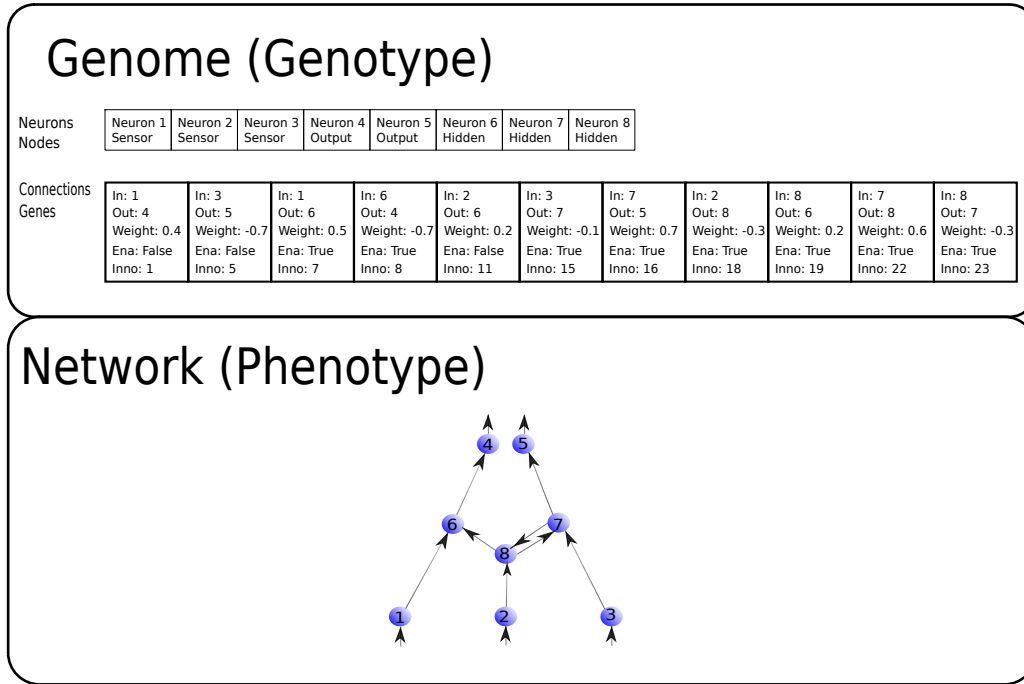


Figure 3.2: Diagram of genes, this diagram is based on [7]

3.2.2.2 Fitness evaluation and selection of the best individuals

Every genome in the population is a generator of solutions. The algorithm has to find suitable neural networks to solve the posed problem; it means finding topologies that make the minimal possible error. For this reason, the algorithm has to select the best individuals in each generation, to create more copies of them. This idea is directly extracted from the concept of natural selection in evolution theory. Natural selection establishes that only the best-adapted to its environment will survive and pass its genes to the next generations. In this case, only the best topologies will create copies of themselves.

The fitness function is used to qualify the performance of genomes, the higher the value of fitness, the better is the performance of the topology. The fitness function has to be specifically designed for the problem. The **reward row** presented in this section will be used in this problem. A **reward row** can be calculated for each of the images used in the training phase. The **reward row** has dimensions $(1, j)$, and the original image has dimensions (i, j) . A reward row can be calculated using the following formula.

$$r_j = \sum_1^i (gp_{ij} - rp_{ij} - bp_{ij}) \tag{3.1}$$

- gp_{ij} is the green component of the pixel in the i row and the j column.
- rp_{ij} is the red component of the pixel in the i row and the j column.
- bp_{ij} is the blue component of the pixel in the i row and the j column.
- r_j is the j element in the reward row.

The computation $(gp_{ij} - rp_{ij} - bp_{ij})$ can be understanding as to the intensity of the color green in the pixel. Therefore, equation 3.1 adds the intensities of the i pixels in the j column to obtain the **reward row**. The calculation used to obtain the direction of the mini-robot will be detailed in the section 4.3.1, particularly equations 4.4, 4.8 are used to compute the rotation angle of the mini-robot. The speeds of the output layer of the neural network will be used to compute the rotation angle, and that, in turn, will be used to compute the fitness. That is because the direction will be pointing to an element of the **reward row**, and that element will be the fitness of the current genome.

In each generation, the algorithm will find the fitness associated with each genome. Then, genomes with lower performance will be deleted. The percentage of the population that will be removed can be controlled using a hyper-parameter called **kill percentage**.

3.2.2.3 Sexual reproduction

The algorithm NEAT uses sexual reproduction. In other words, it generates new individuals by combining two previous existing individuals (two parents). The problem of combining two typologies of neural networks is complicated because it implies coherently combining two graphs. It means finding common structures in graphs to fuse them into one graph and keeping no-common structures of both graphs in a way that offspring improve his performance. The NEAT algorithm creatively solves this problem using *historical markings*.

Each connection has a unique identifier, which is the subfield **innovation** explained in section 3.2.2.1. Every time that a new connection appears as a result of a mutation, the algorithm assigns the number of a global counter to it, and then the global counter is increased. This method is called *historical marking*. Multiples genomes can have the same connection, and this will be reflected in the fact that they have an innovation number in common. A gene could be spread in multiples genomes because these genomes are descendants of a common ancestor.

When two genomes are going to be crossed, It is possible to know what structures they share by analyzing what innovation numbers are common in their genomes. The common structures in the genomes are inherited from any parent, while non-common structures are inherited from the more fitness parent. This process is illustrated in Figure 3.3.

In this example subscripts are representing the innovation numbers in genes.

$$g_1 = \{gen_3, gen_4, gen_5, gen_7, gen_{11}, gen_{12}, gen_{14}, gen_{15}\}$$

$$g_2 = \{gen_2, gen_3, gen_5, gen_7, gen_8, gen_9, gen_{12}, gen_{14}\}$$

The intersection of g_1 and g_2 are common connections and can be inherited from any parent. It is important to keep in mind that the subset $g_1 \cap g_2$ of genes are present in both parents, but despite being the same genes, they could have different information because of mutations. They could differ in the subfields *weight* and *enable*.

$$g_1 \cap g_2 = \{gen_3, gen_5, gen_7, gen_{12}, gen_{14}\}$$

Non-common genes as g'_1 for the genome 1 and g'_2 for the genome 2 are inherited from the more fitness parent.

$$g'_1 = g_1 - (g_1 \cap g_2) = \{gen_4, gen_{11}, gen_{15}\}$$

$$g'_2 = g_2 - (g_1 \cap g_2) = \{gen_2, gen_8, gen_9\}$$

Then, the resulting genome will be g_r

$$g_r = (g_1 \cap g_2) \cup g^*$$

Where g^* is g'_1 or g'_2 , depending on who is the more fitness parent. When a new individual is born, it has a random chance to develop mutations in his genome. The aim of the next section is explaining these mutations.

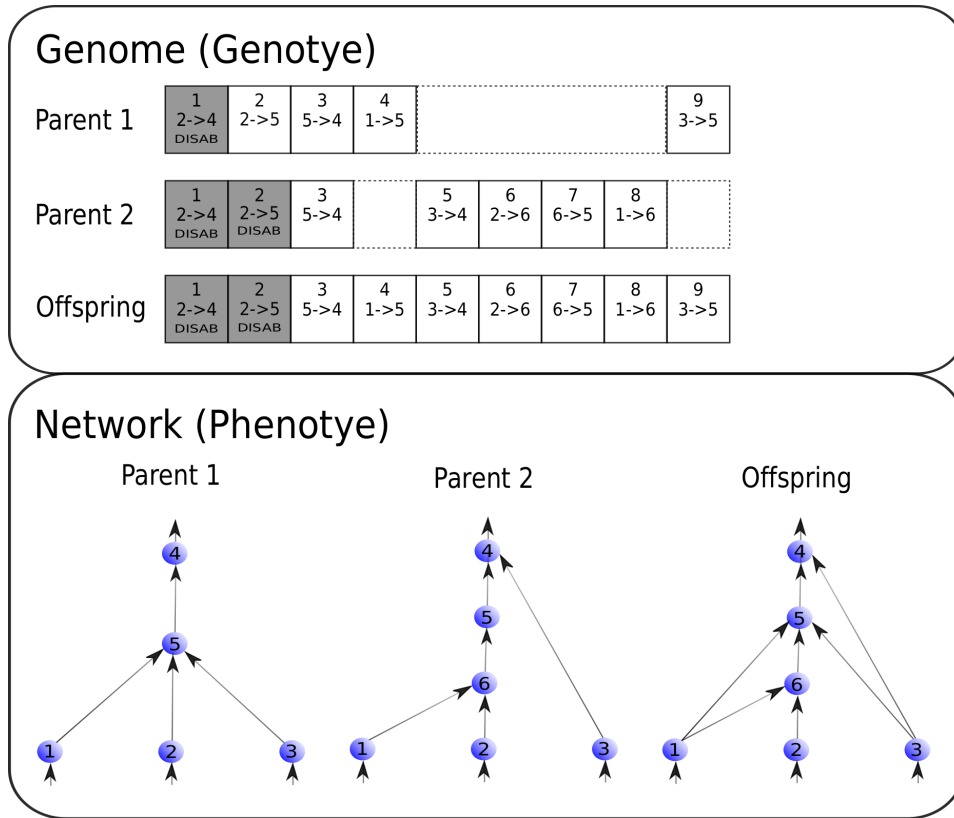


Figure 3.3: Sexual reproduction, this diagram is based on [7]

3.2.2.4 Mutations

Each new genome has a random chance to be born with mutations. These mutations are of three different types; adding a connection, adding a node and changing the weight of the connections. The first two are structural mutations; it means that they change the topology of the neural network. The third one does not change the structure, but it changes the internal information of the genes.

- **Mutation adding connections**

The mutation of adding new connections works by randomly choosing two neurons and joining them using a random weight, as can be seen in Figure 3.4.

$$new_gen = \{input, output, weight, enable, innovation\}$$

input and *output* are two neurons chosen randomly; the value of *weight* is also chosen randomly. The *enable* field is set to true, showing that the new connection is active. The *innovation* field is set to the global counter of the *historical marking*, and it will be the unique identifier of the new gene.

- **Mutation adding neurons**

The mutation of adding a new node works by randomly choosing a connection and adding a new neuron in the center. This process generates two new connections, a connection which has as **output** the new neuron, and a connection which has as **input** the new neuron. The **weight** of the first connection is set to 1, and the **weight** of the second connection is set to the weight of the original connection. The previous connection does not disappear but gets disabled. The new genes get new numbers for the **innovation** field, and they are different from each other. This process is illustrated in Figure 3.5, and in the example below. The gene before the mutation looks like this.

$$previous_gen = \{input = in_1, output = ou_1, weight = w, enable = true, innovation = inno_1\}$$

After the mutation there are three genes, the previous gene get disabled and two new genes.

$$previous_gen = \{input = in_1, output = ou_1, weight = w, enable = false, innovation = inno_1\}$$

$$new_gen_1 = \{input = in_1, output = new_neuron, weight = 1, enable = true, innovation = inno_2\}$$

$$new_gen_2 = \{input = new_neuron, output = ou_1, weight = w, enable = true, innovation = inno_3\}$$

• Mutation changing weights

This mutation can change the **weight** field or the **enable** field.

If the weight is mutated, the algorithm simply chose a new weight for the gene as in the example bellow.

$$gen = \{input = in_1, output = ou_1, weight = w, enable = true, innovation = inno_1\}$$

$$gen = \{input = in_1, output = ou_1, weight = new_w, enable = true, innovation = inno_1\}$$

The mutation of the field **enable** works only on deactivated connections. If the enable field is setting to false, then this mutation has a random chance to activate the connection again by setting the field to true.

$$gen = \{input = in_1, output = ou_1, weight = w, enable = false, innovation = inno_1\}$$

$$gen = \{input = in_1, output = ou_1, weight = w, enable = true, innovation = inno_1\}$$

This process is illustrated in Figure 3.6.

