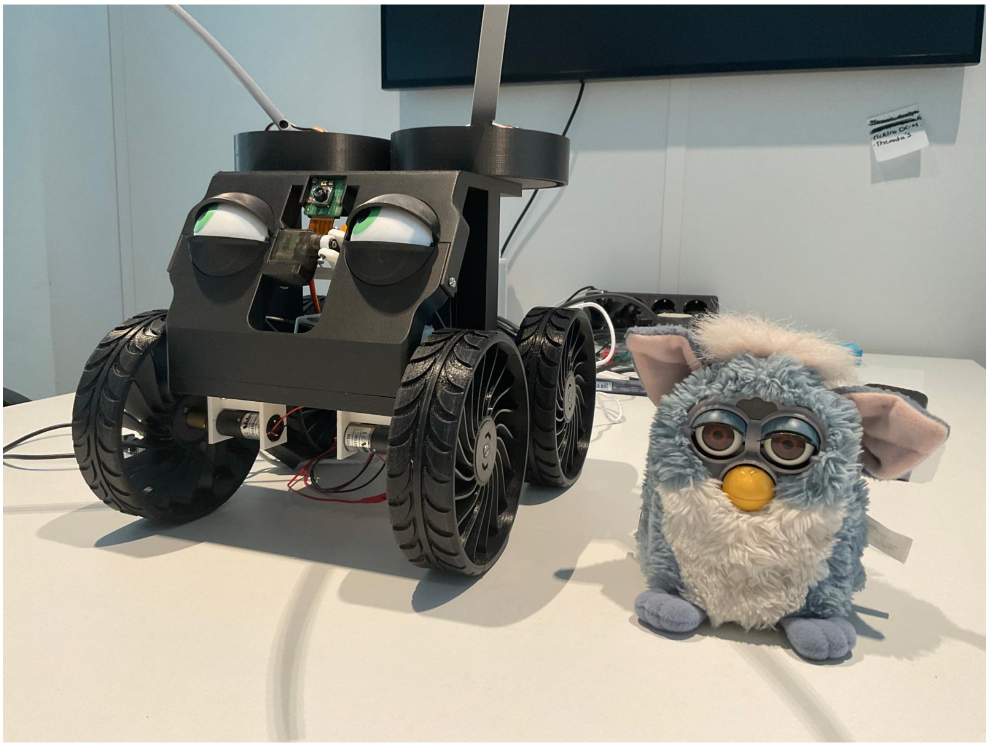
Design of a Small Swarming Robot Showing Intuitive Human Interaction

Master's Thesis



Aart Rozendaal

Design of a Small Swarming Robot Showing Intuitive Human Interaction

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

Aart Rozendaal
born in Eindhoven, the Netherlands

Computer and Embedded Systems Engineering
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
`www.ewi.tudelft.nl`

Cover picture: The swarm-robot symbiote FLIP next to a Furby

Design of a Small Swarming Robot Showing Intuitive Human Interaction

Author:	Aart Rozendaal
Student id:	4870255

Abstract

This thesis presents the design and implementation of a small, bio-inspired symbiotic robot capable of intuitive interaction with humans, aimed at contributing to the field of Human-Swarm Interaction (HSI). Drawing inspiration from nature, the system integrates biologically relevant sensing, like vision and hearing, with expressive actuation to interpret and respond to human presence and behavior in real time.

The robot, named FLIP (Friendly Logic-based Interactive Presence), utilizes a camera and stereo microphones to detect faces, bodies, emotions, and loud sounds, which are interpreted through lightweight, embedded processing. A custom decision-making architecture using a Finite State Machine (FSM) maps these inputs to one of several emotional states, which are then expressed using eyes, eyelids, antennae, and motion designed to evoke intuitive and relatable responses from human observers.

The project combines disciplines across embedded systems, robotics, and interaction design. A modular and accessible design approach was adopted to encourage reproducibility and educational use. The robot was designed as a case study for the TU Delft Cyber Zoo, where its ability to engage with the public was evaluated against technical and interactional requirements.

This work contributes a novel, low-cost platform for studying human-swarm interaction through embodied emotional expressivity, emphasizing simplicity, accessibility, and real-world relevance.

Thesis Committee:

Chair:	Ir. Dr. C.J.M. Verhoeven, Faculty EEMCS, TU Delft
Committee Member:	Dr. J.H. Boyle, Faculty IDE, TU Delft
Committee Member:	Dr. ing. M.C. Rozendaal, Faculty IDE, TU Delft

Preface

This thesis was written as part of the Embedded Systems and Integrated Product Design master's programs at Delft University of Technology. It reflects a personal interest in bio-inspired robotics, which was sparked while working with the Lunar Zebro team during my bachelor's.

I thoroughly enjoyed the process of rapid prototyping and appreciated the opportunity to combine academic research with hands-on experimentation. The project encouraged me to explore topics well outside my original field of expertise, including interaction design and the behavior of both humans and animals – subjects I have come to value as essential components in designing intuitive robotic systems. The process of this project highlights my favorite engineering quote: "Anyone can build a bridge that stands; it takes an engineer to build a bridge that barely stands." It takes an engineer to build a **precisely sufficient system**.

I would like to express my deepest gratitude to my two supervisors, Ir. Dr. C.J.M. Verhoeven and Dr. J.H. Boyle, for their guidance, support, and encouragement throughout this journey. Your expertise and insights have been instrumental in shaping this work. I'm especially thankful for your willingness to take on a long and complex project and for the personal input you provided throughout. I'm glad we could find a topic that genuinely interested all three of us and that you were open to taking creative risks together.

I also wish to thank Dr. Marco Rozendaal for his thoughtful participation as a member of my graduation committee. His perspective and feedback, especially on the design and interaction elements during Human-Robot Interaction lab meetings, were greatly appreciated and added meaningful depth to this project.

I am also grateful to the Lunar Zebro team and the Mirte team for providing the robot and the technical foundation on which this project was built. Their work made this project possible. Furthermore, I sincerely thank the Human-Robot Interaction lab for their constructive input during the exploration and ideation phase. Your insights and feedback added a valuable dimension to the research.

To everyone who has supported me along the way – friends, family, and fellow students – thank you for your encouragement and patience throughout this process.

Aart Rozendaal
Delft, the Netherlands
June 12, 2025

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nature has been an incredible source of inspiration in the fields of Bio-Inspired Design (BID) and Bio-Inspired Robotics [1]. Animals have had billions of years to evolve many effective and efficient sensory organs, which researchers and engineers can take inspiration from [2]. Besides the individual properties of sensory organs and organisms, the organizational structure of groups of animals is an ever-intriguing topic of research [3], [4]. These groups of animals, in some cases, can be called a swarm.

Most people can intuitively grasp swarming as we see it in nature with flocks of birds, colonies of ants, hives of bees, and schools of fish [5]. Without awareness of the system they are part of, the individuals can accomplish tasks exceeding their competencies. Studying these phenomena and their effects shows interesting results and uses in many fields [6]. When implemented in the field of robotics, this is called Swarm Robotics (SR).

Within a robotics project, communication between humans and the robot is crucial, whether through speech, typing, lights, levers, buttons, or any other way of communication. The extent to which humans can understand the robot and vice versa is responsible for its effectiveness. This is why considerable advancements have been made in this field, e.g., Google Home voice recognition and human speech, ChatGPT with almost human responses, and UIX interface design improvements that make apps and websites more intuitive. Human-Robot Interaction (HRI) is the scientific field that studies these interactions. To find the intersections of these fields, let us first discuss swarming more in-depth.

1.1 Swarming

An in-depth discussion on swarming and Human-Robot Swarm Interaction (from now on called Human-Swarm Interaction (HSI)) can be found in [7]. This report contains a literature review section on these topics. This section contains a short conclusion on the findings of that review to provide context for the reader.

1.1.1 A Definition of Swarming

A universal definition of ”a swarm” is challenging to construct. Therefore, this report utilizes the definition as in [7]: **”A Large number of animate or inanimate mobile agents in quasi-homogeneous groups with a common goal, capable of peer-to-peer communication”**. A visual representation of an example of a swarm is given in Figure 1.1.

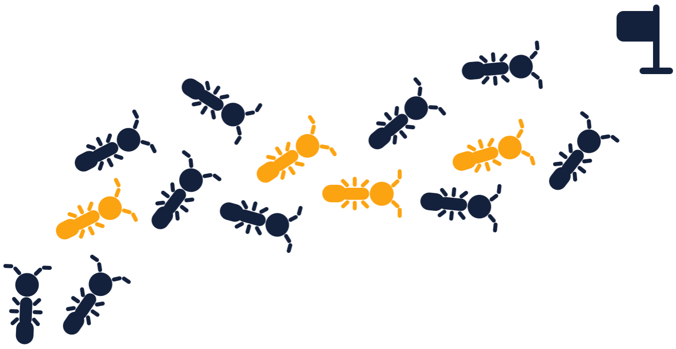


Figure 1.1: Visual representation of a swarm, showing a large number of animate agents capable of communication (ants) that are orange and blue (Quasi-homogeneous) working towards a common goal

1.1.2 Swarm Robotics

This definition changes when talking about Swarm Robotics (SR). Again, the definition from [7] is used. A robot swarm is: **"Multiple autonomous mobile robots working cooperatively on a common goal using communication"**. Let us dissect this sentence to gain a better understanding of its meaning.

- **Multiple:** Abstracting from a specific minimal number. Enough robots are needed to show the characteristics of a swarm. However, a robot can be a swarm robot without functioning in a swarm when it is designed to be able to do so.

- **Autonomous:** Each robot in the swarm should function by itself. It should prioritize its own survival and critical tasks over group activities.

- **Mobile robots:** Each robot in the swarm should be able to move. This can be done in any way that actively performs this task without external forces, e.g., wind. This excludes sensor arrays and assembly line robots from the definition of a swarm. A swarm could contain robots that cannot move independently but require help from others. However, they cannot exist solely out of non-mobile robots.

- **Working cooperatively:** The robots should work together in a symbiotic manner, i.e., helping each other achieve a common goal. Otherwise, they are merely similar robots working on the same task.

- **On a common goal:** The robots should have a common goal on which they can work cooperatively. A simple goal could be survival or interaction with the environment. Without a common goal, it would be similar robots doing different things.

- **Using communication**: The communication between the robots often happens from peer-to-peer, meaning that they use local communication and cannot guarantee global information distribution. Communication can take many forms, from speech, radio, and light to using Stigmergy - communication through the environment [8].

Although not part of the definition, robotic swarms often share some key properties. These properties are not essential for a swarm to function as one, but are responsible for interesting behavioral concepts. Later, these behavioral concepts will be called attributes. The key properties are:

Decentralised Control

Swarms use decentralized control. Some definitions also describe swarms as having "limited control" or "little human intervention." Having decentralized control with autonomous agents allows for emergent behavior to appear. In swarms, this can express itself as swarm intelligence. A phenomenon that is a research topic on its own.