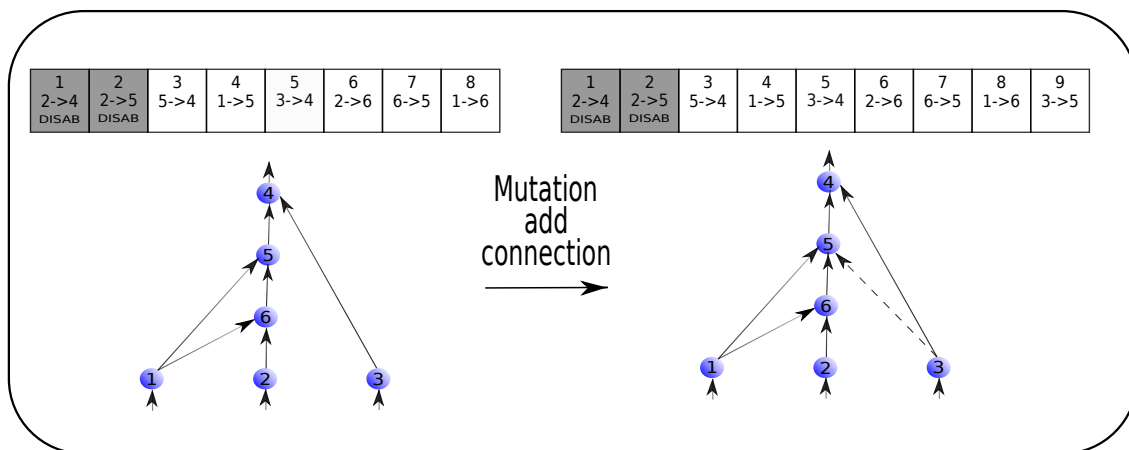


Figure 3.4: Mutation add connection, this diagram is based on [7]

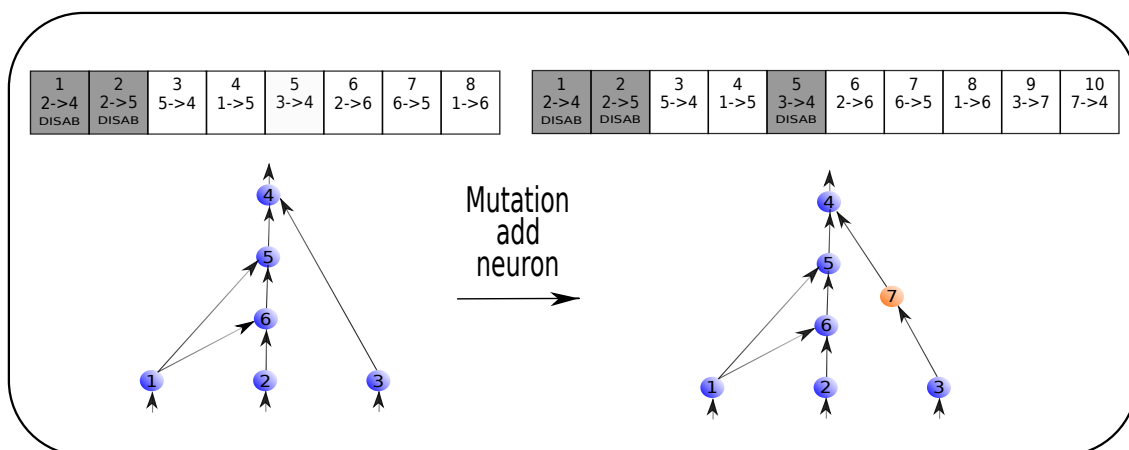


Figure 3.5: Mutation add node, this diagram is based on [7]

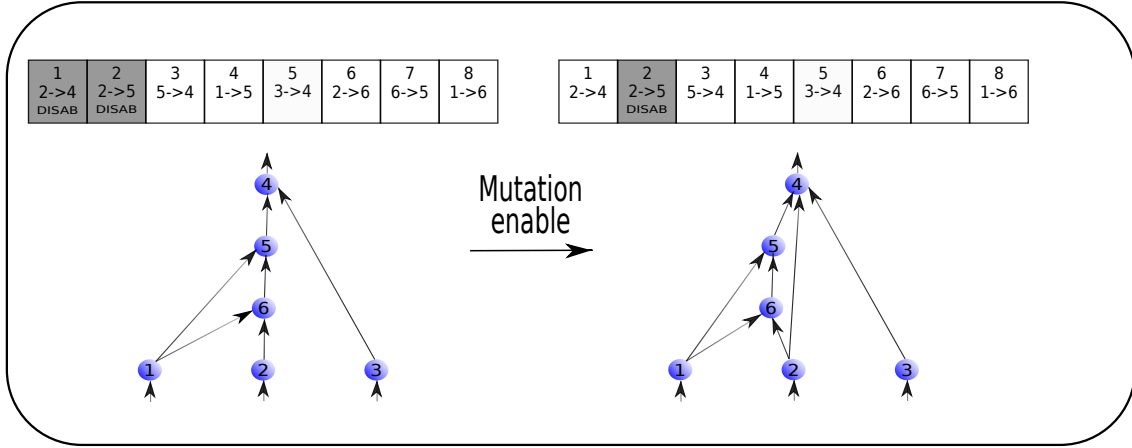


Figure 3.6: Mutation enable, this diagram is based on [7]

3.2.2.5 Speciation

The structural mutations (*mutation add connection* and *mutation add neuron*) proposed in this algorithm tend to decrease the fitness of the networks. This is a problem because lower fitness implies lower survival probabilities. These new structures can be the key to future behavioral innovations. Nevertheless, they could not survive. To solve this problem, Stanley et al. took an idea from nature, the concept of *speciation*. In nature, species do not compete for resources globally, but instead, each species compete in his own locally ecological niche. For this reason, NEAT algorithm groups various genomes in species.

$$sp = \{g_1, g_2, g_3, \dots, g_{np}\}$$

Where sp is the species, g_i are the genomes and np is the number of genomes in that species. In order to know what is the species of a genome, it is necessary to have a mechanism to measure the distance or the difference between two genomes. If two genomes have a greater distance, it means that they are poorly correlated, and they belong to different species. If two genomes have a small distance, it means that they are strongly correlated, and they belong to the same species. *Historical markings* are useful to measure the distance between two genomes.

$$\delta = \frac{c_1 D}{N} + c_2 \overline{W}$$

$$D = n(g_i - (g_i \cap g_j) + g_j - (g_i \cap g_j))$$

$$\overline{W} = \sum_{k=1}^c |w_{ik} - w_{jk}| \quad c = n(g_i \cap g_j)$$

In the above equation, δ is the distance between the species. D is the number of genomes that are non-common, in other words, the number of elements of the set $g_i - (g_i \cap g_j) + g_j - (g_i \cap g_j)$, where g_i and g_j are the genomes of which the distance is being calculated. \overline{W} is the average of the differences in the values of the weights of the common connections ($g_i \cap g_j$). c_1 and c_2 are coefficients to adjust the importance of structural and weight differences. This distance δ is useful to classify the genomes in species.

In order to help genomes with structural innovations to survive, Stanley et al. proposed a correction to the fitness function based on the δ parameter.

$$f'_i = \frac{f_i}{\sum_{j=1}^n sh(\delta(i, j))}$$

$$sh(\delta(i, j)) = \begin{cases} 1 & \text{if } \delta(i, j) < \delta_t \\ 0 & \text{if } \delta(i, j) > \delta_t \end{cases}$$

In the above equation, f'_i is the corrected fitness function. f_i is the original fitness. δ_t is the distance threshold. If the distance δ is lower than δ_t , the genomes belong to the same species. In the other hand, if distance δ is

greater than δ_t , they belong to different species. In this way, the summation $\sum_{j=1}^n sh(\delta(i, j))$ is equal to the number of individuals in the current species. For this reason, if any species has many individuals, the corrected fitness decreases, and structural innovations are protected; that is to say, new species have higher fitness.

Chapter 4

Methodology

This chapter aims to describe the methodology used in this work. It is divided into the stages proposed in Figure 4.1. The main stages are: 1) Data collection, 2) Design and implementation of the NEAT algorithm for self-driving mini-robot by color recognition (NEAT-SDCR), 3) Design and assembly of the mini-robot system, 4) Implementation of mini-robot and computer communication, and 5) Training using the NEAT algorithm. Results obtained to determine whether NEAT-SDCR meets its efficiency expectations in realistic scenarios in indoor environments are covered in chapter 6.

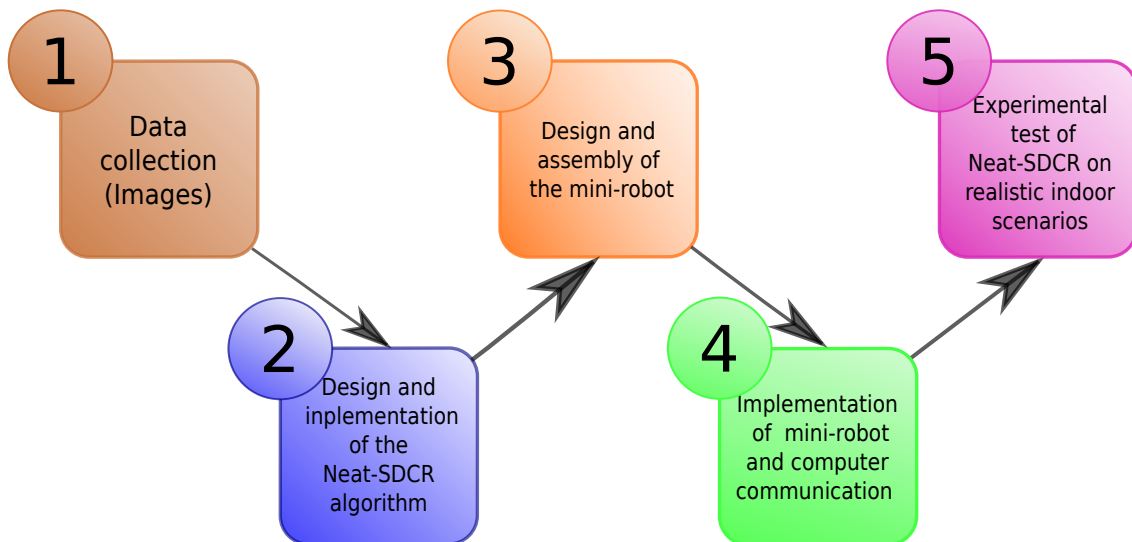


Figure 4.1: Diagram of methodology

4.1 Assumptions

To apply NEAT-SDCR algorithm on visual autonomous navigation of a mini-robot, this work considers the following assumptions.

- The mini-robot is designed to navigate in indoor environments without illumination changes. It is better if there are no obstacles in the path of the mini-robot since it has not been trained to avoid them if there are obstacles in the path the mini-robot could get stuck.
- The mini-robot does not have a mechanism to sense depth in images, and this fact is evidenced in the assumption made to get equation 4.8. This could cause that big-targets (green-objects) could seem little because of the distance, and therefore these targets could lose importance for the mini-robot at making decisions.

- As mentioned in the section 4.3.1.2, the movement of the mini-robot is restricted over 2-dimensions (over a surface); in other words, it cannot climb or fly. It is better if the land or the surface over which the automaton is navigating is flat and not steep.

4.2 Data collection (Images)

This task is based on *Data Generator* procedure as is shown in Algorithm 1. It is used to create the dataset of images. The resulting images are in the RGB color model because the camera of the mini-robot uses this standard. The dimensions of these images can be adjusted by using the parameters *height* and *length*. The parameter *green_probability* can be adjusted to change the percentage of the color green in the image. The procedure *Data Generator* is executed each time that a genome is tested, and the image is not stored; therefore, the image is eliminated after being used. Figure 4.2 shows generated images of size (2x2), and Figure 4.3 shows generated images of size (4x4). This code creates images of the desired size by painting pixels randomly. Nevertheless, the code ensures that there is at least one green pixel or one green region in the resulting image.

For creating an image, the Algorithm 1 picks a random value in the range from 0 to 1 for each pixel; if this value is less than the threshold of *green_probability*, the pixel will be green, if the value is greater than the threshold, the pixel will not be green. In the case of non-green pixels, the code chooses random values from 0 to 255 for each color component. In the case of green pixels, the code picks values for red, green, and blue components in such a way that $red + blue < green$.

In the beginning, a set of images with objects of different colors was thought to be the training set. These images were downloaded from the internet, and they contain objects like flowers, fruits, toys, or any object with different colors. However, these images lost the green component at being resized; the change of the size of the images is necessary to implement an incremental approach, as explained in the section 4.3.2.1. Figure 4.4 shows an example of this fact. For this reason, this work does not use any data set of images downloaded from the internet; instead, this work uses images generated by the code 1. The loss of green color in the images is an important problem because the navigation is based on this color; the mini-robot has to train to follow this color.

Algorithm 1 NEAT-SDCR Data generator

```

1: procedure DATA GENERATOR
2:   for  $i < height$  do
3:     for  $j < length$  do
4:        $color \leftarrow rand(1)$ .
5:       if  $color < green\_probability$  then
6:          $green \leftarrow rand(1) * 255$ 
7:          $red \leftarrow rand(1) * green$ 
8:          $blue \leftarrow rand(1) * (green - red)$ 
9:       else
10:         $green \leftarrow rand(1) * 255$ 
11:         $red \leftarrow rand(1) * 255$ 
12:         $blue \leftarrow rand(1) * 255$ 
13:       end if
14:        $pixel(i)(j)(0) \leftarrow red$ 
15:        $pixel(i)(j)(1) \leftarrow green$ 
16:        $pixel(i)(j)(2) \leftarrow blue$ 
17:     end for
18:   end for
19: end procedure

```

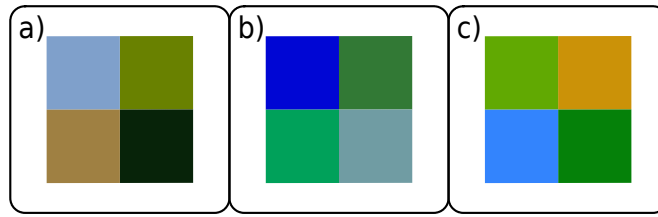


Figure 4.2: Images generated by the Data Generator code size(2x2)

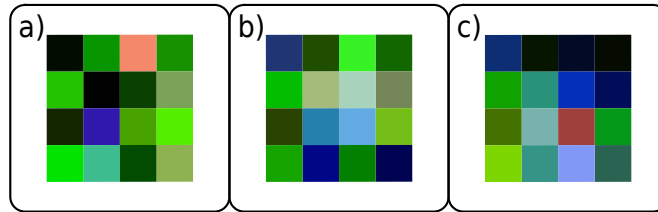


Figure 4.3: Images generated by the Data Generator code size(4x4)

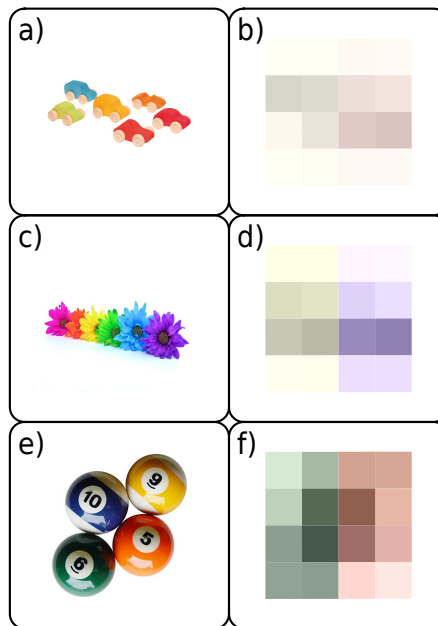


Figure 4.4: Resizing images: b) is the result of resizing the image a) to a size of (4x4), d) is the result of resizing the image c) to a size of (4x4), f) is the result of resizing the image e) to a size of (4x4)

4.3 Design and implementation of the proposed NEAT-SDCR algorithm

NEAT algorithm cannot be applied directly over the color recognition problem; it requires some modifications. The aim of this section is to describe in detail the proposed modifications to the NEAT algorithm called NEAT-SDCR (NEAT algorithm for self-driving mini-robot by color recognition). The changes are mainly focused on the fitness function, reproduction, and incremental training approach.