Homogeneity

Most swarms are (quasi-)homogeneous. Swarms can be non-homogeneous, but this does negate some of their most useful attributes. Paired with decentralized control, homogeneity can allow for a robust and scalable system. Both are attributes that researchers are keen to learn more about. Robustness is a system's ability to remain functioning under disturbances [9]. In the context of swarms, this would be losing individuals. Scalability is the ability of a system, network, or process to handle a growing amount of work capably or its ability to be enlarged to accommodate that growth [10]. In the context of swarms, this means adding individuals to the swarm.

Common Memory

Most swarms have a common memory. Although in nature, individuals in a swarm may not know the goal and state of all other individuals in the swarm, this is easier to achieve for robots. This stems from robots' ability to communicate reliably, quickly, and over large areas while allowing for more reliable and larger memory storage than any animal. Furthermore, cloud technologies and satellites can provide common memory and measurements for all robots. This allows for an attribute not seen in nature – morphic resonance. This term was initially coined by Rupert Sheldrake (1981) in his book A New Science of Life: The Hypothesis of Morphic Resonance. It describes biological growth through a common memory shared through morphic fields. Although it was described as pseudo-science, it now finds its application in the field of SR. In short, it is an attribute that allows swarms to grow and improve through sharing observations over morphic fields, e.g., radio waves and harboring common memory.

Hierarchical Overview

This subsection is summarised in Figure 1.2. It shows a hierarchical overview of robotic swarms' definition, properties, and attributes. Lastly, in this report, a single robot is considered a swarm robot if made to adhere to the definition in 1.2. It means a single robot without a group can be called a swarm robot, even without its companion robots. Throughout this report, tests are performed on a single swarm robot.

1.1.3 Human-Swarm Interaction

Although a lot of research has already been done on SR and HRI, research on HSI is still lacking. The lack of research makes this an interesting research topic. Previous studies mainly show research using simulated robots, but there is a lack of physical prototypes to test the interactions. Examples are research for simulations [11] [12], a proposal [13], a survey [14], or a focus on control interaction without a focus on the social interactions [15]. The closest research to studying social interactions is a paper by Alonso-Mora on giving gestures to swarms to control them [16]. This touches upon the interpretation and communication between a swarm and a human. However, a focused study on the interaction between swarm robots and humans is missing and is the topic of this report.

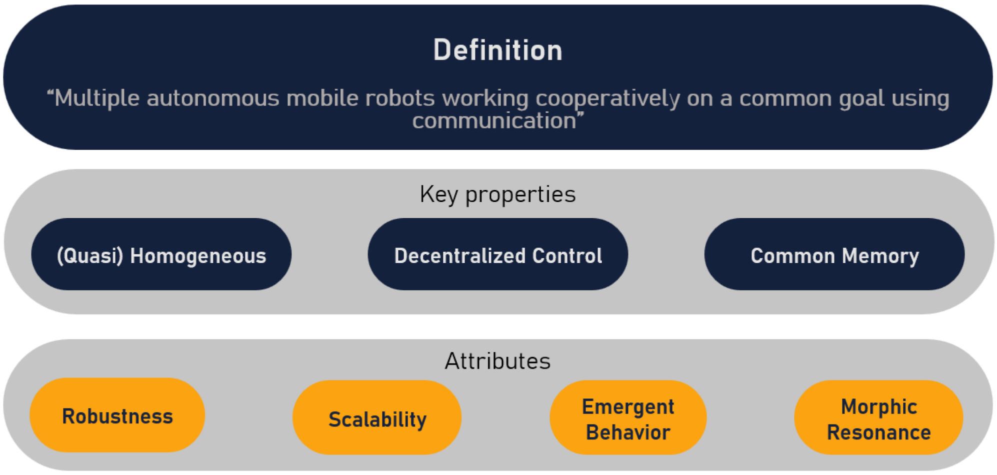


Figure 1.2: Hierarchical overview of the definition, properties, and attributes of a robotic swarm

1.2 Motivation

The research direction of this report is a combination of the research fields of BID, SRs, and HRI. The combination of these research areas, as shown in Figure 1.3, allows for an interesting and novel project to contribute to the knowledge gap for HSI. **As swarming robots could be the future, but research on Human-Swarm Interactions is limited.** Furthermore, this research area also fits the interests and expertise of the student Aart Rozendaal and his graduation supervisor, Dr. Ir. Chris Verhoeven and Dr. Jordan Boyle. Aart is studying Embedded Systems (ES) and Integrated Product Design (IPD) at DUT with an interest in combining hardware and software using BID. Dr. Boyle is part of the HRI group at DUT, which will be consulted during the project. Dr.ir. Verhoeven is Project Director at LZ with a close connection to the science center at DUT. The terrestrial robot of LZ will be used as inspiration, and the science center will function as the location in the case study for this report.

1.3 Research Questions

The gap in research proposes an interesting problem statement. Because research shows the benefits of SR but lacks in answering fundamental interactive challenges. The problem statement is formulated as: **Swarming robots will become more present in our daily lives, and there is no intuitive way to interact with them.**

This problem statement and the advantages of BID naturally lead to the following research question: **How can bio-inspired design aid in intuitive interactions between humans and swarm robots?** This research question is further broken down into sub-questions:

1. What sensors are needed to detect human presence, location, and emotions?

2. What data should be extracted from sensors to capture human behavior relevant to interaction?

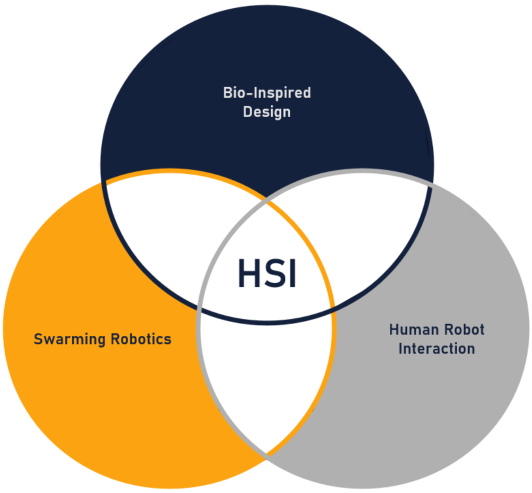


Figure 1.3: Venn diagram of the research area Human Swarm Interaction (HSI) where the three other research interests meet

3. How can the robot process the interpreted data to decide on an appropriate emotional state?

4. How can actuators be used to express a range of different emotional states intuitively?

1.4 Case Study

In 2014, the Cyber Zoo at the Science Centre on campus DUT opened [17]. It opened to allow more research on robot swarms whilst also allowing visitors to interact with the robots. Since opening, the Cyber Zoo has been looking for innovative interactive robot designs and experiments.

The LZ team has a terrestrial robot, Traici (Figure 1.4 left), which is used to test important subsystems of the space-grade version that will go to the moon. The design of Traici and the other Zebro robots resembles a beetle (Dutch: kever). It is a small, six-legged swarming robot consisting of basic forms and performing basic functions. Traici can already move, sit, stand, change stance, and do other basic locomotive tasks. This report focuses on communication, where locomotion can help convey information.

Mirte is a low-cost, fully open-source educational robot track from primary school to university level [18]. It is created by DUT. Mirte has several versions, ranging in capabilities and implementation difficulty. A visual and mechanical modification created by a different student is shown on the right in 1.4. The base of this version was used in this report as a starting point for the design. Next to the mechanical base, inspiration was taken from the open-source software approach. Creating legible, explainable, and maintainable code is crucial for the longevity of a (swarming) robot.

For the case study, the task was to **build a robotic artifact to add to the Cyber Zoo that can interact with visitors on itself or via attaching itself to Traici, Mirte, or any other (swarming) robot**.

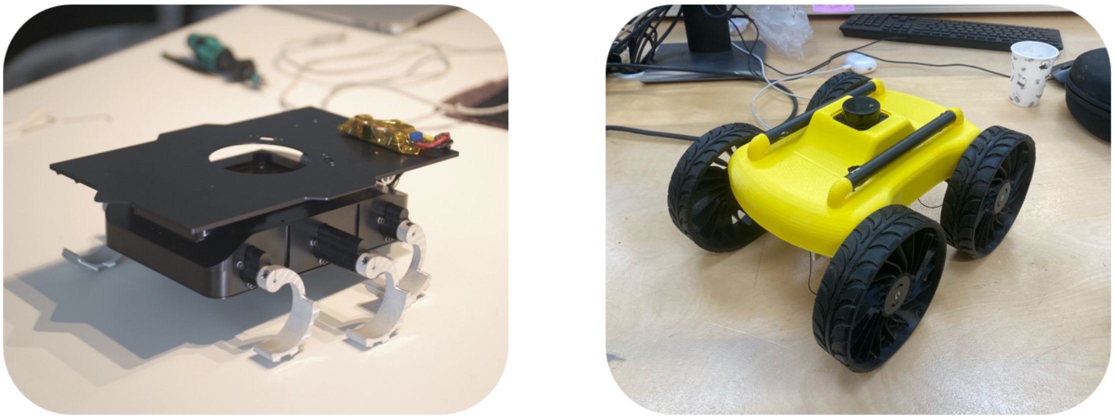


Figure 1.4: The two robots used for the case study with Lunar Zebro (LZ) (Left) and Mirte (Right)

1.5 Project Structure

Because this graduation project is the basis for two graduation reports – the other being [7]– , an overview of the full project is given in Figure 1.5. Building an intuitive and interactive robot requires research in HRI, the main topic of the IPD part shown in orange. This discusses how humans show and perceive emotions and behavior with special attention towards dogs and insects. The first is closest to humans and easier to read, and the latter is simpler and has a closer resemblance to the LZ. The design of the electroid – a silicon-based life form – in this report mimics insects in both looks and intelligence. Both visual and manufacturing aspects are part of the IPD part. The robot uses sensors and actuators to show and perceive emotions and behavior. The software and hardware capabilities are part of the junction between both reports. The question of what to perceive and how to show emotions is behavioral and is treated in the IPD report. The technical specifications are discussed in this report.

Everything mentioned above discusses the physical and behavioral design of the robot. However, the robot is still lifeless. In this ES report, life is blown into the design. Interpretation and decision-making transform the design of a robot into an electroid. **The final product of both reports is an autonomous robot (consisting of a robotic symbiote that is attached to a robotic host) that can interact with people, similar to but different from other simple existing life forms.**

1.6 Objective

In the IPD report, the design objective or vision is formulated: **”Designing a bio-inspired symbiote that aids swarming robots in sensing the environment and expressing themselves to humans in an intuitive way.”** . Here, the environment refers to people and their emotions. The vision is the basis of the full project. The technical requirements in the next section were formulated in the IPD report to achieve this objective [7].

1.7 Requirements

Here, the objective is broken down into several requirements, as mentioned in the requirements section in [7]. They are stipulated using the MoSCoW method [19]. This method

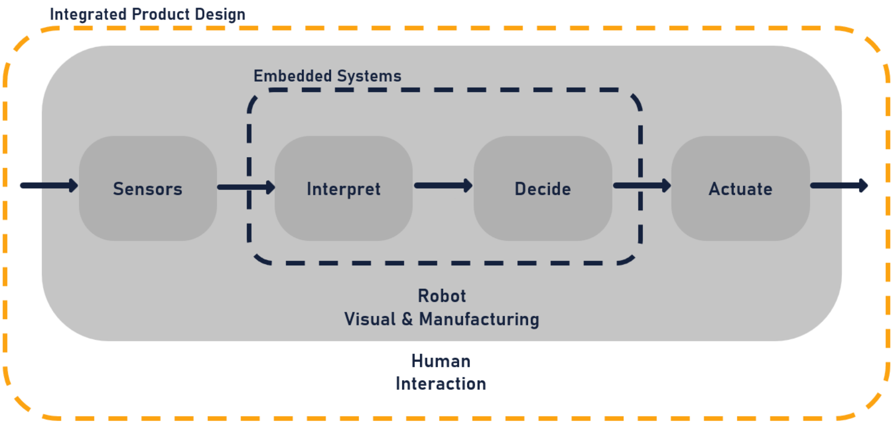


Figure 1.5: Schematic overview of the robot within the full project, with the orange dotted line showing the scope of the Integrated Product Design report and the blue dotted line the scope of this report

categorizes requirements into ”must have, ””should have, ””could have,” and ”will not have. ”

- Must have: Non-negotiable requirements that are mandatory for a successful outcome.
- Should have: Important requirements that are not vital but add significant value.
- Could have: Requirements that are nice to have, add only small value, and are pursued if time allows.
- Will not have: Requirements that are out of the project's scope due to prioritization or time constraints.

The requirements of the robot are sensitive for the case study, and the requirements that mention the symbiote are important for the whole project. The requirements for this project are defined below:

Must Have Requirements (M)

- **M.01:** The robot must successfully finish the device safety check of TU Delft
- **M.02:** The symbiote must show predictable behavior
- **M.03:** The symbiote must be able to detect a human from at least 3 meters
- **M.04:** The symbiote must be able to detect at least 3 bodies
- **M.05:** The symbiote must be able to detect a face from 1 meter
- **M.06:** The symbiote must be able to detect at least 3 faces
- **M.07:** The symbiote must be able to discriminate the closest person
- **M.08:** The symbiote must be fully autonomous except for an external power source
- **M.09:** The robot must be rechargeable
- **M.10:** The robot must be able to run for at least 3 hours
- **M.11:** The robot must be able to move around on a flat surface

- **M.12:** The symbiote must be able to function on startup without I/O devices, e.g., screens, keyboards, or mice
- **M.13:** The symbiote must be able to operate at least at 10Hz
- **M.14:** The symbiote must have a camera angle of at least 60 degrees vertically and horizontally
- **M.15:** The symbiote should show distinguishable emotions
- **M.16:** The symbiote should show detectable emotions

Should Have Requirements (S)

- **S.01:** The symbiote should be able to detect loud noises >70 dB
- **S.02:** The symbiote should not be triggered by normal conversation, <65 dB
- **S.03:** The symbiote should perform error handling on false positives and false negatives detections
- **S.04:** The robot should be rechargeable using a universal charger
- **S.05:** The symbiote should allow for modifications
- **S.06:** The symbiote should allow options for additional sensors
- **S.07:** The symbiote should use standard communication protocols
- **S.08:** The symbiote should use basic tools to construct
- **S.09:** The symbiote should use materials with nationwide availability
- **S.10:** The symbiote should detect an emotion within 200 ms.
- **S.11:** The symbiote should detect the correct emotion with an accuracy of 90%
- **S.12:** The symbiote should move naturally with no perceived jumps in speed
- **S.13:** The symbiote should be less than 500 euros when bought in bulk

Could Have Requirements (C)

- **C.01:** The symbiote could be able to detect the direction of sounds
- **C.02:** The symbiote could be less than 500 euros in cost
- **C.03:** The symbiote could be ROS 2 compatible

1.8 Report Structure

The report structure follows the information flow through the robot as schematically shown in Figure 1.6. The sensors – camera and microphones – gain data from the surroundings. Chapter 2 discusses the selection and technical details. These inputs are then interpreted in Chapter 3. In this stage, the raw camera data is converted into useful insights. This is done through face, body, and emotion detection. Similarly, using the data from the microphones, the loudness and direction of sounds can be detected. After the data is interpreted, it enters the robot's decision-making machine – the brain – as discussed in Chapter 4. The

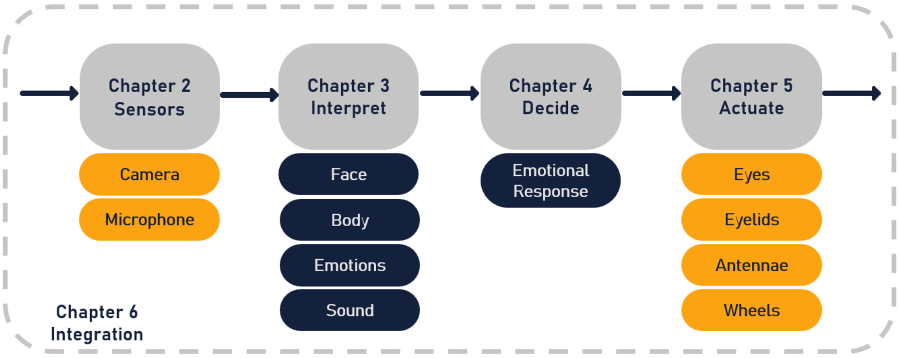


Figure 1.6: Schematic overview of the report where every chapter treats one of the four subsystems of the robot following the flow of information through the system, with chapter 6 discussing the integration of these subsystems

decision-making mechanism aims to stay simple to more closely mimic insects rather than more intelligent animals. A simpler system that allows for a more thorough understanding will create a foundation from which more intelligent robots can be created. Instead of using AI to simulate intelligence, building simpler systems and building on top of them allows for a more fundamental understanding that stimulates research [20]. After deciding on the emotional response, the robot aims to perform the emotion by controlling several motors. The four expressive bio-inspired appendices are the eyes, eyelids, antennae, and wheels that mimic natural movement and facial expressions. The expression and interaction are discussed more in-depth in the interaction design section in [7]. The actuation and software are discussed in Chapter 5.

After discussing all the subsystems in the robot separately, the integration occurs in Chapter 6. This chapter discusses integration challenges, technical insights, system analyses, and system overviews. It should provide an overview of the whole design and performance of the robot. After integration, Chapter 7 discusses the evaluation of the design concerning the requirements and answers the research questions. Combining everything and coming to a discussion, conclusion, and recommendation for future work.

Chapter 2

Sensing

This chapter discusses the sensors used and gives a general overview of the subsystem. Sensors that mimic vision and hearing were selected. Humans read and express 55% of their emotions through facial expressions and 38% through voice intonation [21]. **This means vision and hearing account for roughly 93% of our emotional sensing.** The exploration section in [7] discusses the reasoning concerning behavioral and interactional design.

2.1 Sensors

Choosing senses to mimic is not the same as selecting the right sensors. Different sensors like cameras, Lidar, and radar can simulate vision, but all provide advantages and disadvantages. For example, Lidar and radar provide distance information, but cameras can give more detailed information in a 2D matrix.

2.1.1 Sensor Selection

Options are a camera, an infrared camera, Lidar, or even ultrasonic sensors to simulate vision. However, to satisfy Req M.03, the robot needs to be able to detect humans. An infrared and a color camera could be chosen depending on the interpretation method. Because of its wider availability and the possibility of interpretation, **a regular camera has been chosen.** This also allows for emotion recognition in the interpretation phase.

Furthermore, only one camera is used for the symbiote. This choice is not arbitrary. Stereo cameras can also provide detailed distance information, unlike a single camera. The reason for not using stereo is that other measurable proxies can give an indication of the closest person, which suffices for this project. The section on facial and body feature detection in the interpretation chapter discusses how the relative size of the detected people is used to provide this data. Besides this, the decision chapter discusses how the face and body size on the screen can be used to make a gross estimation of the closeness of a person. This than can than be used in comparison with the previous fram to determine movement direction.

For hearing, fewer options are available. Only simple data is required, i.e., loudness and direction. To achieve this, **two simple microphones were chosen**. More expensive microphones could allow for more complex interpretation methods, such as speech recognition or tone of voice detection.

2.1.2 Camera

Because of its worldwide availability, **the RP Camera 3 wide is used [22]**. Figure 2.1 shows the small camera that is part of the RP camera series. The camera and all other components

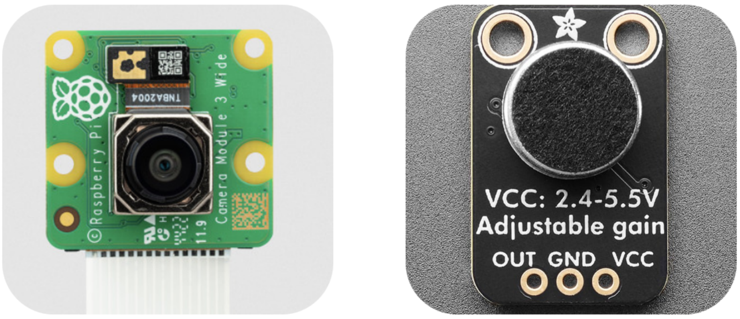


Figure 2.1: The RP Camera 3 wide (left) and the Adafruit MAX4466 microphone (right) used in the project

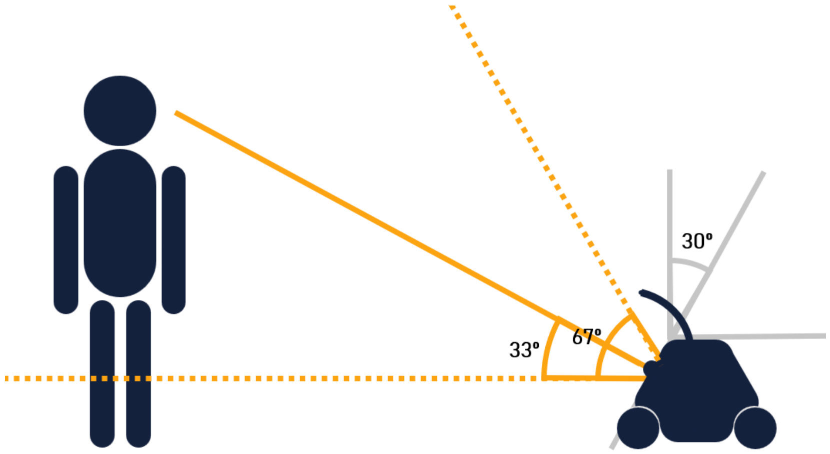


Figure 2.2: Side view of the viewing angle of the camera (in orange) on the robot (in blue) when mounted on a 30° slope

can be found in the Bill of Materials (BoM) in Appendix A. It uses the IMX708 12-megapixel camera sensor with an autofocus lens. The wide-angle lens has a 67-degree vertical and a 102-degree horizontal camera angle. It is mounted on a 30-degree slope at 30cm height and 5cm inward, as shown in Figure 2.2. This makes full use of the wide angle and means that the robot can see the head of an average Dutchman (180cm) standing up straight at 45cm. As obtained by filling in the values in Equation 2.1, with $h_{avgman} = 180cm$, $h_{sensor} = 30cm$, $d_{sensor} = 5cm$ and $d_{human} = 45cm$.

$$\tan(60^{\circ}) = \frac{h_{avgman} - h_{sensor}}{d_{human} - d_{sensor}} \tag{2.1}$$

The camera can both output video and pictures on request. Both use a 4608 x 2592 pixel format that requires some formatting for the interpretation.