The fitness function is different for each problem; for this reason, the problem of following colors requires the design of a specific fitness function. This section is divided as follows: a) Design of the fitness function, and b) Novel proposed changes regarding the original NEAT algorithm.

4.3.1 Design of the fitness function applied to color recognition

The procedure to calculate the fitness of each genome in the population is described in the following. As explained in the subsection 3.2.2.2, the NEAT algorithm uses the fitness function to test the performance of each genome (neural network) of the population. Genomes with higher fitness will have more offspring, and consequently, their genes will spread over the population. In this way, the NEAT algorithm optimizes its solutions; therefore, the design of the fitness function is a crucial step in the implementation of NEAT. In this work, a neural network receives an image as input, and his fitness is measured by determining if the neural network can turn in the direction of the higher green color intensity. The mini-robot has to decide which movement to carry out based on the identified color. In this sense, this section describes how the outputs of the neural network are related to the movement of the mini-robot, and how the movement is related to the fitness function. There are two neurons in the output layer of all the networks. The values in these neurons determine the movement and fitness of each genome. The process of this transformation is detailed in this section. The necessary steps to model the motion and the fitness function of the self-driving mini-robot are a) choosing the appropriate angle, and b) determining the goal object that is pointed by the mini-robot.

4.3.1.1 Choosing the appropriate angle

The mini-robot follows a target object (green) by changing its direction; in other words, it turns to go to the target object. The difference in velocities of two wheels is what causes a rotation angle in the mini-robot. This stage exploits the use of a network evolved by the modified NEAT algorithm. It receives the inputs images tacked by the camera, while its output is compounded of two neurons v_1 and v_2 in the output layer of the neural network. These are a pair of speeds for the left and the right wheels. Therefore, the network sees an image, and it takes a decision represented as a pair of velocities v_1 and v_2 . v_1 is the velocity for the left-front wheel, and v_2 is the velocity for the right-front wheel. Back wheels do not have speeds, because the mathematical model of the movement is easier in this way. Output speeds values range from 0 to 1, to get larger values, it is necessary to multiply the velocities in the output layer v_1 and v_2 by a hyper-parameter called maximum velocity v_{max} . By doing so, it is possible to restrict the maximum velocity of the car-robot and ranging the velocity over a big range of predefined values.

Different speeds in wheels implies different covered distances d_1 and d_2 , and this, in turn, will change the initial direction with an α rotation angle, as shown in Figure 4.5. It can be demonstrated that α_1 is equal to α using Figure 4.5 and the laws of complementary angles, therefore the problem of finding the rotation angle α can be reduced to the problem of finding α_1 . The distances covered by the left wheel and the right wheel can be expressed as $d_1 = v_1 * t_{reac}$ and $d_2 = v_2 * t_{reac}$, where t_{reac} is the time that the velocity is applied. It is possible to use d_1 and d_2 to find the α angle because, these displacements are portions of two circumferences centered in the same point as can be appreciated in Figure 4.5. The circumference of d_1 has r as ratio and the circumference of d_2 has $r + l_{car}$ as ratio, where l_{car} is the length of the car. These two relations are represented in the equations 4.1 and 4.2.

$$\alpha * r = d_1 \quad (4.1)$$

$$\alpha * (r + l_{car}) = d_2 \quad (4.2)$$

solving r and finding the value of α in the equations 4.1 and 4.2.

$$\begin{aligned} r &= \frac{d_1}{\alpha} \\ r &= \frac{d_2}{\alpha} - l_{car} \\ \alpha &= \frac{d_2 - d_1}{l_{car}} \end{aligned} \quad (4.3)$$

the displacement is related with the velocity by the equation $d = v * t$, using this fact in equation 4.3

$$\alpha = \frac{(v_2 - v_1) * t_{reac}}{l_{car}}$$

where reaction time t_{reac} can be interpreted as the time which the velocity is applied over the mini-robot. Finally it is necessary to multiply v_1 and v_2 by the max velocity v_{max} and also by minus 1 in order to make right rotations positive angles and left rotations negative angles

$$\alpha = \frac{(v_2 - v_1) * t_{reac} * -v_{max}}{l_{car}} \tag{4.4}$$

the relation between the speeds in the wheels v_1 , v_2 and the resulting angle α is given by equation 4.4. Reaction time t_{reac} , max velocity v_{max} and the length of the car l_{car} are hyper-parameters that can be adjusted at the beginning of the training process of the neural network.

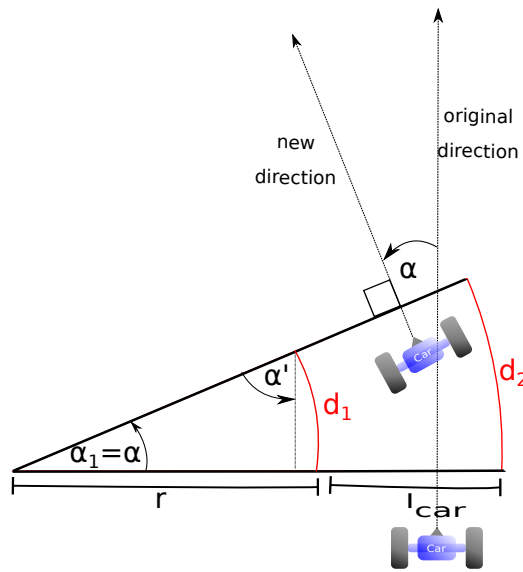


Figure 4.5: Rotation angle caused by the difference in the speeds of the left and the right wheels

4.3.1.2 Determining the target object

After choosing the appropriate angle, the next step is to know what object is being exactly pointed by the mini-robot. The movement of the mini-robot is restricted by its automobile shape; in other words, it cannot climb or go down or fly. It means that the mini-robot movement is limited in two dimensions (over a surface). For this reason, we will represent each input image of size (i, j) (height, width) as a **reward row** with dimension $(1, j)$, that is to say, keeping the width j of the original image, as shown in Figure 4.6. In this **reward row**, each element represents the corresponding j_{th} column of the original image. If the angle α is pointing to a specific element in the **reward row**, it means that the mini-robot is looking to all the i pixels presented in the column of the original image associated with that element. The specific values of the **reward row** are giving by equation 3.1, and they are described in detail in section 3.2.2.2; using this procedure, it is possible to handle the motion and rotations of the mini-robot in a two-dimensional space.

Figure 4.6 represents the position of the robot and the position of the image captured from the camera; each image is represented with its **reward row**, as we discussed earlier. The following steps are necessary to relate the rotation angle and the pointed element in the reward row. From Figure 4.6 two triangles are obtained. One triangle is related to the rotation angle α with the relative position p of the selected element from the center of the image. The other triangle is associated with a maximum possible rotation angle α_{max} with the maximum possible p that is to say $j/2$ the half of the image width. Equations 4.5 and 4.6 represent the relations between both triangles.

$$\tan(\alpha) = \frac{p}{CA} \tag{4.5}$$

$$\tan(\alpha_{max}) = \frac{j/2}{CA} \tag{4.6}$$

CA is equivalent to the distance from the mini-robot to the image. Solving CA and finding the value of p in the equations 4.5 and 4.6.

$$CA = \frac{p}{\tan(\alpha)}$$

$$CA = \frac{j/2}{\tan(\alpha_{max})}$$

$$p = \frac{\tan(\alpha)}{\tan(\alpha_{max})} * (j/2) \tag{4.7}$$

Equation 4.7 relates the rotation angle α with the relative position p , in order to represent the absolute position instead of the relative position it is necessary add a half of the width of the original image to the equation 4.7.

$$p = \frac{\tan(\alpha)}{\tan(\alpha_{max})} * (j/2) + (j/2) \tag{4.8}$$

Finally, equation 4.8 relates the rotation angle α with the absolute position p that is to say the element in the *reward row* that is being pointed by the rotation angle. Using the equation 4.8 and the *reward row* created by the equation 3.1, it is possible to determine the fitness of each genome in the population. A strong assumption is used in these computations; that is, all the images captured by the camera are at distance CA from the mini-robot, the implications of this assumption were discussed in the section 4.1. The maximum possible rotation α_{max} and the width image j are hyper-parameters that can be adjusted at the beginning of the training process.

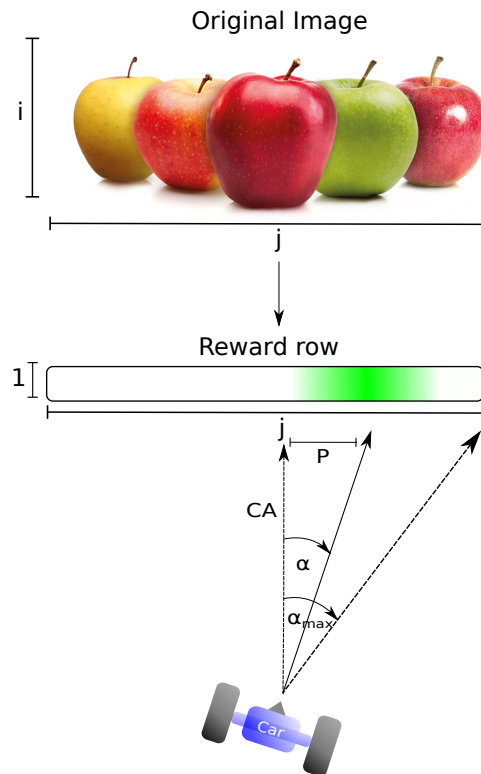


Figure 4.6: Mini-robot pointing to green objects and representation of the reward row

4.3.2 Novel changes explored on the original NEAT algorithm

These changes are focused on a) adjusting NEAT to self-driving navigation of a mini-robot by color recognition, and b) increasing the performance of the algorithm.

4.3.2.1 Incremental training approach

Reinforcement learning algorithms are mostly used to solve video game problems as pong, breakout, or Pac-man. In this kind of problem, the first step is reducing the initial dimensionality of the problem by giving to the agent pre-processed information. It will decrease the number of inputs that the network has to deal with. In image processing problems, there is a large number of inputs, for example, an image of size (640x480) pixels with three channels, would have 921 600 (640x480x3) inputs. This large amount of inputs will require a huge amount of connections to process the information. Some techniques can be applied to reduce the input data in image processing as convolutive layers and pooling layers. Nevertheless, the aim of this work is finding equivalent structures by the use of evolutionary algorithms.

Because the NEAT algorithm is not well suited to deal with an enormous amount of input data, this work proposes a new incremental approach to the training phase. First genomes will train with a reduced version of the problem; it means, initial networks are trained to recognize the maximum intensity of green color in small images of (2x2) pixels with three channels. When the task is entirely learned by a genome, multiples copies of this network are grouped to solve the original problem in big images. This approach will reduce the excess load in complexity and training time over the original NEAT algorithm [7].

4.3.2.2 Asexual reproduction instead of sexual reproduction

In the original NEAT algorithm, sexual reproduction is a phase with excessive load in terms of time. This is because, when two genomes are going to be crossed, it is necessary to find common connections in both of the parents. The complexity of such operation is $\mathcal{O}(nm)$, where n is the number of connections of one genome, and m is the number of connections of the other genome. The complexity of this process is quadratic. The crossover operation is made in each generation for every new genome, that is to say, it is a recurrent procedure. Furthermore, the number of connections is increasing with each generation because of mutations; then, the time needed for this operation is growing continuously. This fact, along with a large number of inputs, is the reason why the original NEAT is not suitable for this problem.