2.1.3 Microphones

The two used Adafruit microphones are shown in Figure 2.1 on the right. These are $20Hz - 20kHz$ microphones that capture the full range of the human voice. **They utilize the MAX4466 amplifier with an adjustable gain of 25x-125x.** The microphone is suitable for sampling audio and projects that require Fast Fourier Transforms. This allows for an interesting follow-up

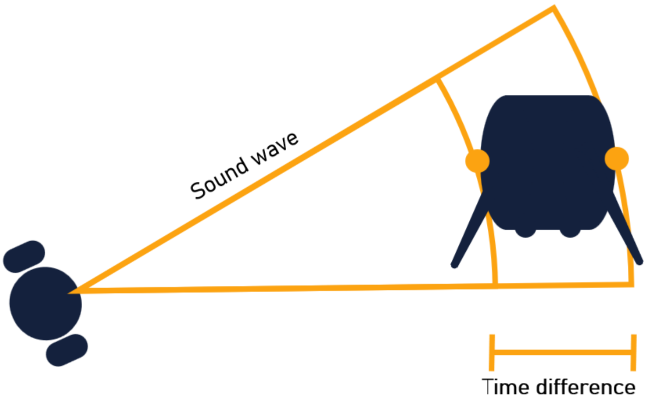


Figure 2.3: Difference in time of arrival for both microphones for a soundwave from a source on the left to the robot on the right

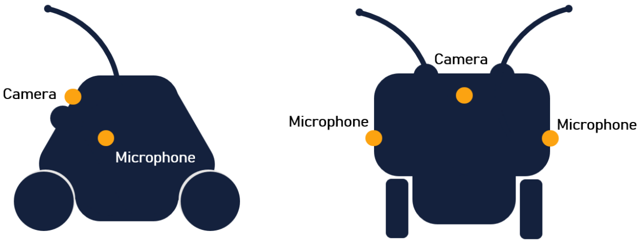


Figure 2.4: Location of the camera and microphones on the robot from a side view (left)and front view (right)

topic in further projects. **This project uses the 125x gain setting with a rail-to-rail V_{pp} output to detect loud sounds.** It has a $V_{cc}/2$ DC bias and is best connected to the 3.3v supply of the Arduino, which has less noise than the 5v supply.

Another reason for choosing the higher gain setting is that it increases the likelihood that the sound is detected because the chosen Arduino has no analog interrupts (issues and other solutions discussed in Chapter 6).

Although not used in this report, the robot can handle stereo sound. This means there are two microphones, one on either side of the robot, as shown in Figure 2.3. Using the difference in arrival time for the sound wave, an estimation can be made on where the source is – on the left or the right.

2.2 Sensor Location

The robot uses three sensors – two microphones and one camera. More sensors could be added, allowing for more interesting interactions. The location of these sensors is shown in Figure 2.4. The two stereo microphones are placed as far apart as possible to provide the best performance in locating sound sources. The camera is mounted between and slightly above the eyes. It is secured steadily in the frame to minimize image quality degradation through shaking or movement.

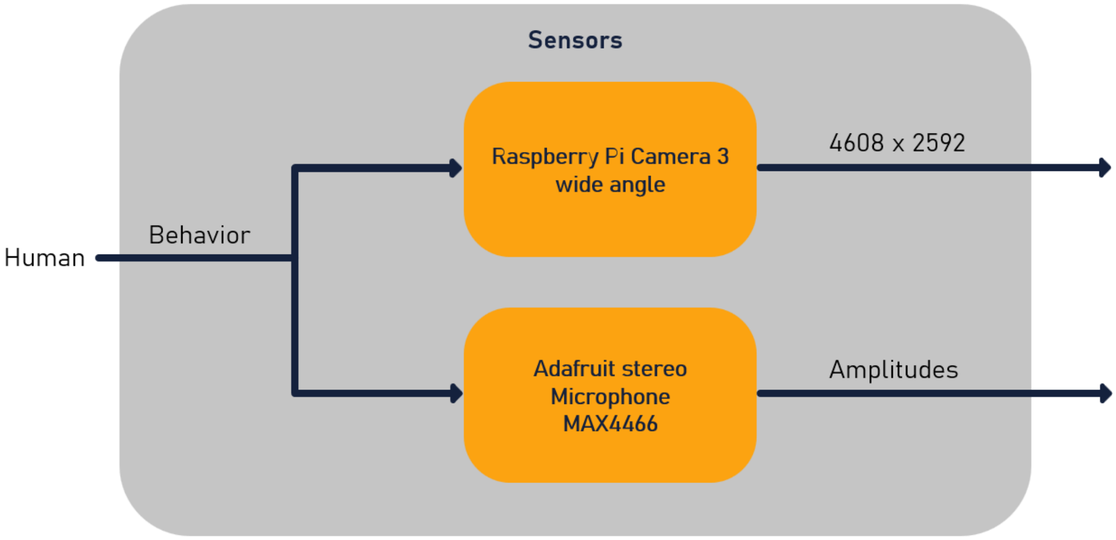

Sensors
Human
Behavior
Raspberry Pi Camera 3 wide angle
4608 x 2592
Adafruit stereo Microphone MAX4466
Amplitudes

Figure 2.5: Schematic overview of the sensors of the robot showing the camera and microphones and their respective output

2.3 Sensor Overview

To conclude this chapter, an overview is given in Figure 2.5. It shows how the camera and microphones transform human behavior into data. **The raw sensor data enters the next step in the robot's data flow: interpretation.**

Chapter 3

Interpretation

Having selected the sensors, the task is to process the raw data and create useful information. This chapter discusses this task for both the camera and the microphone and what devices are used to do this. Furthermore, it treats the communication between these devices. **After the interpretation stage, all raw data from sensors is converted into useful information so that a decision can be made.**

3.1 Features

A colored camera and two microphones allow for extracting many useful features. The features required for intuitive interaction determine how the data is processed. This section discusses the selected features that are shown in Figure 3.1 as mentioned in [7]. Let us go over these features.

3.1.1 Camera Features

The camera handles three categories of features. These are the face, body, and emotion detection. Each feature contains information that will be used during the decision stage.

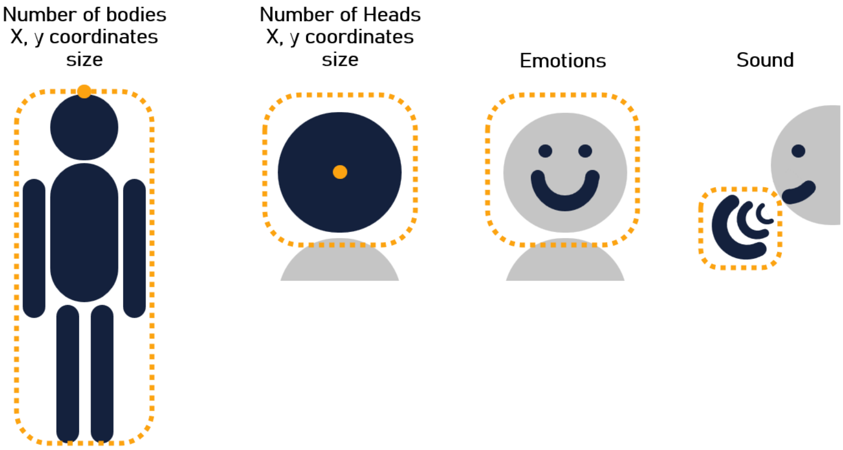


Figure 3.1: Detected features (in dark blue) that are extracted from the camera and microphone, with the detection area (in dotted orange) and the location of the x and y coordinates (orange dot)

Facial Features

The first thing that is looked for in the image is faces. All detected faces are counted and given a bounding box. A bounding box is a box or rectangle that encompasses the face. Information about these boxes is used to calculate and retrieve the required information.

All bounding boxes are saved in a list. The length of this list is the number of detected faces or heads. There are three distinct cases.

1. **The first case** is that the list length is zero, and no faces are in the image. The number of faces, the x and y coordinates, and the size are set to zero.

2. **The second case** is that there is one face. In this case, the size or area of the face is $A_{face} = w_{bb} * h_{bb}$, with w_{bb} the width of the bounding box and h_{bb} the height. The coordinates of the bounding box are given as the top left corner, with the x value increasing to the right and the y value increasing downwards. Calculating the x,y coordinates of the centre is thus $x_{face} = x_{bb} + wbb/2$ and $y_{face} = y_{bb} + hbb/2$.

3. **The third case** is when there are two or more faces. The face size is calculated first, and the coordinates are updated only if the face size is larger than the previous face in the loop. This ensures the largest – an approximation for the closest – face coordinates are stored. The code used can be found in Appendix B.1. It is important to mention that this does not show how close a person is, but rather who the closest person is.

Body Features

Similar calculations can be done for the bodies. The three cases are also similar. The only difference is that the top of the bounding box is stored for the body. This information is later used to look at the closest person and for the height. The coordinates are calculated as $x_{body} = x_{bb} + wbb/2$ and $y_{body} = y_{bb}$. The code is given in Appendix B.3.

Emotional Features

Reading emotions is done using the face. Using the face's bounding box, the image is cut into a smaller image of just the face. Some padding is added if the bounding box cuts off important facial features. This image is analyzed to extract an emotion. The emotions are the 7 basic emotions as used in FER2013 data set – 0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral – [23]. These are the basic emotions as given in [24]. This research formed the basis for the FER2013 dataset; in this report, 7=none is used when no face is present or no emotion is dominant. The code for this can be found in Appendix B.2.

3.1.2 Microphone Features

Using a stereophonic microphone setup allows the extraction of two pieces of information. First, the loudness or amplitude of the sound. Using sound level thresholds can enable a response to loud noises. This could be done dynamically, where the threshold adjusts to a different level depending on the loudness of the environment. Secondly, the direction of the sound. This can be done by using the time difference in the arrival of the sounds at the different microphones, as explained in 2.3.

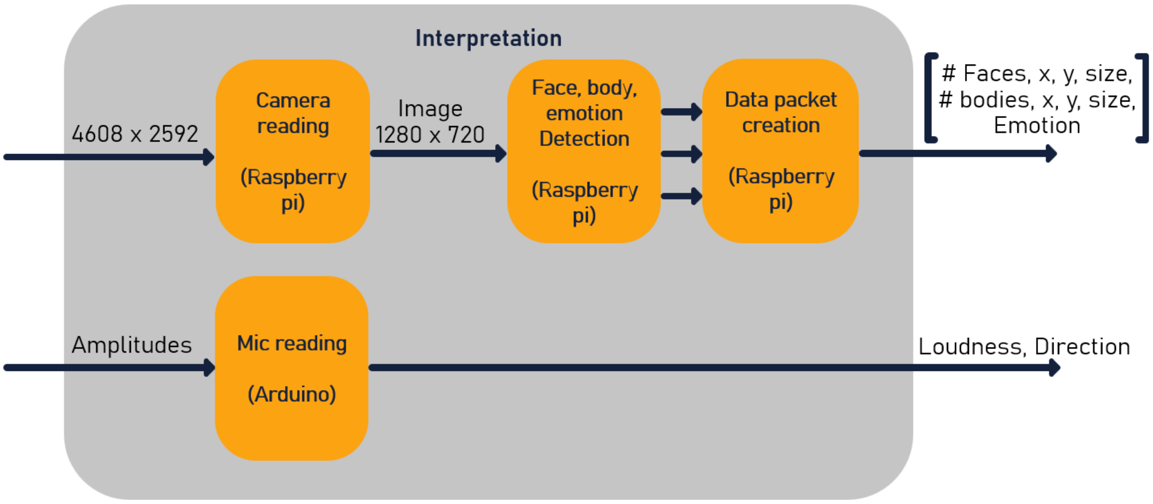


Figure 3.2: Schematic of the interpretation stage of the robot showing the inputs (raw data), the subsystems (in orange) and the outputs (features)

3.2 Overview

Now that the output of the interpretation stage is discussed and the input is known, an overview is given. The overview in 3.2. The subsystems in orange are treated more in-depth in the following sections, with face, body, and emotion recognition treated separately.

3.3 Camera Reading

The RP Camera 3 Wide allows for several options in reading the data [22].

3.3.1 Video or Image

Both options of reading the camera data – continuous video or images – have advantages and disadvantages. The continuous video can obtain 1280x720 resolution at 30FPS but struggles at higher resolutions. Images can use the full quality of the sensors, but this requires processing time. **Taking images allows for more flexibility in resolution using the RPs built-in camera processing.**

3.3.2 Resolution, Accuracy, and Speed

In real-time applications, the resolution needs to be carefully selected as it determines both accuracy and speed. Using lower resolutions scales down the 4608x2592 image as long as the 16:9 aspect ratio is adhered to. Using a lower resolution increases the processing speed and thus the FPS but decreases the accuracy, decreasing the distance at which it detects people. Table 3.3 shows the common resolutions 16:9 with the average processing time [25]. The code that was used for this test is shown in Appendix B.

3.4 Microphone Reading

The two microphones are connected to the Arduino's analog pins, A0 and A1. Because the Arduino does not have analog interrupts, polling is used to read them. This means it is subject to other functions running on the Arduino, i.e., motor control and state handling

Resolution	Name	FPS	min FPS
3840×2160	4K UHD	14.36	8.40
2560×1440	QHD / 1440p	56.05	34.60
1920×1080	Full HD / 1080p	70.20	56.92
1280×720	HD / 720p	110.34	61.72
960×540	qHD	119.43	67.76
854×480	FWVGA / 480p	120.12	71.36
640×360	nHD	122.70	72.99

Table 3.3: Common 16:9 resolutions with the corresponding processing time in FPS on a RP 4GB RAM with the Pi camera 3 wide

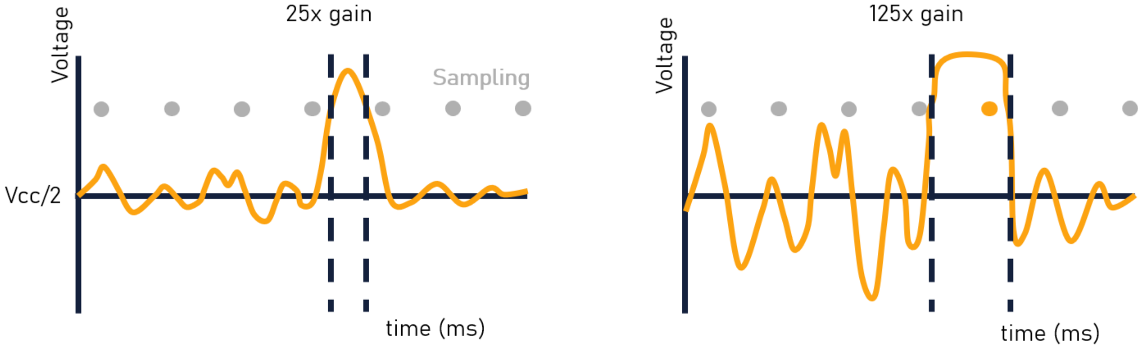


Figure 3.4: Effect of the audio gain settings on the noise detection and clipping

(both discussed later). A higher gain can help in loud noise detection when using a polling method for the sensor reading. It does not retain sound quality, but this is not an issue for this application. Figure 3.4 shows how a higher gain helps detect loud noises when polling. The analog inputs on the Arduino utilize a 10-bit ADC, meaning 0v-5v is converted to a 0-1023-bit integer. Because the microphone is connected to the 3.3v input, it uses a range of 0 to $3.3v/5v * 1024 - 1 = 674$ with $V_{cc}/2 = 674/2 = 337$. Measurements show that $V_{cc}/2 = 330$. This means the sound thresholds should be $threshold_{high} = 330 + x$ and $threshold_{low} = 330 - x$ with x being any arbitrary number smaller than 330.

3.5 Face Detection

Face detection lays the foundation for the robot's interactive capabilities. Without this, the robot does not know where people are, where their face is, and it cannot determine the emotion. In this section, different face detection methods are tested and compared. The tests and comparisons are all performed on a RP 5 4GB RAM with an image resolution of 1280x720.

3.5.1 Terminology

Before diving into the different methods, it is important to clarify some subtle differences in terminology. Three terms – detection, recognition, and verification – are often used interchangeably, but for clarity, they are defined below:

- **Face Detection:** detects the presence of a face in an image or video feed. It can detect multiple people in a single frame but cannot directly identify or discriminate between people.

- **Face Verification:** detects the presence of a face in an image or video feed. It can verify if the detected person is the person it is looking for. This method functions using one-to-one comparisons.
- **Face Recognition:** detects the presence of a face in an image or video feed. It can identify (recognize) people in a frame. This method compares people to a database. This method uses one-to-many comparisons.

Choosing between these three is not arbitrary, as all methods have advantages and disadvantages and different legal implications. **Detection is superior in simplicity and speed, and removes any privacy concerns.** It requires no database or photo storage. Because discrimination is not required, also not for emotions, detection is the straightforward option and is chosen for the symbiote.

3.5.2 Methods

There are many different methods and implementations possible for face detection. Methods can differ in fundamental mathematical approaches, software implementations, or hardware configurations. In this section, several methods are discussed, along with their implementation.

Deep Neural Network (Opencv)

Deep Neural Networks can be used for face detection and recognition. These networks can be optimized for face detection. One such model is the Opencv DNN Face detection model [26]. Deep Neural Networks are neural networks with many layers and, depending on their structure, are specialized for different use cases. The benefit of these models is that they can handle different lighting conditions and face orientations if properly trained. The code for implementing the well-trained Opencv model is given in Appendix B.1.

Convolutional Neural Network (dlib)

Convolutional Neural Networks are a type of DNN. They use a convolutional kernel. This allows the model to use adjacency of pixels. This is extra useful for image recognition applications. A popular CNN model is given in [27]. The implementation is shown in Appendix B.1.

BlazeFace (Mediapipe)

Google created a highly efficient lightweight model for real-time embedded applications called BlazeFace [28]. It has a pre-trained model that uses its media pipe framework, which is well-suited for an RP. The downside is that it is trained on larger heads that occupy 20% of the screen or more, making it less effective for faces further away. The implementation is given in Appendix B.1.

Haar Cascades (Opencv)

A lesser-known but widely used method uses the Viola-Jones algorithm [29]. This method is used in digital cameras and other applications with low processing power. It uses a cascade of many small calculations to check an image for faces. Training a model for these calculations can be intensive, but by using mathematical tricks like converting images into integral images, the execution is high-speed. Opencv has many trained models, one for face detection [30]. The model can struggle with non-frontal faces. The implementation can be found in Appendix B.1.

Histogram of Oriented Gradients (dlib)

Using Histogram of Oriented Gradients is not often used but is still helpful for real-time embedded applications [31]. Although usually outperformed by CNN, some models show potential, especially for hardware-programmable applications. Dlib has a Histogram of Oriented Gradients using a Support Vector Machine classifier [32]. The implementation is shown in Appendix B.1.

Facial Landmark Detection (dlib)

Lastly, the facial landmark detector. This method uses key points on the face to detect it and its orientation. It stores these points in a vector. Using landmarks can yield higher accuracy and even help when faces are partly covered, but it is often slower. Dlib has a pre-trained model [33]. The implementation can be found in Appendix B.1.

3.5.3 Comparison

The implementations mentioned above were used to compare these methods. Each method ran a thousand times. The built-in performance measurement timer of the RP was used. The results are shown in Table 3.5. The variance is calculated assuming a normal distribution. The mean of HOG, facial landmarks, and CNN methods exceeds the Req M.13. Although the average conforms to the requirements, the maximum time does not $(1/0.1296 < 10FPS)$. This leaves the Haar Cascade and BlazeFace Method.

Because both methods have similar performance in terms of speed, they will be assessed on range. The BlazeFace model is optimized for selfie images where the face takes up more than 20% of the screen. Because of the wide-angle camera, this is only the case up to 40 cm away from the camera. Testing showed no faces were detected after roughly 70 cm. Testing using the Haar Cascade model showed that it efficiently works up to at least 100 cm. **This means that the Haar Cascade model is selected.**

Method	Mean Time (s)	Variance	Max Time (s)	Min Time (s)	Avg FPS
Haar Cascades	0.0114	0.000003	0.0243	0.0091	88.09
HOG	0.3056	0.000143	0.4363	0.2857	3.27
Blaze Face	0.0127	0.000002	0.0228	0.0106	78.53
DNN	0.0580	0.000029	0.1296	0.0516	17.23
Landmarks	0.1378	0.000003	0.1547	0.1340	7.26
CNN	11.4305	0.031514	11.8602	11.2960	0.09

Table 3.5: Performance comparison of six face detection methods run one thousand times on the RP 5 4GB RAM, with the variance calculated assuming normal distribution

3.6 Emotion Detection

Emotion detection is more complex as it has to build on face detection. **If the face detection model does not work well, then the emotion detection model will also not work well.** To properly assess emotion detection, a categorization of emotions needs to be constructed.