Asexual reproduction is proposed as an alternative to reduce the excessive load of sexual reproduction because the proposed method does not need a crossover operation. Asexual reproduction is based on the replication of bacteria, archaea, plants, and fungi. These living beings can create new offspring from one single organism. This organism copies its DNA to generate clones of itself. New offspring has mutations which makes them a little bit different from their predecessor. It is easy to implement the asexual reproduction; it is necessary to choose only one parent, copy all its genes, and add a mutation to the new genome. The complexity of asexual reproduction is $\mathcal{O}(n)$, where n is the number of genes or connections of the parent. In asexual reproduction, each gene is copied from the parent to the child, and it counts as a basic operation; n copies are necessary for cloning the n connections of the parent. The complexity $\mathcal{O}(n)$ of asexual reproduction is clearly an advantage compared to the complexity $\mathcal{O}(nm)$ of sexual reproduction.

4.3.2.3 Changes for improving the fitness in the population of genomes

- **Bias and activation function:**

The original paper [7] does not mention any specific activation function, but in this work, the ramp function will be used. Figure 4.7 shows a graphic representation of the ramp function. The Equations 4.10 and 4.9 show how to calculate the output of every single neuron in the neural network. Where x_j is the output of the j neuron, w_{ij} is the weight between the neurons i and j , b_j is the bias of the neuron j , and s_j is the synapsis of the neuron j . The inputs of the neural network are images, and the output is calculated by using equations 4.10 and 4.9 repeatedly until reaching the neurons in the output layer.

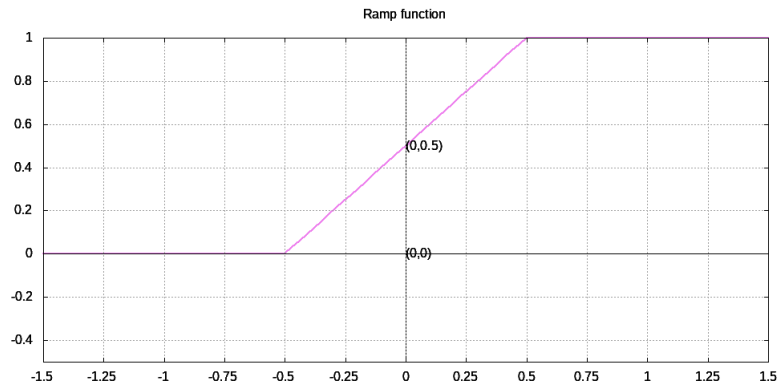


Figure 4.7: Ramp activation function

$$s_j = \left(\sum_{i=1}^n x_i \cdot w_{ij} \right) - b_j \quad (4.9)$$

$$x_j = \begin{cases} 0 & \text{if } s_j < -0.5 \\ 1 & \text{if } s_j > 0.5 \\ s_j + 0.5 & \text{otherwise} \end{cases} \quad (4.10)$$

- **Multiple tests:**

The fitness of all the networks in the population is measured in each generation. If every genome is tested just one time, a genome could get a high fitness just by good luck and survive. The present work proves each genome many times in order to minimize this effect. The number of tests can be adjusted using a hyper-parameter called *test_numbers*.

- **Death sentence and age as reproductive parameter:**

During the experiments of this work, it was observed that some genomes (networks) optimize their solution to the problem by always giving the same output, independently of what data is inputting. This kind of behavior ensures good fitness, because most of the time, the answer is right or almost right, so these genomes survive for many generations. However, this kind of conduct is useless, so in the proposed work, if a genome answers the same solution to multiple tests, it will be deleted, independently of its fitness. This is called a *death_sentence*

In each generation, genomes have the chance to pass to the next generation without mutations, if they have enough fitness. A genome which has survived for many generations must be good at doing the assigned task because it has gotten a good fitness many times. For this reason, these old genomes should have a higher reproductive chance. In the present work, each genome has a parameter called *Age* to show the generations that the genome has survived. Genomes with more age have more chances to reproduce.

- **Delete deactivated connections:**

In the original NEAT algorithm, when a connection is split into two new ones (**mutation add neuron**), the initial neural relation does not disappear, but it is deactivated. Later, mutations could turn the link on. This fact generates unnecessary connections and excessive growth in the size of genomes. For this reason, in this work, when a connection is split, it is deleted, and the new ones replace it.

4.4 Design and assembly of the proposed mini-robot system

The self-driving mini-robot system is illustrated in Figure 4.8. The Arduino is the central element exploited to build the mini-robot. It sends the images took by the camera to the computer. The Arduino also receives

commands as a pair of velocities (v_1 and v_2), as explained in the section 4.3.1.1, these speeds control the movement of the mini-robot. The computer stores and execute the proposed NEAT-SDCR algorithm. It has 7GB of RAM, an i7-4500U processor, and 4 cores of 1,80GHz. The computer is connected to the mini-robot via Bluetooth. As the auxiliary hardware, a camera is used to capture images and send them to the computer as input to the NEAT-SDCR algorithm. The designed system is visible in Figure 4.9.

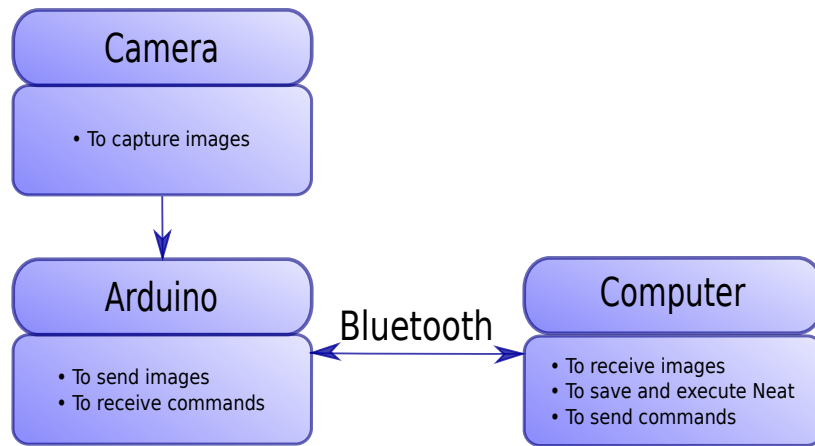


Figure 4.8: Top-level architecture of the proposed self-driving mini-robot system for indoor navigation.

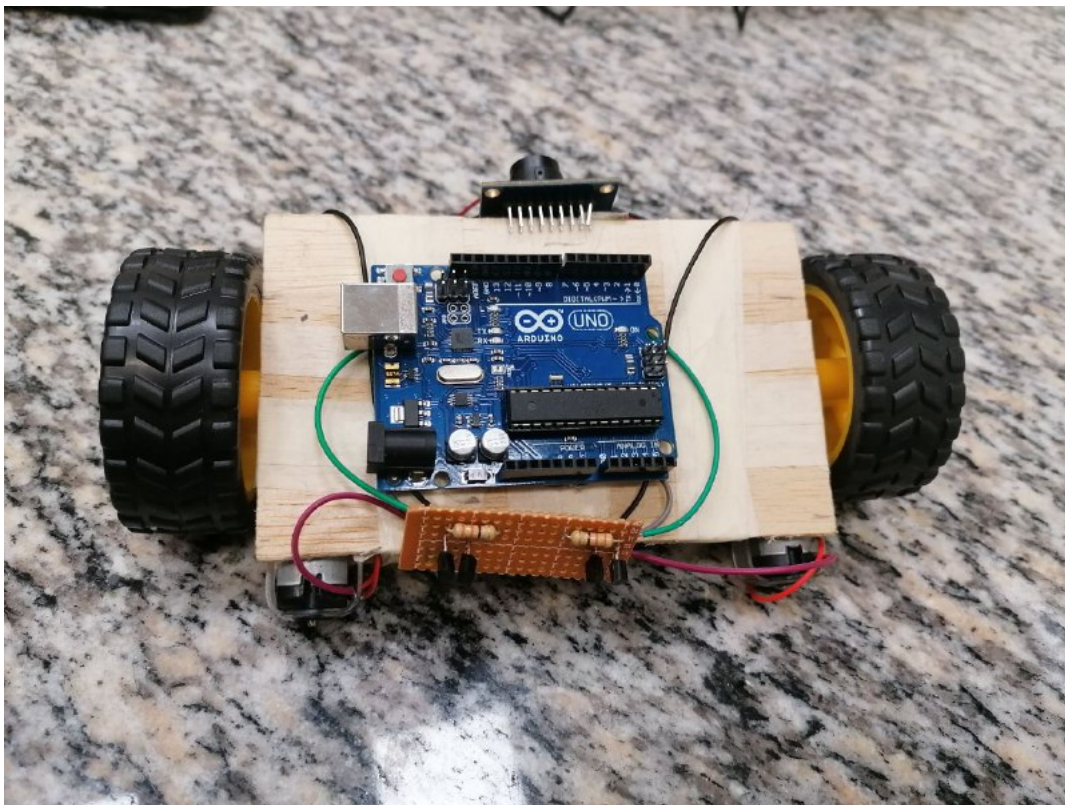


Figure 4.9: Prototype

The hardware of the mini-robot, as well as the assembly process, is described in the following sections.

4.4.1 Hardware design

The mini-robot was created using Arduino because it provides many services. Arduino sensors and actuators are easy to integrate, and they are cheap. Furthermore, there is good documentation on the internet, and there is a big community of developers. The main hardware components are shown in Figure 4.10, and they are also detailed in the list below.

- wires male/male and wires male/female
- 2 resistors of 330 ohms, 2 resistors of 4.7 kilo-ohms, 2 resistors of 10 kilo-ohms.
- 2 transistors 2N2222A
- 2 Arduino tyre wheels
- 2 Arduino gear motors of 3v - 12v
- 2 batteries of 9v
- 1 module Bluetooth hc-05 of 4v - 6v
- 1 Arduino UNO
- 1 Camera ov7670 of resolution 640x480 VGA, voltage of 3v, maximum Zhen rate of 30fps.

The hardware of the mini-robot is designed to fulfill three important purposes; controlling the wheels, controlling the Bluetooth module, and controlling the camera. The first purpose of the mini-robot hardware is controlling the speeds of the wheels; these speeds control the movement of the mini-robot and its rotation angle. The circuit diagram necessary to control the speeds of the wheels is detailed in Figure 4.11. The second purpose of the mini-robot hardware is controlling the Bluetooth module; it is used to send images to the computer and to receive commands from it. The circuit diagram necessary to control the Bluetooth module is detailed in Figure 4.11.

The circuit diagram for controlling the speeds of the wheels and the circuit diagram for controlling the Bluetooth module are fused because the wiring is not so complex. Nevertheless, the wiring needed to control the camera is so complex, and for this reason, it will be represented in another figure. The third purpose of the mini-robot hardware is controlling the camera; it is useful to get images inputs for the neural network. The circuit diagram necessary to control the camera is detailed in Figure 4.12. The aim of the next section is explaining the functioning of the circuits that control the movement to the robot.

4.4.2 Sending signals to gears motors to control movement

The speed v_1 is sent to the left motor by the pin 3 of the Arduino board using 5v. This signal passes through a resistor of 330Ω to reduce the voltage in the base of the transistor. The transistor itself acts as a door for voltage, a high voltage in its base will allow high pass voltage through the transistor. In this way, transistors can work as amplifiers of voltage. The battery, the gear motor, and the transistor form a closed circuit. When a high voltage is sent from the pin 3, the transistor allows passing more voltage through the circuit battery-motor, this will increase the speed in the motor. In this way, the speed of the wheel is controlled by the amount of voltage sent by pin 3. The operation of the right wheel is exactly the same, but using the pin 5.