3.6.1 Emotions

Creating a set of basic human emotions is already done in [23]. The Facial Expression Recognition (FER) dataset created in 2013 forms the basis for many emotion recognition methods.

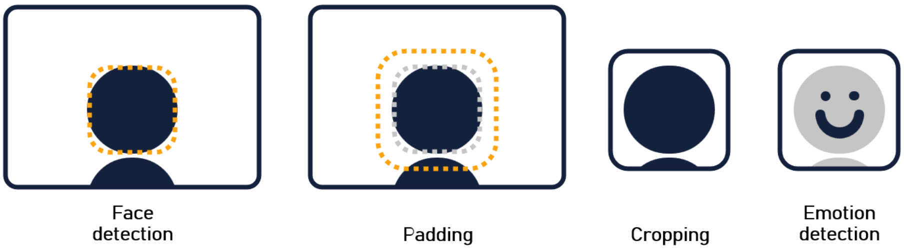


Figure 3.6: Processing steps for emotion detection from left to right

Although the FER2013 set is used, it is certainly not perfect. The dataset is unbalanced. It has the largest number of ”happy” images (1774) and the smallest number of ”disgusted” images (111), which can heavily skew the outcomes. Furthermore, it performs poorly on non-frontal faces, angled faces, faces with glasses, or partially obstructed faces. **Real-time application models often struggle and perform at 66% accuracy** [34]. This is not enough to satisfy the requirements using the current hardware and model, but better hardware could allow for larger models that could outperform these.

To perform emotion detection, the code first runs the Haar Cascades face detection model. The largest bounding box is selected, i.e., the closest person. The bounding box is often too small and cuts off too much of the head. Thus, padding is added on each side. The padding influences the computation speed as it enlarges the image that is processed. The lowest value that gave at least a 75% success rate was selected. The success rate is defined as $rate_{succes} = \frac{\sum DetectedEmotions}{\sum DetectedFaces}$. This means the emotion model success rate is only assessed on images where a face is detected. The padded box is cropped from the image and sent into the emotion detection model. The processing steps are shown in Figure 3.6.

3.6.2 Methods

The three different methods all use the FER2013 data set, and a more accurate description would be other models of a similar method. These methods are compared on performance and speed metrics.

DeepFace

The DeepFace model can perform at 86% accuracy on their test set [35]. The DeepFace Python library uses a lightweight model made for real-time applications. This is the most popular emotion detection library for the RP and can perform well on CPUS. The implementation is given in Appendix B.2.

Facial Emotion Recognition (FER)

FER is a Python library that is inconveniently called different but uses the same acronym as the dataset it uses. The library has an even more lightweight model than DeepFace that can perform faster but lacks performance. Often with lower accuracy on the test sets [34]. The implementation is given in Appendix B.2.

FER with a Multi-Cascading CNN

Using the same library but enabling the MTCNN option lets the model perform extra processing and an extra CNN step. This makes the model more robust and gives better performance, but it will take longer. Using the CNN can also better handle covered and rotated faces. The implementation is provided in Appendix B.2.

3.6.3 Comparison

The comparison of these methods is shown in Table 3.7. Again, the variance is calculated assuming a normal distribution. It is interesting to mention that the DeepFace model always performs at a 100% success rate as it always assigns a dominant emotion. Looking at the combined time of the face and emotion detection, the improved MTCNN model can be disregarded as it is too slow for this project, as stated in Req M.13. Both other models are extremely close to the requirement. Because body recognition is needed to reach Req M.03, whereas Req S.10 and S.11 are of the "Should have" category, body recognition will get the preference. This is why the decision is moved to the end of this chapter.

One last remark is that the success rate of the DeepFace model is significantly larger than that of the FER model. Both maximum times surpass the requirement thresholds, but using the variance and mean, it can be calculated with what probability this happens. Equation 3.1 calculates the probability the processing time is longer than Req M.13 of 100ms. Filling in the mean (μ) and variance (σ) of both shows that **the probability the time is larger than 0.1s is 46.0% for DeepFace and 40.03% for FER.**

$$P(T > 0.1) = 1 - \Phi(Z_{0.1}), withZ_{0.1} = \frac{0.1 - \mu}{\sigma} \tag{3.1}$$

Both the emotion recognition models, FER and Deepface, use the FER2013 data set. This means they both will most likely not outperform other real-time models. However, no tests have been performed. Both models could prove useful when better hardware and data are available. The final decision is shown in the end of this chapter.

Test Name	Detection	Mean (s)	Variance	Min (s)	Max (s)	Success (%)
Deepface	Emotion	0.0859	0.000441	0.0567	0.4590	100.00
	Emotion + Face	0.0979	0.000436	0.0678	0.4754	
FER	Emotion	0.0763	0.001570	0.0071	0.2246	74.8
	Emotion + Face	0.0904	0.001540	0.0205	0.2983	
MTCNN	Emotion	0.1915	0.000680	0.0000	0.5462	97.8
	Emotion + Face	0.2044	0.000754	0.0119	0.6518	

Table 3.7: Performance breakdown of emotion detection methods alone and with face detection

3.7 Body Detection

The last of the three detection methods is body detection. **The reason for using body detection is the limitations of range on face detection.** Without it, Req M.03 could not be met. Before diving into different methods, it is important to make a distinction between pose and body detection.

3.7.1 Pose vs Body Detection

Both pose and body detection are ways of detecting humans at an extended range. For all intents and purposes of this report, body detection is the detection of a body on an image or video. Body detection is capable of detecting the size (bounding box) and the coordinates of a body in the image or video. Pose detection is similar but utilizes key features like shoulders, hands, and head to give information about the pose of the humans. This can include orientation, the relative size of body parts, and even gestures. Although interesting for further research, **this report will disregard pose detection due to time constraints and complexity.**

3.7.2 Methods

The different methods for body recognition are shown below. For some of the methods, multiple models were tested as each model provided different advantages and disadvantages.

You Only Look Once (YOLO)

The You Only Look Once model is one of the most widely used body detection CNNs. YOLO is made for real-time embedded applications but has different models for different purposes. Yolov5 was long the most used model, but the current Yolov8 model is said to outperform it in almost every metric [36]. Furthermore, YoloFastestv1.1 is a stripped-down version optimized for speed [37]. The implementations of these three models are in Appendix B.3.

Haar Cascades (OpenCV)

Similarly, as for face detection, Haar cascades can be utilized for body detection. However, with faces, the frontal face model was chosen, but here, a whole body and upper body model needs to be selected [30]. Implementations for both can be found in Appendix B.3.

Histogram of Oriented Gradients (Dlib)

The HOG method is very similar to that of face recognition. Again, using Dlib, the implementation can be found in Appendix B.3.

Motion (CV2)

The cv2 library that handles the images and image conversions can also perform pixel subtractions. This can be utilized to detect changes in the image compared to the environment and background. Setting the correct threshold values for body detection is crucial. The implementation of this method is shown in Appendix B.3.

3.7.3 Comparison

The results for all the implementations are shown in Table 3.8. Firstly, let us use Req M.13 to eliminate several options. Yolov5, Yolov8, and the HOG method all take longer than 100ms, even with their fastest cycle. Adding the Haar cascade upper body method mean duration and the chosen face detection gives $0.0897 + 0.0114 = 0.1001$. With the knowledge that other peripherals are running and the processor will heat up, hampering performance, this option is also disregarded.

The motion subtraction method performs in terms of processing time. However, the method struggles with mobile applications and is better suited for stationary ones, e.g., security cameras. This is due to the difficulty of background subtraction when the background is changing because of movement. This means there are two methods left – YoloFastest and Fullbody Haar cascade.

When testing both models, the Yolo model proved significantly more reliable. The Haar cascade model would give 12 false positives in 5 minutes of runtime. In the same environment, the Yolo model had no false positives. Although tuning and retraining can try to get rid of these false positives, this is a well-known issue [38]. It works better for frontal faces because Haar-like features perform worse on objects with different angles and different poses, i.e., bodies. The false positives are further explained through the method itself, as complex environments have a higher likelihood of having shapes that trigger these features. **This is the reason for choosing the YoloFastestv1.1 method.**

Method	Mean (s)	Variance	Min (s)	Max (s)	FPS
Yolov5	0.4776	0.004931	0.4389	2.5231	2.09
Yolov8	0.5109	0.005330	0.4708	2.5914	1.96
YoloFastestv1.1	0.0650	0.000015	0.0586	0.1009	15.39
Haar Cascade Upper	0.0897	0.000061	0.0714	0.1414	11.15
Haar Cascade Full	0.0279	0.000019	0.0201	0.0608	35.80
HOG	0.2869	0.000073	0.2673	0.3767	3.49
Motion	0.0261	0.000006	0.0219	0.0687	38.32

Table 3.8: Performance breakdown of Body detection methods

3.8 Data Packet Creation

The serial port is used to transfer all the data from the RP to the Arduino, which will use this data in the decision stage. The serial port of both devices is an easy interfacing method with a baud rate of 115200. This means that it can send 14400 bytes per second.

The data packets themselves are created by creating a Comma Separated Values (CSV) string. CSV is a standard data format that is widely used. The data is always sent in the same order, and interpretation of the data is performed on the receiving side. Using this method has two advantages. Firstly, adding, removing, or altering data or data types is easy. Secondly, changing the receiver to a different device with a lower-bit processor or different data types is also easy. This makes it a more universal method that allows for alterations satisfying Req S.05.

The data packet is a string in the following format $(n_{Faces}\ , x_{largestface}\ , y_{largestface}\ , size_{largestface}\ , n_{Bodies}\ , x_{largestBody}, y_{largestBody}, size_{largestBody}, Emotion)\ \backslash n$. All these values are as described in section 3.1.

Because the robot uses this convention, it is not optimized for speed. Every symbol in the string is interpreted as a single byte. This means that 1-9 are 1 byte and 10-99 are 2 bytes. The duration of a message is thus dynamic. However, an upper bound on the time can be calculated. For the number of faces and bodies, a maximum of 1 byte can be used, as no more than nine people will be detected. The X coordinates take at most 4 bytes (1280), and the y coordinates 3 bytes (720). The largest sizes are 6 bytes $(1280 * 720 = 921600)$. Lastly, the emotions are 1 byte, and the commas with $\backslash n$ total 9 bytes. Adding everything up gives $2*1+2*4+2*3+2*6+1+9 = 38$ bytes. **This number is improbable, but a 38-byte upper bound would result in a time of** $38/14400 = 2.5ms$**.** The lower bound results in $9 + 9 = 18$ bytes and gives $17/14400 = 1.2ms$.