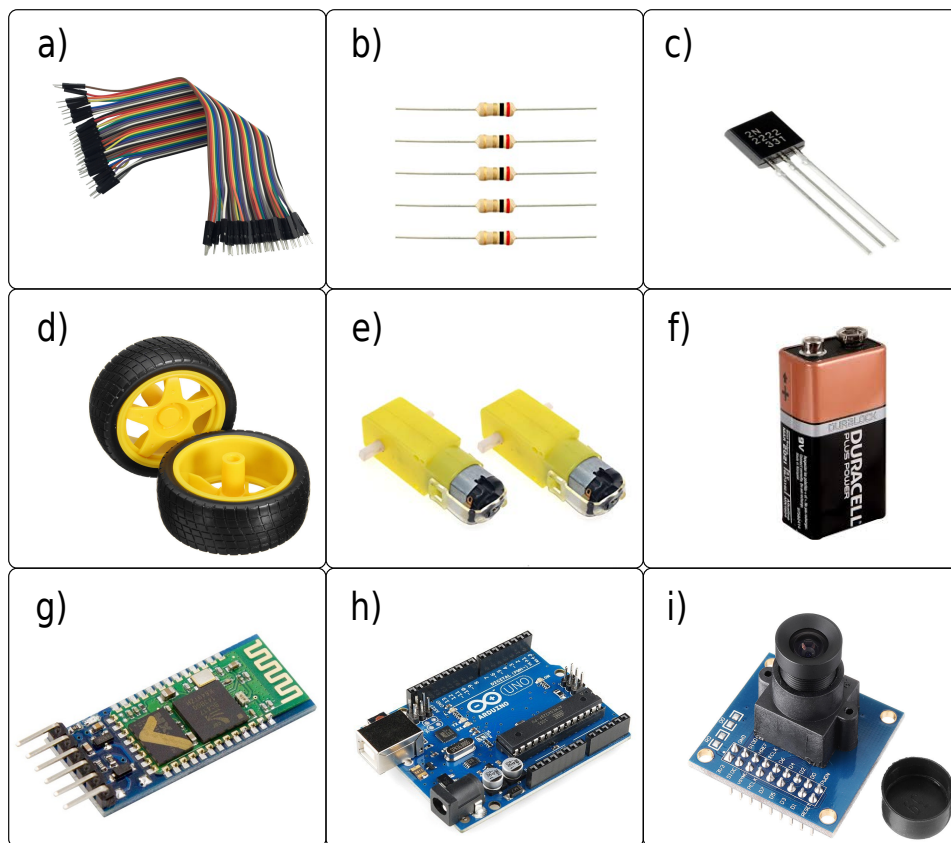


Figure 4.10: a) Wires, b) resistors, c) transistors, d) Arduino tyre wheels, e) Arduino gear motors, f) batteries, g) Arduino bluetooth module hc-05, h) Arduino UNO board, i) Arduino Camera ov7670

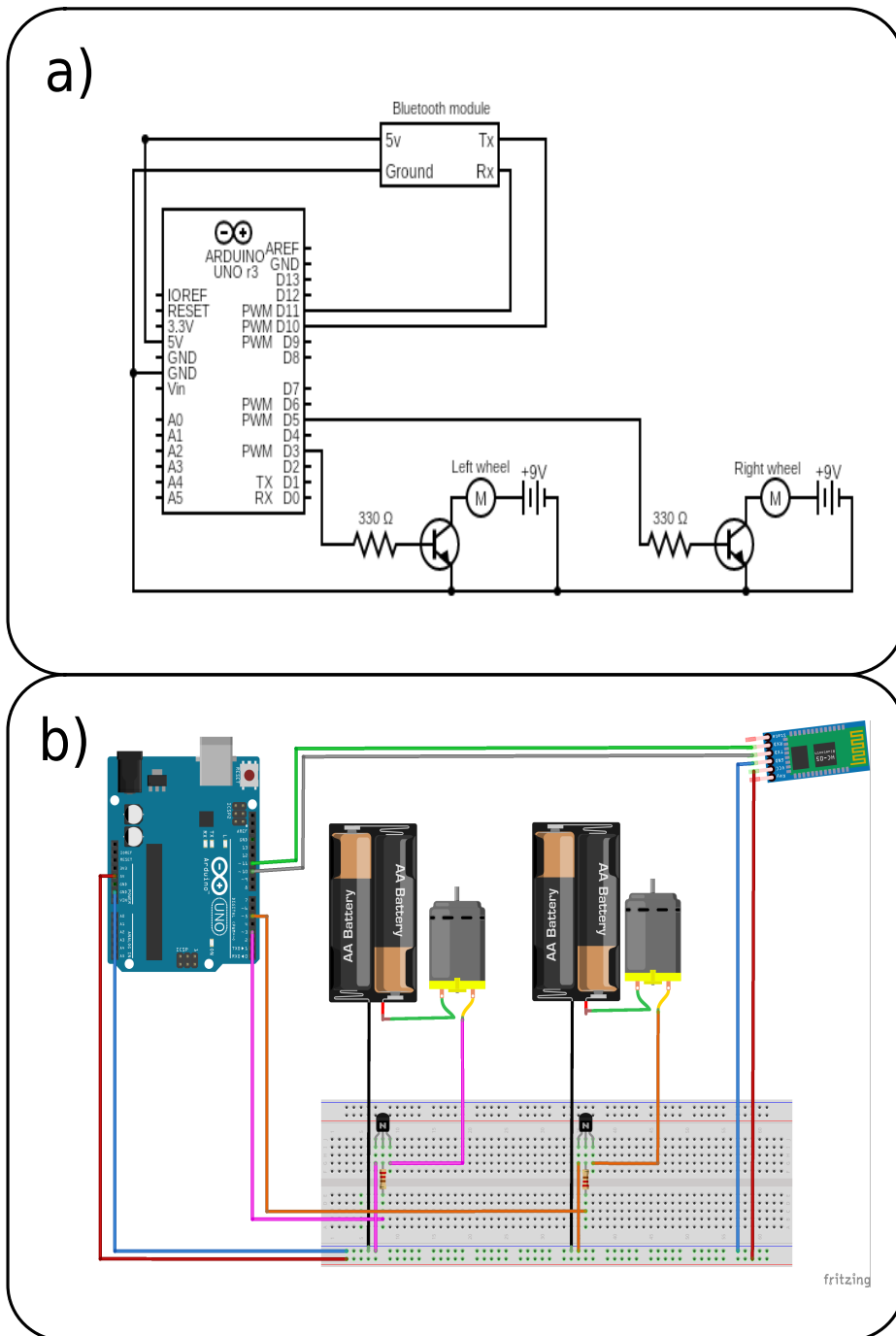


Figure 4.11: a) Circuit diagram of the bluetooth and the wheels components, b) Pictorial circuit diagram of the bluetooth and the wheels components

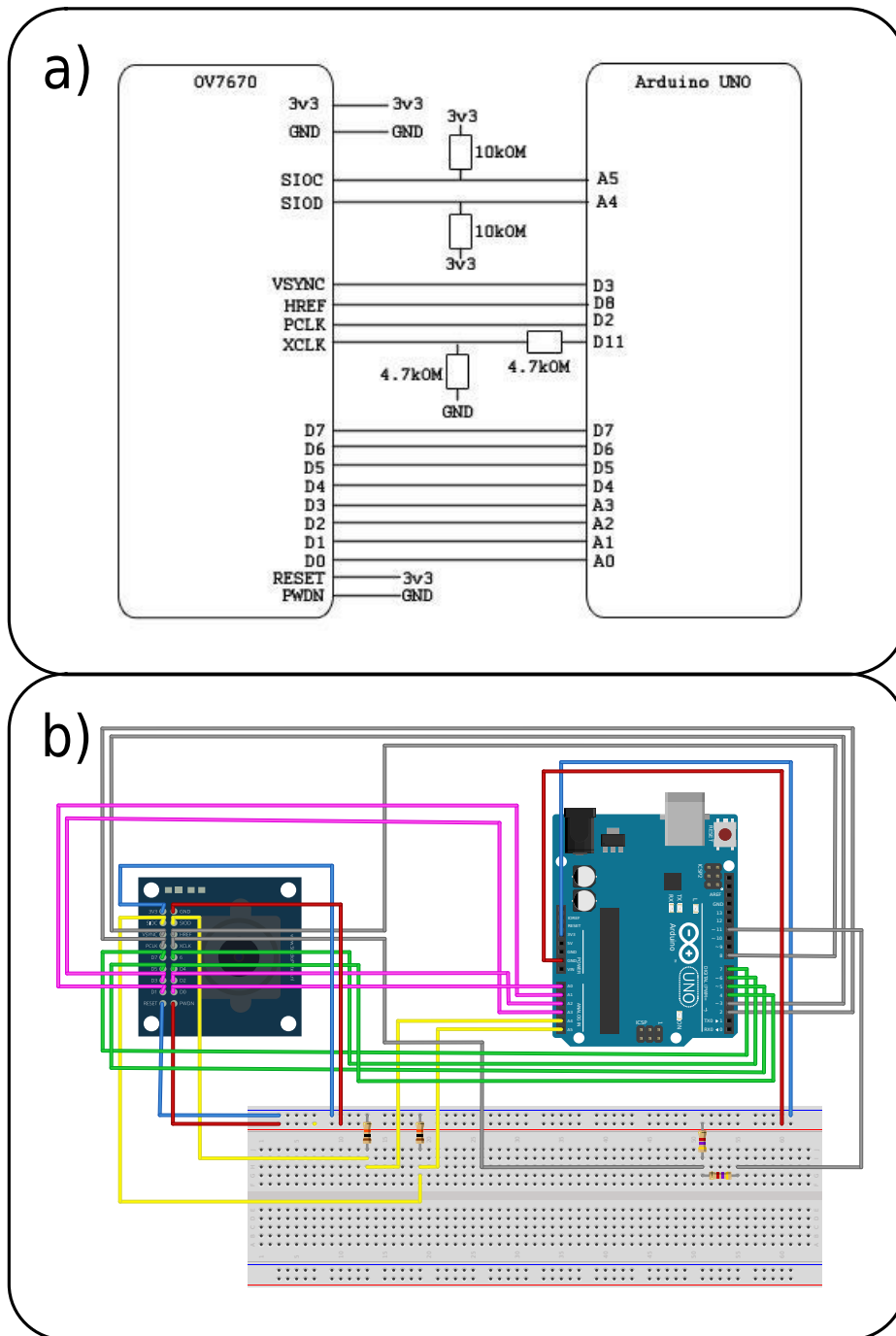


Figure 4.12: a) Circuit diagram of the camera components, picture extracted from [27], b) Pictorial circuit diagram of the camera components

4.5 Implementation of the mini-robot and computer communication

The aim of this section is to explain the communication process between the camera, the Arduino board, and the computer. Figure 4.8 shows a schematic representation of this communication.

4.5.1 Sending images from the camera to the arduino

The diagrams in Figure 4.12 shown how to connect the pins in the camera OV7670 to the pins in the Arduino board. These connections allow image data to pass from the module OV7670 to the Arduino board, and also, they serve as a power supply for the module OV7670. Arduino is an open software community, and for this reason, there are many tutorial codes for making a connection OV7670-Arduino on the internet; it is remarkable the code presented in [28] that will be used for this work. The code itself acts as a software library; therefore, there is no reason for explaining it in this work.

4.5.2 Sending image data and receiving commands (Mini-robot)

The mini-robot has to receive wireless signals from the computer by Bluetooth; these signals are in the form of a pair of numbers (v_1, v_2) . These are used to control the speed of the left wheel v_1 and the right wheel v_2 . First, it is necessary to configure the Bluetooth module using AT commands. The more critical parameters that have to be set are the name, the password, and the role(server, master) of the Bluetooth module. The commands are:

```
1 AT + NAME = <name>
2 AT + PSWD = <password>
3 AT + ROLE = <1,0>           //1 for master, 0 for slaves
```

The first command is useful to set the name whereby other devices will recognize the Bluetooth module. The second command is helpful to set the password that other devices will have to use in order to confirm the connection with the Bluetooth module. The third command is used to set the role of the Bluetooth module; this can be a server or master. The Bluetooth module is the server, and the master is the computer because it starts the transmission of data.

The Bluetooth module has four pins. The first one has to be connected to 5v. The second one has to be connected to the ground. The third one is the transmitter Bluetooth pin, so it has to be connected to an Arduino pin set as the receiver. The fourth one is the receiver Bluetooth pin, so it has to be connected to an Arduino pin set as the transmitter. The speeds (v_1, v_2) are receiver as a chain of UTF-8 strings. The Arduino code will convert to integer these speeds.

4.5.3 Receiving image data and sending commands (Computer)

The computer has the role of receiving images from the mini-robot, executing the neural network developed by the NEAT algorithm, and sending commands to the mini-robot. Bluetooth technology was used to perform this communication; it is wireless. The library *bluetooth-serial-port* of Node.js was used to perform the communication in the side of the computer; this library allows receiving data and sending back commands to the mini-robot.

Chapter 5

Experimental setup

This chapter has the aim of explaining the conditions under which the experiments presented in the section 6 were performed. All experiments use the same designed system described in section 4.4. The experiments presented in the section 6 were carried on a computer with intel core i7-4500U processor, 7GB of RAM, and 4 cores of 1,80GHz, the code is not parallelized, so just one core was used. All the experiments were repeated 10 times.

5.1 Parameter settings

This section describes all the values of hyper-parameters of the NEAT-SDCR algorithm. These values were used to perform all the experiments detailed in the chapter 6. Tables 5.1, 5.2, 5.3 shown a summary of the parameter settings tuned to achieve high accuracy.

5.1.1 Hyper-parameters for mutations

As explained in the subsection 3.2.2.3, when a new genome is born, it has the probability of developing some mutations. The parameter settings related to these mutations are explained in the list below and in the table 5.1.

- **Chance mutation add node:** It is the probability that a new genome(network) develops a new node(neuron) in its genome because of a mutation. This parameter is set to 0.2, which means that 20% of the new genomes will be born with a new neuron.
- **Chance mutation add connection:** It is the probability that a new genome develops a new connection in its genome because of a mutation. This parameter is set to 0.3, which means that 30% of the new genomes will be born with a new connection.
- **Chance mutation change weight :** It is the probability that a new genome changes the weights of its connections because of a mutation. This parameter is set to 0.9, which means that 90% of the new genomes will change its weights. This value has to be high because of two main reasons. The first reason is that this parameter controls the optimization of the connection weights, and it is very important for improving the accuracy of the model. The second reason is that, if the three possible mutations are not present in the new genome, then, the new network will be a copy of his parent, to avoid that, one of the mutations has to be present in the most cases. This mutation can change not only the weight of the connections but also the bias and the activation values of the neurons.
- **Variation weights:** This hyper-parameter explains how much the weight of a connection can change because of a mutation. It is set to 0.1, which means that after the mutation, the previous weight and the new weight will have a difference of less than 0.1.
- **Variation bias:** This hyper-parameter explains how much a bias value can change because of a mutation. It is set to 0.1, which means that after the mutation, the previous bias and the new bias will have a difference of less than 0.1.

- **Variation activation:** This hyper-parameter explains how much an activation value can change because of a mutation. It is set to 0.1, which means that after the mutation, the previous activation value and the new activation value will have a difference of less than 0.1.

5.1.2 General hyper-parameters

The next parameters are related to the mortality, the reproduction, and the test of the genomes, a summary of these parameter settings is shown in the table 5.2.

- **Population mortality:** It is the percentage of the population that will be eliminated in each generation. It is set to 0.75, which means that 75% percent of the population will die in each generation, and just the 25% of the genomes with the best fitness will survive to the next generation; also, these survivors will have offspring to replace the eliminated genomes.
- **Age importance:** As explained in the section 4.3.2.3, *age importance* contributes to the reproductive chance of the genomes. A high *age importance* value will cause those ancient genomes will have more chance of reproduction, and consequently, its genes will spread through the population. A low *age importance* will cause that the algorithm does not consider the age of the genomes at assigning new offspring.
- **Test numbers:** It is the number of tests that are applied to each genome in order to get its accuracy. It is set to 50, which means that each genome is tested 50 times in each generation. It is explained in more detail in subsection 4.3.2.3.
- **Green probability:** The images used in the training stage are generated using the Procedure [1], the parameter *Green probability* controls the percentage of the color green in the images generated by this procedure. It is set to 0.7, which means that 70% of the image will be covered by some tone of green.

5.1.3 Hyper-parameters for the car

The following parameters are used to adjust the physical characteristics of the car(mini-robot), these parameters are used in the equations 4.4 and 4.8 to model the movement of the mini-robot. A summary of these parameter settings is shown in table 5.3.

- **Max velocity:** This value shows the maximum velocity that the mini-robot can reach, it is set to 10 cm/s.
- **Reaction time:** As explained in the section 4.3.1, *reaction time* shows the time that v_1 and v_2 speeds have to be applied in order to obtain the rotation angle α in the car. It is set to 1 second which means that the robot will take a new decision each second.
- **Car length:** It is the length of the car as explained in Figure 4.5, it is set to 10 cm.
- **Max angle:** As explained in Figure 4.6, the *max angle* parameter is related to the visibility of the camera, it is set to 45°

Hyper-parameters for mutations					
chance mutation add node	chance mutation add connection	chance mutation change weight	variation weights	variation bias	variation activation
0.2	0.3	0.9	0.1	0.1	0.1

Table 5.1: Hyper-parameters for mutations

General hyper-parameters			
population mortality	age importance	test numbers	green probability
0.75	2	50	0.7

Table 5.2: General hyper-parameters

Hyper-parameters for the car			
max velocity	reaction time	car length	max angle
10	1	10	45

Table 5.3: Hyper-parameters for the car

The *hyper-parameters for mutation* were adjusted in trial and error experiments, in general, all these parameters must have little values, with the exception of the hyper-parameter *change mutation change weight*. This is because, in general, neural networks improve accuracy better by changing the weights of its connections rather than by changing its topology [7]. For that reason, there have to be a few structural mutations and a lot of mutations in the weights of the connections in order to improve the accuracy of networks. Therefore, any small value in these parameters should work fine. Analyze how small these values have to be in order to obtain the maximum benefit of NEAT is beyond the scope of this work. The *hyper-parameters for the car* were adjusting by taking into account the physical characteristics of the mini-robot. The *general hyper-parameters* were adjusted by trial and error experiments; the values used in this work are not tuned. Therefore, this could be explored in future works.

5.2 Experiments

The following experiments are focusing on measuring the performance of the proposed changes on the NEAT-SDCR algorithm. The first set of experiments 5.2.1 will compare the sexual reproduction method proposed in the original NEAT algorithm [7] to the asexual reproduction method proposed in this work (NEAT-SDCR). The second set of experiments 5.2.2 will test the performance of the reduction in the size of the images proposed on (NEAT-SDCR) to reduce the dimensionality of the inputs of the problem. The original NEAT algorithm [7] does not propose any modification to reduce the dimensionality in the inputs. The third set of experiments 5.2.3 will test the effect in the change of the number of population NEAT-SDCR algorithm. The results of these experiments will be presented in the section 6

5.2.1 Sexual reproduction vs. asexual reproduction experiments

This set of experiments aims to measure and compare the performance of sexual reproduction and asexual reproduction; these methods will be contrasted, taking into account factors as time, generations, accuracy, and size of the networks. The asexual reproduction method was proposed in the section 4.3.2.2. An image size of (2x2) pixels and a population of 500 neural networks were used in all the experiments of this set.

- **Experiment 1: Sexual reproduction vs. Asexual reproduction compared in terms of generations**
This experiment measures the increasing trend of accuracy throughout the generations. The *max generations* parameters were set to 300 because, after this generation, the accuracy does not improve largely.
- **Experiment 2: Sexual reproduction vs. Asexual reproduction compared in terms of total time**
This experiment measures the increasing trend of accuracy and the time needed to achieve that accuracy. A maximum time of 250 seconds was imposed as a restriction.
- **Experiment 3: Sexual reproduction vs. Asexual reproduction compared in terms of time per generation**
This experiment measures the change in time needed per generation throughout the generations. The *max generations* parameters was set to 300.
- **Experiment 4: Sexual reproduction vs. Asexual reproduction compared in terms of size of the networks)**
This experiment measures the average size of all the networks in the population throughout the generations. The *max generations* parameters was set to 300.

5.2.2 Variation in the image size experiments

The aim of this set of experiments is to test the performance of the reduction in the dimensionality of the problem; this approach was proposed in section 4.3.2.1. For this reason, the following two experiments will test how the accuracy and the time per generation change as the size of the image changes. Five images sizes

were taking into account, images of (2x2), images of (4x4), images of (10x10), images of (20x15), and images of (40x30) pixels. Additionally, four generations were considered, generation 25, generation 50, generation 75, and generation 100. The reason for taking four generations is that it will show how accuracy and time per generation change as the generation changes. Asexual reproduction method and a population of 500 neural networks were used in all the experiments of this set.

- **Experiment 5: Variation in the image size tested in terms of accuracy**

This experiment measures the change of accuracy when the model is used with images of different sizes.

- **Experiment 6: Variation in the image size tested in terms of time per generation**

This experiment measures the variation of the time needed per generation when the model is used with images of different sizes.

5.2.3 Variation in the population experiments

The purpose of this set of experiments is testing the effect of the population over the accuracy and the time of the model. Populations of size 50, 100, 150, 200, 250, 300, 350, 400, 450, and 500 neural networks were used in all these experiments. Additionally, four generations were considered, generation 25, generation 50, generation 75, and generation 100. Asexual reproduction method and an image size of (2x2) pixels were used in all the experiments of this set.

- **Experiment 7: Variation in the population tested in terms of accuracy**

This experiment measures the change in the accuracy of the model when it is started with different population sizes.

- **Experiment 8: Variation in the population tested in terms of time per generation**

This experiment measures the change of the time needed per generation when the model is started with different population sizes.

Chapter 6

Results

This section is focused on test the performance of the NEAT-SDCR implementation based on seven measurements, which are the method of reproduction, size of the image, initial population, generations, accuracy, time, and size of the network. The first four measurements are important parameters, and the remaining three measurements are the metrics of the NEAT-SDCR algorithm. These four parameters are important because the accuracy of the model depends largely on the tuning of these parameters, and also they are closely related to the modifications proposed in the section 4.3.2.

In the experiments of this chapter, many images are given to each genome (neural network), these networks have to complete the assigned task in each image. As explained in section 4.3.1, the performance of every network is measured by using equations 4.4 and 4.8 to determinate the column that is being pointed by the mini-car. The higher the green intensity that is being pointed, the higher the fitness the network gets. All figures showed the mean as a solid line, and the first standard deviation is represented as a colored shadow surrounding that line.

Parameters

- **Method of reproduction:** The original NEAT algorithm proposed by Stanley [7] uses sexual reproduction to create new genomes. This work (NEAT-SDCR) proposes asexual reproduction as a better option because of the speed of this method; the new method was explained in the subsection 4.3.2.2.
- **Size of the image:** The inputs of the neural networks presented in this work are images. The size of these images is represented as (height x length). *height* and *length* are two hyper-parameters that are set at the beginning of the algorithm.
- **Initial population:** This hyper-parameter defines the number of genomes existing in the population. This value does not vary through the generations. As explained in the sections 3.2.2.2 and 3.2.2.3, a new genome borns in the reproduction step to replace each dead genome. In this way, the population does not change.
- **Generations:** A generation covers the steps of the testing of a population, the elimination of the worst genomes, and the birth of the new generation, which is to say all the steps explained in the section 3.2.2. As illustrated in the Picture 3.1, the NEAT-SDCR algorithm stops its execution when the number of generations reaches the value adjusted in the *max_generations* parameter.

Metrics

- **Accuracy:** The accuracy of the model can be calculated using the equation 6.1, where f_i is the fitness obtained by the neural network according to the process explained in section 4.3.1, f_{max_i} is the maximum possible fitness, and t is the parameter *test_numbers* explained in section 4.3.2.3 and set in Table 5.2. This measure represents the performance of a genome, and it could be 100% as maximum. There are various genomes existing in the population in each generation; for this reason, there are many accuracy values, one by each genome. Only the accuracy of the best genome will be taking into account for these experiments.

$$ac = \frac{100}{t} \sum_{i=1}^t \frac{f_i}{f_{max_i}} \quad (6.1)$$

- **Time:** It can be accumulative time or time per generation. Accumulative time is measure since the start of the algorithm; on the other hand, time per generation is measure since the start of the generation. Figure 6.2 shows an accumulative time, and Figures 6.3, 6.6, and 6.8 show a time per generation. The time in all the experiments is measured in seconds.
- **Size of the network:** Every genome is composed of various genes, as explained in the section 3.2.2.1. Each gene is a connection between two neurons. The size of the network is equal to the number of genes in the genome; that is to say, the number of connections. This metric is related to the complexity of the algorithm because the more size the networks have, the more time is needed to process the data. There are various genomes existing in the population in each generation; for this reason, there are many network sizes, one by each genome. The average size of all the genomes in the population will be taking into account for these experiments.

This work proposed two major modifications to the original NEAT algorithm [7]. These modifications are explained in sections 4.3.2.1 and 4.3.2.2. This section is divided into three subsections. The purpose of the first subsection is to measure the performance of the proposed asexual reproduction modification 4.3.2.2. The purpose of the second section is to measure the performance of the proposed incremental training modification 4.3.2.1. The purpose of the third section is to measure the effect of the change in the initial population. All the experiments presented in this section were repeated ten times.

6.1 Results of sexual vs. asexual reproduction

6.1.1 Experiment 1: Sexual reproduction vs. asexual reproduction compared in terms of generations

Figure 6.1 shows that the accuracy of both methods is similar in terms of generation. Sexual reproduction is more accurate at first fifty generations, but after that, asexual reproduction becomes more accurate. Nevertheless, this difference is little significance, because the means are very close, and the standards deviation of both methods are overlapped. This result is positive because it means that the proposed asexual reproduction does not decrease the general accuracy of the method; on the contrary, asexual reproduction seems to be a little bit more accurate than sexual reproduction. Figure 6.1 shows the generations but does not show the time needed to achieve accuracy; for this reason, the next experiment considers time.

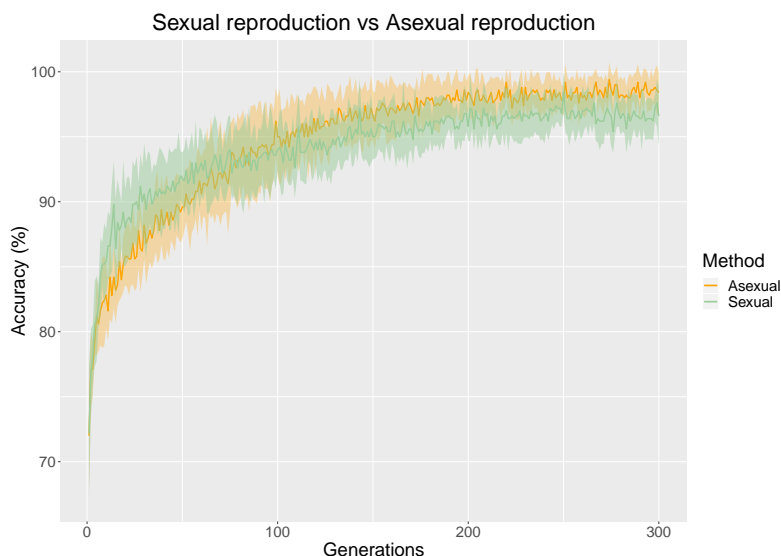


Figure 6.1: Generation vs Accuracy

6.1.2 Experiment 2: Sexual reproduction vs. asexual reproduction compared in terms of total time

This experiment is similar to the previous one, but it considers time instead of generations. In doing so, Figure 6.2 shows that Asexual reproduction reaches higher accuracy values faster than sexual reproduction. Actually, asexual reproduction reaches an accuracy of 97% in 75 seconds; on the other hand, sexual reproduction reaches an accuracy of 96% in 225 seconds. This result is positive because it shows that the asexual reproduction method is faster than sexual reproduction. This is not a surprise because, as discussed in the section 4.3.2.2, sexual reproduction needs the crossing genomes operation. The complexity of such operation is quadratic, asexual reproduction does not need a crossing operation, and that is why it is faster. Figures 6.1 and 6.2 show the same data trend in the accuracy increasing, but they differ because generations do not last the same time, it will be verifying in the next experiment.