Using fixed lengths for every data type would mean 1 byte for the number of faces and bodies, 2 for the x and y coordinates, and 4 for the sizes (as they require 2^{32} bits or a ”long” type) and 1 byte for the emotions. No bytes are required for separators as the length is known. This would result in $2*1+2*2*2+2*4+1 = 19$ bytes. If, for simplicity, all data types were the same length (all longs), it would take $9*4 = 36$ bytes. This is similar to the improbable worst-case scenario for the dynamic CSV version.

The dynamic CSV implementation was chosen because it does not interfere with the requirement on time, and it helps standardization and allows customization. The implementation is shown in Appendix B. Lastly, it is essential to mention that the implemented method does not utilize any error-checking algorithm, e.g., CRC, checksum, but uses the $\backslash n$ as an alignment byte.

3.9 Parameter Selection

Finishing the section with parameter selection. Some parameters have been set but have not yet been discussed, or they were dependent on other systems. Here, the selections are explained and summarized.

3.9.1 Detection

The face detection and body detection methods are crucial for Reqs M.04, M.05, and M.06. Adding the mean durations of the YoloFastest, Haar cascades face model, and the worst-case data packet gives a total processing time of $0.0650 + 0.0114 + 0.0025 = 0.0789s$. That is well below the 100ms in Req M.13. However, adding any of the emotion detection models doubles this value and far exceeds the requirement. **For this report, emotion detection is not further pursued.** However, a faster processor or optimizations could allow this to fulfill the requirements. This could add interesting insights to further research.

3.9.2 Camera

For the camera itself, the RP Camera 3 wide angle was chosen for the larger field of view, and the RP system allowed for customization, alteration, and improvements. **The resolution was set on 1280x720 as a default (also recommended by most models).** Going to the next resolution size of 1920x1080 makes the calculations $(1920*1080)/(1280/720) = 2.25$ times more intensive. This was visible in the performance results, which went up to 0.1875s in the meantime, which is too high for the requirements. Going to a lower resolution does not make sense as it degrades performance, and the current resolution qualifies with respect to the time requirements. **With a faster processor, higher resolutions could be tried.** This is further discussed in the integration chapter.

3.9.3 Microphone

The threshold of the microphone was determined not to be lower than a constant 65dB sound. The trigger was set to be around 70 dB. This implies a loud clap or shout can be detected, but it is not triggered due to regular conversation, even at a very close range. **This resulted in a threshold of 250 above and below the mean of 330.** This is an integer value that comes from the ADC and has no unit. The implementation using the `void readMicrophone()` function is shown in Appendix B.

3.10 Towards Decisions

Misinterpreting the raw data that comes in through the sensors can lead to decisions based on nothing. This is shown as erratic behavior in the final robot. For this reason, adhering to the requirements and delivering the correct data is crucial. **If the models are not able to detect multiple people and where they are, the robot cannot act accordingly.** The next chapter discusses how all this interpreted data is used to make decisions on the robot's actions. Let us create the robot's brain.

Chapter 4

Decision

This chapter discusses the decision-making of the robot – the brain. Assuming the information that comes in from the interpretation stage is correct, it should decide on what to do or how to respond. This chapter explains how the input data is used to determine the robot's states but will gloss over the reasons why these states are selected; for a more in-depth discussion on why, the reader is pointed towards the other report [7].

4.1 Overview

Figure 4.1 shows an overview of the decision stage. The data packet from the RP in the interpretation stage is processed. This data needs some filtering and can be used to generate more data that could be of interest. Besides this, the internal clock of the Arduino is used to trigger many time-based events later. Using the clock, processed data, and loudness, the inputs for the decision-making are in place.

The decision-making is discussed in more depth. First, different methods are discussed, followed by a comparison and implementation.

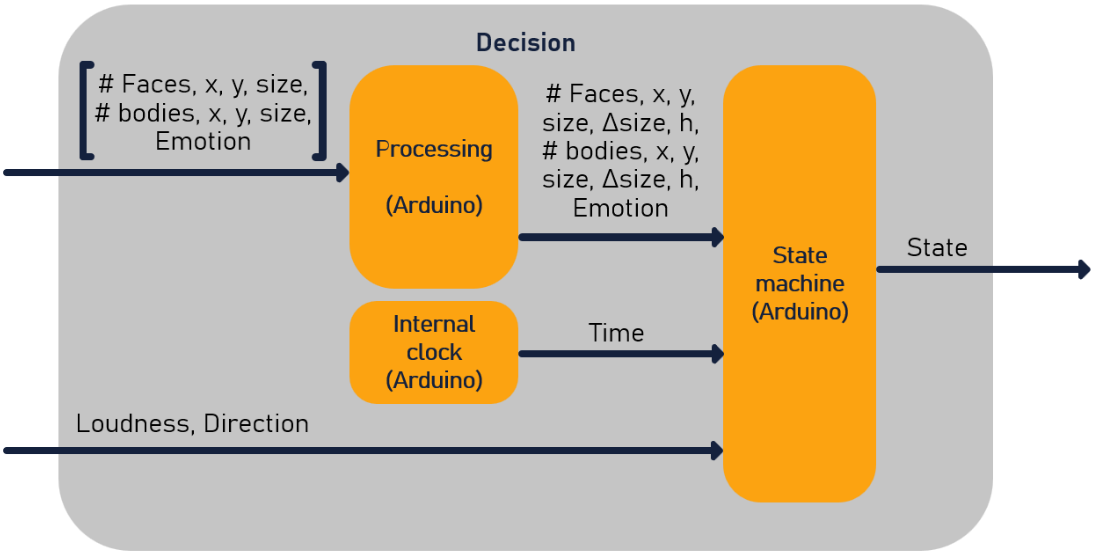


Figure 4.1: Schematic of the decision stage of the robot showing the inputs (data packet and loudness), the subsystems (in orange), and the output (state of the robot)

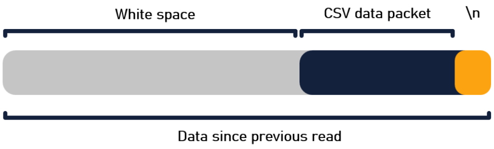


Figure 4.2: Serial data packet structure

4.2 Processing Data

Processing the data consists of three tasks. These tasks are reading the data packet from the serial port, using temporal filtering on the received data to remove errors, and using the time series of the data to extract features and generate more data.

4.2.1 Data Packet Reading

As the main loop function on the Arduino is running (Appendix C), the `void receiveData()` function is called every cycle. If the serial port has received data, it will read the data until a $\backslash n$ character. The packet structure is shown in Figure 4.2. When it reads the alignment character, it knows it has the whole data packet. It trims the whitespace before the packet and stores the values in an array with 32-bit integers. After this, the values are assigned to their corresponding variables. The implementation of this function can be found in Appendix B.

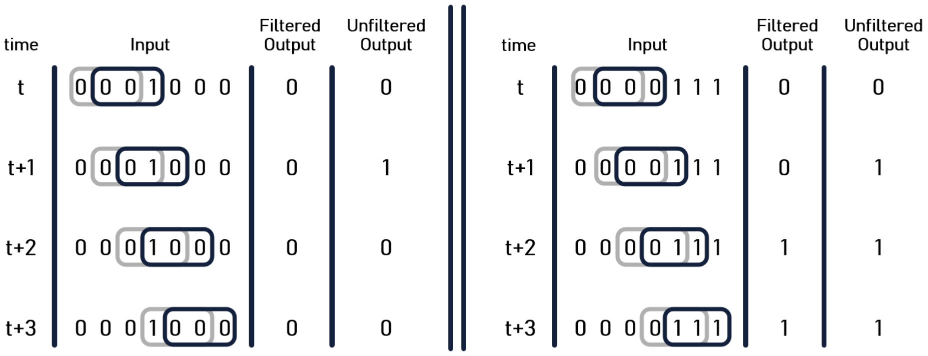
4.2.2 Filtering Data

The data coming in can have errors, e.g., due to wrong sensory input or misclassification. The latter is a more pressing issue. If a false positive or false negative occurs in the body or face recognition, the mistake needs to be spotted and removed. This is especially the case for values that are discrete and where minor differences have significant influences on the outcome. Other errors on larger discretized continuous signals, like the coordinates, require a different type of filter. Signal filtering over time is called temporal filtering and can be used to remove these types of errors. Let us discuss the more pressing errors to correct.

Number of faces and bodies

The number of faces and bodies is a discrete value that should not show high-frequency changes. In an analog circuit, this would require a low-pass filter. Because it is a discrete value, a simple implementation is majority voting. **Majority voting is a nonlinear, discrete analog of a digital low-pass filter.** It is different in terms of implementation but functionally works for this system. It uses a sliding window on the last N results, with N being the window width. Within this window, it looks at the count of each value and selects the value with the highest count.

An example of majority voting is shown in Figure 4.3. This example is for $N = 3$ and shows the window moving through a discrete time-space. On the left, it shows how this filter can successfully filter out unwanted glitches. The number one here could be any number, and the filter would still work. The downside of this filter is shown on the right; here, the phase change takes an additional timestep. The delay can be calculated as $t_{delay} = \lfloor N \rfloor$. **For this report, a window size of $N = 5$ was selected as almost all measured errors were either 1 or 2 frames.** Furthermore, the number of faces and number of people information is used only for slow transients.



time	Input	Filtered Output	Unfiltered Output	time	Input	Filtered Output	Unfiltered Output
t	0 0 0 1 0 0 0	0	0	t	0 0 0 0 1 1 1	0	0
t+1	0 0 0 1 0 0 0	0	1	t+1	0 0 0 0 1 1 1	0	1
t+2	0 0 0 1 0 0 0	0	0	t+2	0 0 0 0 1 1 1	1	1
t+3	0 0 0 1 0 0 0	0	0	t+3	0 0 0 0 1 1 1	1	1

Figure 4.3: Filtered and unfiltered outputs over time for two input sequences for a majority voting filter with window size $N = 3$, with error handling on the left and a transition on the right

Coordinates and Body Size

Because of the continuous nature of the coordinates and body size, a different filter is required. The required filter is still a low-pass filter: a Moving average filter. The main error that needs to be corrected is a single frame false negative on a person when there are multiple people in the room. The symbiote looks at the largest bounding box. As discussed in the next section, this is used to estimate who is closest. If there is a person in the left part of the frame with a larger bounding box than a person in the right part of the frame, it will look at the first. If, for one frame, the person on the left is not detected, the coordinates suddenly change, and in the next frame, they change back. These errors can cause very erratic behavior. Especially because the eye follower will use the coordinates.