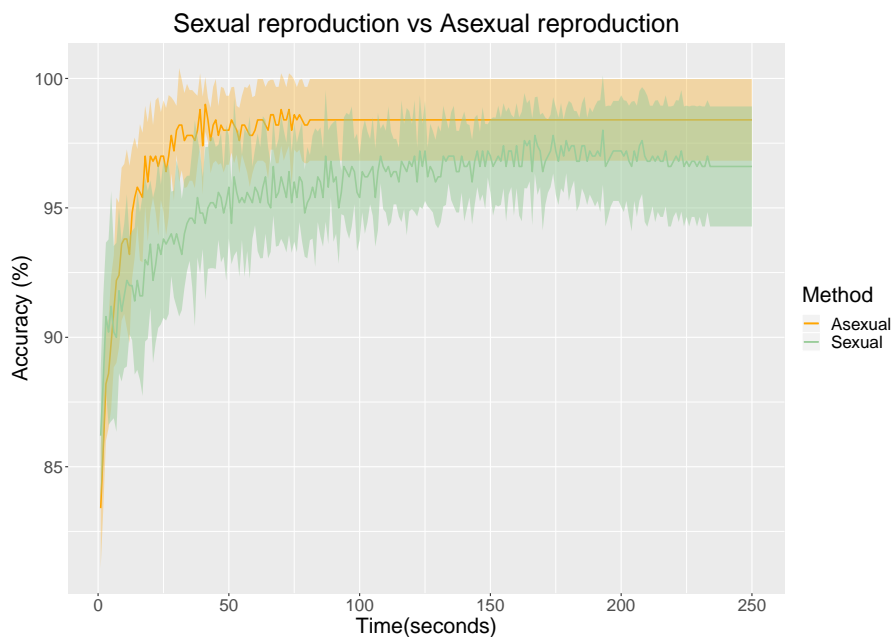


Figure 6.2: Time vs Accuracy

6.1.3 Experiment 3: Sexual reproduction vs. asexual reproduction compared in terms of time per generation

Figure 6.3 shows that the time that each generation needs is constantly increasing in both methods. However, the time needed for each generation is clearly higher in sexual reproduction than asexual, even if the standard deviation of sexual reproduction is big. Furthermore, the needed time for sexual reproduction is increasing much faster than asexual reproduction. This experiment shows that the asexual reproduction of NEAT-SDCR is more efficient in terms of time. A reason for the inefficiency of sexual reproduction is the increase in the size of networks; this will be discussed in the next experiment.

6.1.4 Experiment 4: Sexual reproduction vs. asexual reproduction compared in terms of size of the networks

The size of the network is closely related to the time needed for each generation, because the more complex the neural network is, the more time it takes to perform operations as testing or mutations in each generation. Figure 6.4 shows that the size of networks tends to increase faster using sexual reproduction than asexual reproduction. This is because sexual NEAT [7] combines the genome of both parents when they have the same fitness, this process is explained in detail in the section 3.2.2.3, and this combination process increases the network size of new generations largely. It is also remarkable the similarity between Figures 6.3 and 6.4; it supports the idea that network size and time per generation is closely related.

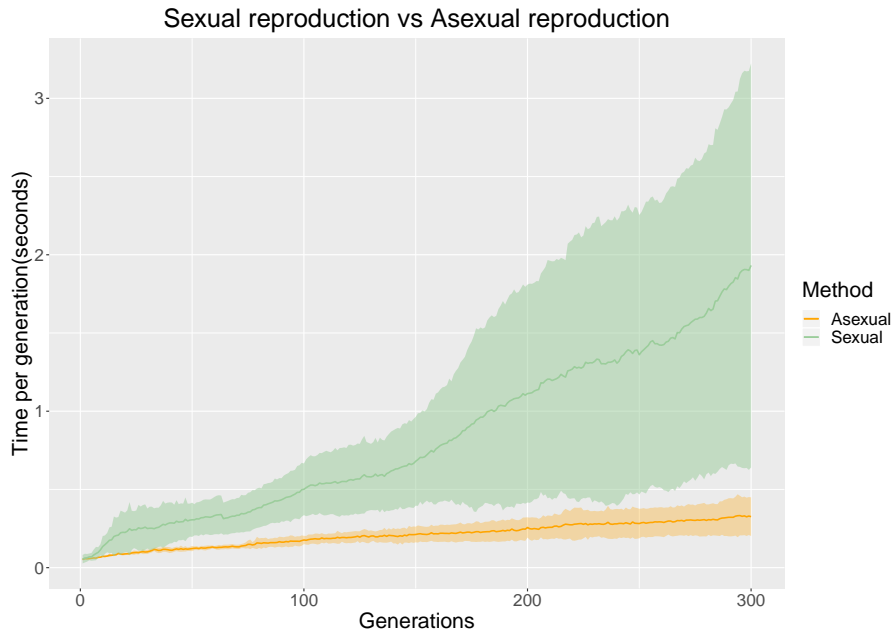


Figure 6.3: Generation vs Time per generation

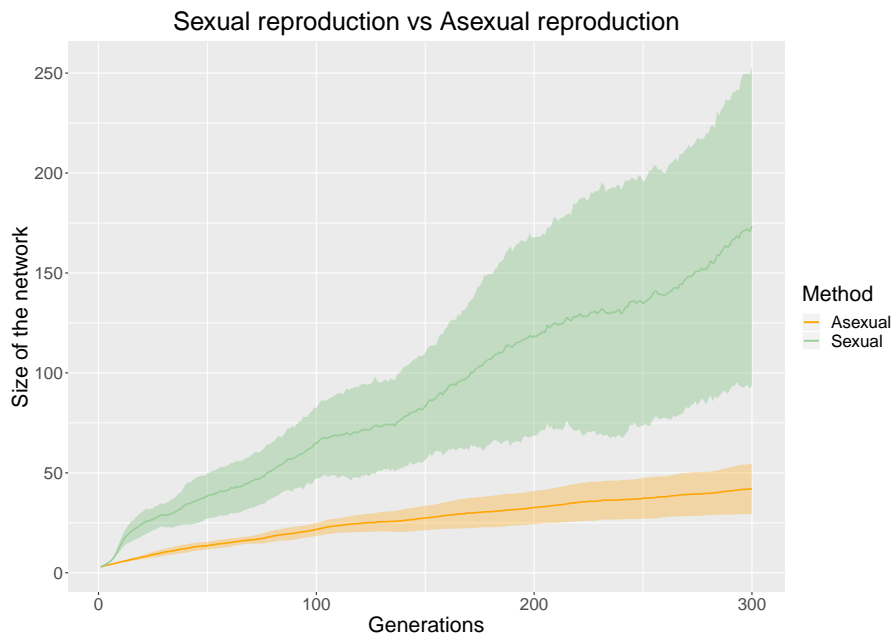


Figure 6.4: Generations vs size of the network

The four experiments presented in this section support the idea that asexual reproduction proposed in this work (NEAT-SDCR) has better performance than sexual reproduction (NEAT [7]). It is because asexual reproduction reaches higher values of accuracy faster than sexual reproduction. These experiments also support the idea that the inefficiency of sexual reproduction is due to the complexity of crossing genome operation and the fast increase in the size of networks that it causes.

6.2 Results of variation in the image size

6.2.1 Experiment 5: Variation in the image size tested in terms of accuracy

Figure 6.5 shows that the accuracy of the model tends to decrease quickly in large image sizes. The graph looks like a decreasing exponential function. Furthermore, the accuracy does not improve even as the generations pass by. For example, in an image size of (40x30), the accuracy in all the generations is almost the same. It means that it is useless to wait for many generations when the image size is large because the accuracy of the model will not advance. Figure 6.5 shows the accuracy, but it does not show the time per generation, that is the aim of the next experiment.

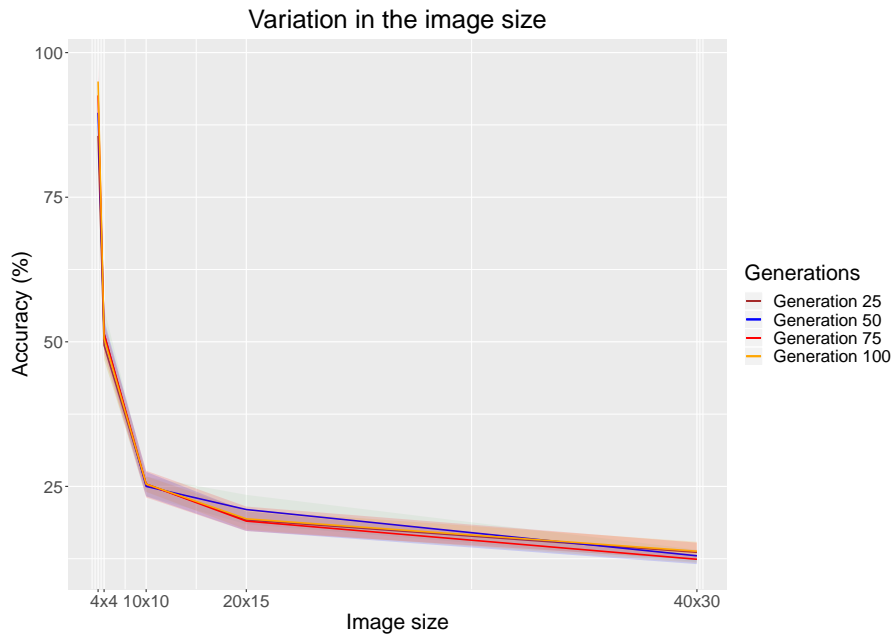


Figure 6.5: Size of the image vs Accuracy

6.2.2 Experiment 6: Variation in the image size tested in terms of time per generation

Figure 6.6 shows that the time per generation is increasing linearly as the size increases. Figure 6.6 also shows that the time needed is increasing by the pass of generations; in other words, the higher the generation is, the more time is needed to complete a generation. This result is similar to the one showed in Figure 6.3. In comparison, the time needed for the reduced version of the problem (2x2) is minimal compared to the time needed for process images of (40x30) pixels.

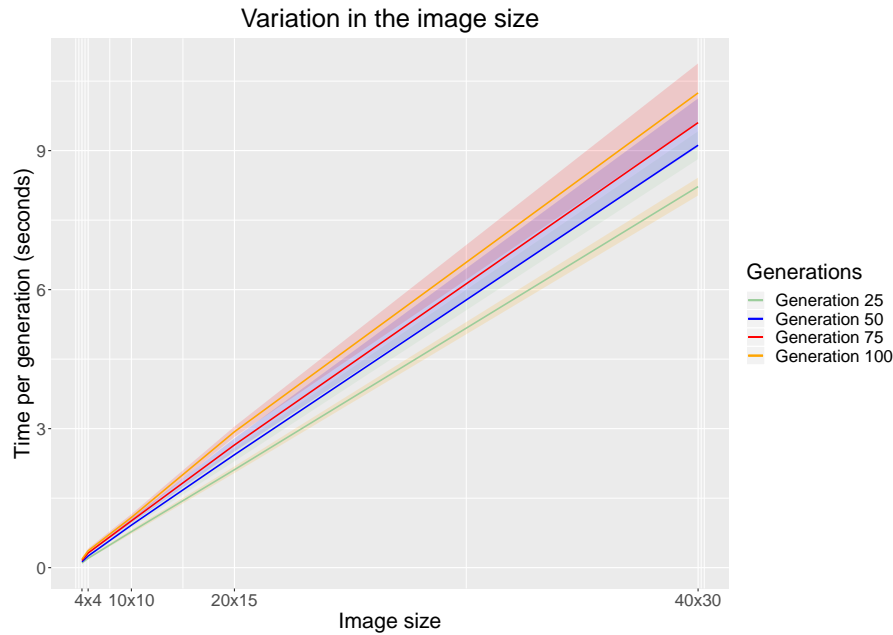


Figure 6.6: Size of the image vs Time per generation

These experiments show that the NEAT-SDCR algorithm applied to large images is very expensive in terms of time (Figure 6.6). Furthermore, the invested time is useless because the accuracy of the model is poor, and it does not improve through generations (Figure 6.6). The time needed is increasing linearly, but the accuracy is decreasing exponentially. For this reason, the incremental training approach was proposed 4.3.2.1. The camera of the mini-robot has a resolution of (640x480) pixels; it is much higher than the resolution of (40x30) showed in these experiments. For this reason, applying the NEAT-SDCR algorithm directly over the images of the camera will be impossible.

6.3 Results of change in population

The population parameter is very important in the NEAT-SDCR algorithm because of the more population available, the more capacity of search the algorithm NEAT-SDCR has. Each genome in the population represents a solution to the problem with a specific topology and weights. Then, if the population is big, a higher number of topologies and weights can be explored. Higher populations should be related to higher values of fitness in all the generations.

6.3.1 Experiment 7: Variation in the population tested in terms of accuracy

Figure 6.7 shows that in general, the accuracy tends to increase if the initial population is high. This rise in the trend is clear even if the standard deviation of the Figure is high. This result indicates that it is worth to have a bigger population because it will affect the accuracy of the model positively. The initial population also affects the time needed for each generation, that is the objective of the next experiment.

6.3.2 Experiment 8: Variation in the population tested in terms of time per generation

Figure 6.8 shows that the higher the number of the population is, the more time is needed per generation. The time needed grows up following a linear trend, so it does not increase so fast. Another remarkable fact in Figure 6.8 is that the four lines that represent generations are closer when the population is low, and they get away when the population is high. It means that a high population causes that time per generation increases faster as the generations pass by; in contrast, in low populations, the time needed per generation increases slowly as the generations pass by.

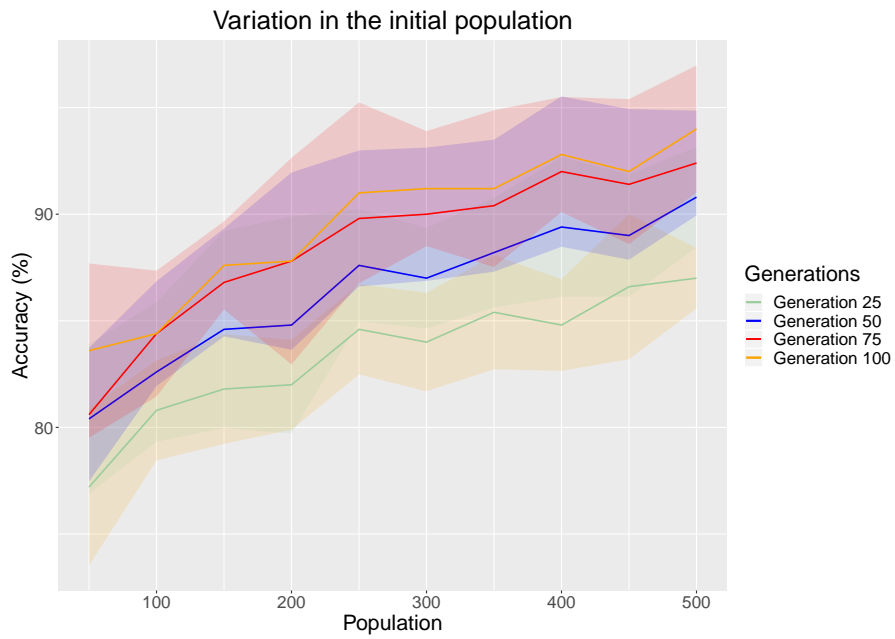


Figure 6.7: Population vs Accuracy

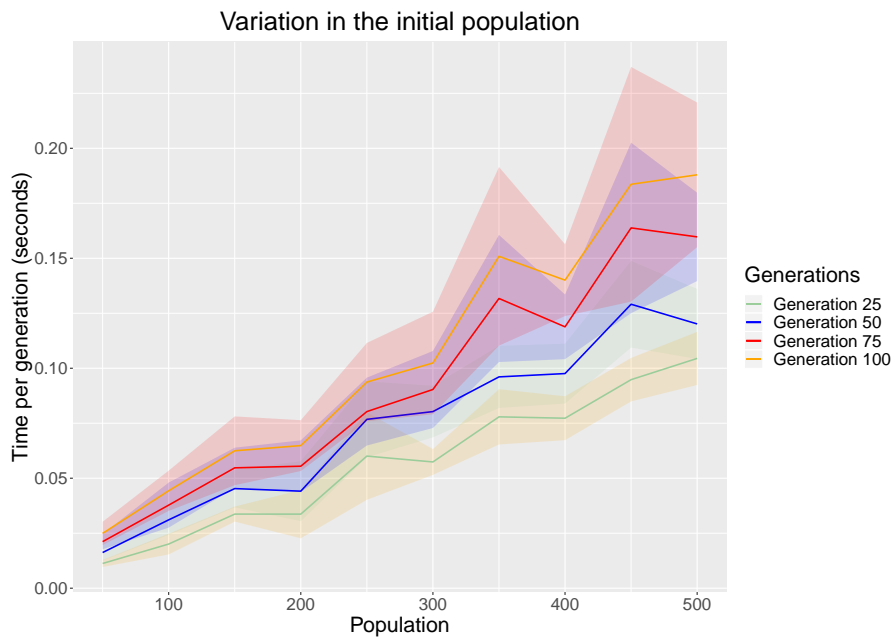


Figure 6.8: Population vs Time

The experiments presented in this section show that a good way of increasing the accuracy of the algorithm is to enlarge the population. Figure 6.7 shows that the accuracy will grow up largely and Figure 6.8 shows that the time per generation will increase, but the extra time is not very much.

Chapter 7

Conclusions and future work

This work aims to apply the NEAT algorithm to autonomous visual navigation of a mini-robot in conjunction with color recognition. This work states the following conclusions:

- In previous works, NEAT algorithm has been successfully applied in video game applications [9] and in simulated environments [8], [10], [11]. In these problems, the amount of input data is much smaller than in image processing problems. Under conditions of large input data as the one presented in image processing, the performance of the NEAT model decreases dramatically. This was proven in the experiments of the section 6.2. Particularly, Figure 6.5 shows that the accuracy of the model decreases drastically, and Figure 6.6 demonstrates that the training time is constantly increasing.
- The reduction in the performance of NEAT explained in the last paragraph is due to processing large input data that requires a very complex and large topology (neurons and connections) and further optimization in the weights of the connections. This complexity is constantly increasing as the input data increases. In the present work the camera OV7670 is used, and it has a definition of (480x640) pixels, the inputs of such a picture is $480 \times 640 \times 3 = 921\,600$. For this reason, the NEAT algorithm cannot perform well on this problem, even if the task is very simple as following green color objects. Therefore, the NEAT algorithm cannot be applied directly over this problem; therefore the NEAT-SDCR algorithm is proposed. Also, It is advisable to combine the NEAT algorithm with some convolutional approach in future works of image processing.
- Another way of reducing the complexity of the problem is by using the approach proposed in section 4.3.2.1. It is an incremental approach, and the aim is training the NEAT algorithm with a reduced version of the problem, when the task is completely master, the resulting neural structure can be combined with many copies of itself in order to solve the big problem. This approach was used in this work to control the robot. The present work just covers the initial stage; it means training the network using reduced images. The stage of network combination is not covered in this project, and it can be explored in future works. The proposed incremental approach has some major drawbacks, for example, the combination could be a non-trivial task, and there are problems where an incremental approach cannot be applied.
- The other major modification proposed in this work is based on asexual reproduction instead of sexual reproduction. The experiments of the section 6.1 show that asexual reproduction is more efficient in terms of training time than sexual reproduction. Particularly, Figure 6.3 shows that training time is much less in asexual reproduction. It is because sexual reproduction needs a time-consuming operation for crossing genomes; it is explained in detail in section 4.3.2.2. The other reason for the inefficiency of sexual reproduction is that it causes unnecessary growth in the size of the networks, as confirmed in Figure 6.4. Finally, the experiments of the section 6.3 show that a good way to enhance the accuracy of the NEAT model is by increasing the initial population, this also raises the training time of the algorithm, but the extra time is not so much.

References

- [1] H. Sadeghi, S. Valaee, and S. Shirani, "A weighted knn epipolar geometry-based approach for vision-based indoor localization using smartphone cameras," in *2014 IEEE 8th Sensor Array and Multichannel Signal Processing Workshop (SAM)*, 2014.
- [2] T. Ueno, Y. Nakamura, T. Takuma, T. Shibata, K. Hosoda, and S. Ishii, "Fast and stable learning of quasi-passive dynamic walking by an unstable biped robot based on off-policy natural actor-critic," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2006, pp. 5226–5231.
- [3] T. Matsubara, Jun Morimoto, Jun Nakanishi, M. Sato, and K. Doya, "Learning sensory feedback to cpg with policy gradient for biped locomotion," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, April 2005, pp. 4164–4169.
- [4] R. Tedrake, T. W. Zhang, and H. S. Seung, "Stochastic policy gradient reinforcement learning on a simple 3d biped," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, Sep. 2004, pp. 2849–2854 vol.3.
- [5] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 3, April 2004, pp. 2619–2624 Vol.3.
- [6] A. El-Fakdi, M. Carreras, and E. Galceran, "Two steps natural actor critic learning for underwater cable tracking," in *2010 IEEE International Conference on Robotics and Automation*, May 2010, pp. 2267–2272.
- [7] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *The MIT Press Journals*, vol. 10, no. 2, pp. 99–127, 2002. [Online]. Available: <http://nm.cs.utexas.edu/?stanley:ec02>
- [8] Shengbo Xu, H. Moriguchi, and S. Honiden, "Sample efficiency analysis of neuroevolution algorithms on a quadruped robot," in *2013 IEEE Congress on Evolutionary Computation*, June 2013, pp. 2170–2177.
- [9] J. K. Olesen, G. N. Yannakakis, and J. Hallam, "Real-time challenge balance in an rts game using rtneat," in *2008 IEEE Symposium On Computational Intelligence and Games*, Dec 2008, pp. 87–94.
- [10] J. Lowell, S. Grabkovsky, and K. Birger, "Comparison of neat and hyperneat performance on a strategic decision-making problem," in *2011 Fifth International Conference on Genetic and Evolutionary Computing*, Aug 2011, pp. 102–105.
- [11] G. Nitschke, "Behavioral heterogeneity, cooperation, and collective construction," in *2012 IEEE Congress on Evolutionary Computation*, June 2012, pp. 1–8.
- [12] A. Rodriguez-Ramos, C. Sampedro, H. Bavle, I. G. Moreno, and P. Campoy, "A deep reinforcement learning technique for vision-based autonomous multicopter landing on a moving platform," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018, pp. 1010–1017.
- [13] T. D'Orazio, G. Cirirelli, and A. Distanto, "Development of a vision-based behavior by reinforcement learning," in *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028)*, vol. 5, Oct 1999, pp. 453–457 vol.5.
- [14] A. Jeerige, D. Bein, and A. Verma, "Comparison of deep reinforcement learning approaches for intelligent game playing," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan 2019, pp. 0366–0371.

- [15] G. H. W. Gebhardt, K. Daun, M. Schnaubelt, and G. Neumann, "Learning robust policies for object manipulation with robot swarms," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 7688–7695.
- [16] A. J. L. Correia and W. R. Schwartz, "Oblique random forest based on partial least squares applied to pedestrian detection," in *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016.
- [17] R. Waranusast, V. Timtong, N. Bundon, and C. Tangnoi, "A computer vision approach for detection and counting of motorcycle riders in university campus," in *2014 International Electrical Engineering Congress (iEECON)*., 2014.
- [18] D. D. Luong, S. Lee, , and T.-S. Kim, "Human computer interface using the recognized finger parts of hand depth silhouette via random forests," in *2013 13th International Conference on Control, Automation and Systems (ICCAS 2013)*, 2013.
- [19] J. Deng, J. Li, and X. Zou, "Extraction of litchi stem based on computer vision under natural scene," in *2011 International Conference on Computer Distributed Control and Intelligent Environmental Monitoring*, 2011.
- [20] M. S. Allili, N. Bouguila, and D. Ziou, "Finite generalized gaussian mixture modeling and applications to image and video foreground segmentation," in *Fourth Canadian Conference on Computer and Robot Vision (CRV '07)*, May 2007, pp. 183–190.
- [21] J. Xiao and X. Huang, "Research of moving target detection and tracking based on background difference and camshift," in *2018 Chinese Control And Decision Conference (CCDC)*, June 2018, pp. 3014–3019.
- [22] J. Cheng, J. Yang, Y. Zhou, and Y. Cui, "Flexible background mixture models for foreground segmentation," *Image and Vision Computing*, vol. 24, no. 5, pp. 473 – 482, 2006. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0262885606000473>
- [23] R. Jangra and R. Kait, "Analysis and comparison among ant system; ant colony system and max-min ant system with different parameters setting," in *2017 3rd International Conference on Computational Intelligence Communication Technology (CICT)*, Feb 2017, pp. 1–4.
- [24] C. Kambhampati, F. Garces, and K. Warwick, "Approximation of non-autonomous dynamic systems by continuous time recurrent neural networks," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 1, July 2000, pp. 64–69 vol.1.
- [25] J. M. Abdullah and T. Ahmed, "Fitness dependent optimizer: Inspired by the bee swarming reproductive process," *IEEE Access*, vol. 7, pp. 43 473–43 486, 2019.
- [26] N. T. Siebel and G. Sommer, "Evolutionary reinforcement learning of artificial neural networks," *International Journal of Hybrid Intelligent Systems*, vol. 4, pp. 171–183, 2007. [Online]. Available: <https://dl.acm.org/citation.cfm?id=1367016>
- [27] mybotic. (2018) Ov7670 arduino camera sensor module framecapture tutorial. [Online]. Available: <https://www.instructables.com/id/OV7670-Arduino-Camera-Sensor-Module-Framecapture-T/>
- [28] ComputerNerd. (2016) ov7670-no-ram-arduino-uno. [Online]. Available: <https://github.com/ComputerNerd/ov7670-no-ram-arduino-uno>

Appendices

.1 Appendix 1.

This document, a presentation with this topic and the source code are available for free on the website <https://sites.google.com/site/degreethesislorenaguachi/2020-joseph-gonzalez-self-driving-mini-robot-using-neat-algorithm>, the code is also available on gitgub on the link <https://github.com/JosephGonzalez96/NEAT-SDCR>.