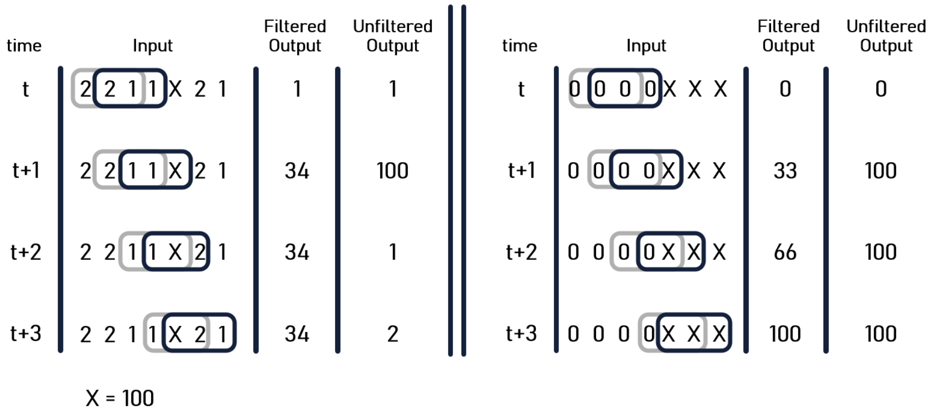
In Equation 4.1, the mathematical concept of this filter is shown. Here, N is the window size, and all values in this window are summed and divided by N. Examples of how this filter works are given in Figure 4.4. On the left, it shows how it handles such an erroneous situation, as described above. A sudden significant change in the value, when unfiltered, causes a considerable change. This situation would also be one where a majority voting system would struggle as all inputs are different. **When a moving average is applied, it ”spreads out” this error over multiple time stamps.** This is also why it is sometimes called a smoothing filter. The smoothing effect depends on the window size. However, again, the window size also decreases response time, as can be seen on the right.

Within the software implementation, it is possible to make this window size dynamic per state. This can help simulate the slower response time in a relaxed state as compared to an alert state. Because it influences expression, it will be discussed further in that chapter.

$$\text{SMA}_t = \frac{1}{N} \sum_{i=0}^{N-1} x_{t-i} \tag{4.1}$$

Emotions

The emotions were eventually not implemented. However, a majority filter is advised for this application. The discrete nature and the relatively slow changes make this a well-suited filter. The window size for the filter requires testing.



time	Input	Filtered Output	Unfiltered Output
t	2 2 1 1 X 2 1	1	1
t+1	2 2 1 1 X 2 1	34	100
t+2	2 2 1 1 X 2 1	34	1
t+3	2 2 1 1 X 2 1	34	2

X = 100

time	Input	Filtered Output	Unfiltered Output
t	0 0 0 0 X X X	0	0
t+1	0 0 0 0 X X X	33	100
t+2	0 0 0 0 X X X	66	100
t+3	0 0 0 0 X X X	100	100

Figure 4.4: Filtered and unfiltered outputs over time for two input sequences for a moving average filter with window size $N = 3$, with error handling on the left and a transition on the right

4.2.3 Generating More Data

Two more interesting features can be extracted from the incoming data. These are calculated and also used as input by the decision-making subsystem.

Difference in Body and Face Size

Using the fact that the data is collected in a time series means that changes in values can be measured. Not all changes ″mean″ anything useful. However, the change in body and face size is of more interest. **Just as the size is interpreted as distance, the change in body size can be detected as a change in this distance, i.e., movement speed towards or away from the robot.** This is visualized in the left part of Figure 4.5.

To calculate the difference in body size, a discrete calculation method is used as in Equation 4.2. It is important to note that $\Delta bodysize$ is not a good measure for the change in distance from the robot. This is because of the proportionality as shown in Equation 4.3. However, it is essential to consider the implications. This information is used to assess sudden movements towards the robot, which will make it scared when higher than a threshold. For that reason, the squared nature can be of great use. **The symbiote is more sensitive to movements that occur close to it.** The body size calculation is performed on the filtered body size values.

$$\Delta bodysize = bodysize_t - bodysize_{t-1} \tag{4.2}$$

$$\Delta bodysize \propto \Delta s^2 \tag{4.3}$$

Height

The height of a person in the frame can also provide helpful information. A higher person means the person is more dominant. An example is shown on the right of Figure 4.5. Here, the person sits on their knees, and their height value is lower. The height is only used when a face is detected, which means the person is already close. Similar to the change in body size, this feature is also more sensitive when people are closer. This is because the camera is placed far below head level and tilted up. This means the closer you are, the higher you are in the frame. **Thus, a closer person has to stay lower to not go above a certain height threshold.**

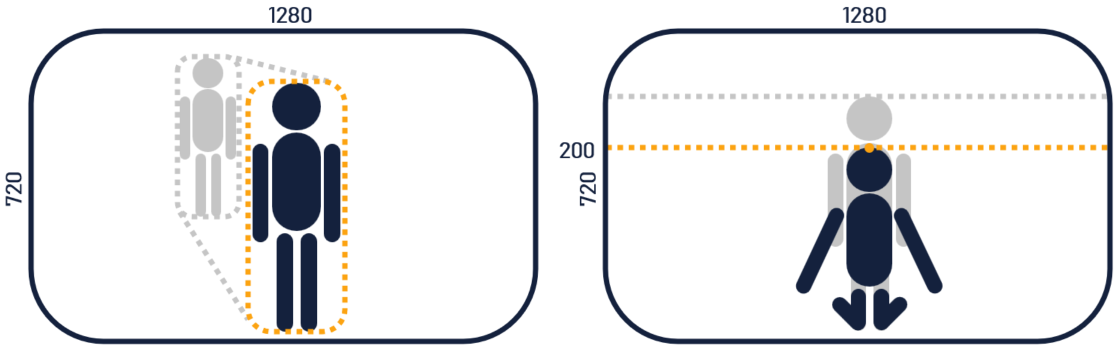


Figure 4.5: Two types of generated data, with on the left a difference in body size and on the right the height

4.3 Internal Clock

The Arduino Wifi Rev 2 uses a 16 MHz quartz crystal oscillator as their system clock. This oscillator drives the main CPU and counters used for timing functions. The function that is used throughout the code to set timers is the `millis()` function. This function reads the counter and returns the number of milliseconds since startup. Because the function relies on hardware timers and interrupts, it is non-blocking. This method allows for accurate relative time measurements with a resolution of 1 ms. Overflows of the timer take place every $2^{32}ms \approx 50days$. For this project, this is not a concern, but this should be addressed for longer applications. A solution could be a daily reset through the RP when the robot is charging.

4.4 Decision Making

Choosing a method for decision-making is far from arbitrary. It determines complexity, functionality, and predictability. Many methods exist, and all have advantages and disadvantages. Multiple methods need to be discussed to make a good choice.

4.4.1 Methods

The methods in this section can be categorized into two groups. Firstly, the machine learning methods and, secondly, the traditional methods.

Machine Learning

Machine learning refers to a class of algorithms that enable a system to identify patterns and make predictions or decisions based on data [39]. In robotics, it allows a robot to learn from sensor inputs and adapt to new situations without being explicitly programmed for every possible scenario. A machine learning model can be trained offline using historical or simulated data and then deployed on the robot.

Reinforcement Learning

Reinforcement learning builds on machine learning. With this method, an agent learns to make decisions by interacting with its environment [40]. It receives rewards or penalties based on its actions and uses this feedback to improve its behavior over time. This method is often used to learn complex behaviors through trial and error, such as walking or object manipulation, but can require extensive training and computational resources.

Finite State Machine

An FSM is a mathematical model composed of a finite number of states. Transitions between these states are only based on inputs. In an FSM, not all states have to be connected to all other states through these transitions. Each state could represent a specific behavior, e.g., with the soccer robot in [41]. The robot switches states according to predefined rules. FSMs are simple, deterministic, and easy to implement, making them suitable for structured tasks where behavior can be broken down into clear stages.

Rule Based

A rule-based system makes decisions by evaluating a set of ”if-then” rules [42]. Each rule defines a condition and a corresponding action. These systems are transparent, easy to debug, and work well when domain knowledge is available, and behavior can be encoded explicitly. However, they can become challenging to manage as the number of rules increases or when dealing with uncertainty.

Decision Tree

A decision tree is a hierarchical model that splits data into branches based on feature values, ultimately leading to a decision at each leaf node [43]. It resembles a flowchart, where each node represents a test, and each branch corresponds to the outcome.

Bayesian Network

A Bayesian network is a probabilistic graphical model that represents relationships among variables using a directed acyclic graph [44]. Each node represents a variable, and edges encode probabilistic dependencies. Bayesian networks are used to reason under uncertainty by combining prior knowledge and observed data. They are effective for tasks such as sensor fusion or decision-making in uncertain environments.

4.4.2 Comparison

First, let us discuss the largest difference among the methods. Both the neural network methods and the Bayesian network method are effectively non-deterministic. This means it is nearly impossible to obtain predictable behavior. This breaks Req M.14. Besides this, all three require extensive knowledge or data of similar situations, which is not available. Lastly, the case of [20] is used. **Building intelligence requires a fundamental understanding of the system.** For these reasons, these methods were not selected.

Decision trees are hierarchical and input-driven. This means that they respond to inputs using different levels of importance. Not only does this make it more challenging and progressively complex to add sensors to the robot, but it also makes it nearly impossible to remove them without fundamentally changing its behavior. **When using an input-driven system, it is also more difficult to prevent specific transitions.**

Rule-based systems and FSMs are sequential or can be made as such. This means that rules can follow each other instead of being hierarchically structured. The reason for choosing an FSM over a rule-based system is mainly due to FSMs knowing its previous state. This allows for better and more natural progressions of emotions. It is easy to prevent a transition from fearful to sleeping. Rule-based systems are better for situations where domain knowledge is well known. **In contrast, an FSM is better for situations where behavior can be broken down into states, i.e., emotions.**

4.4.3 Implementation

Now that the choice of the FSM is made, the implementation can be discussed. A FSM consists of three components: Inputs, states, and transitions. Lastly, one of the states needs to be the starting state. The output depends only on the state the robot is currently in. The output is discussed in the next chapter.

Inputs

The first thing to define for a FSM are the inputs. These drive the transitions. Let's define all the inputs here:

- (**S**) Sound level exceeds a predefined threshold.
- ($\mathbf{T_s}$) The current state duration timer exceeds its threshold.
- ($\mathbf{T_p}$) Elapsed time since no individuals have been detected.
- (**P**) Presence of at least one detected person; !P indicates no one is present.
- (**#B**) Number of detected bodies (assumes no faces were detected); **B** indicates $\#B \geqslant 1$.
- (**#F**) Number of detected faces; **F** indicates $\#F \geqslant 1$.
- ($\mathbf{B_s}$) Body size.
- ($\mathbf{F_s}$) Face size.
- ($\mathbf{\Delta B_s}$) Temporal change in detected body size.
- ($\mathbf{\Delta F_s}$) Temporal change in detected face size.
- (**H**) Estimated height of the closest detected face.

States

Similarly, let us discuss the states. The states are categorized into emotional states and transitional states. **The transitional states purely function as transitions from certain states to other states.** They show a predefined animation, and when this animation is completed, they always go to the next state. The transitional states are:

- Go to sleep
- Wake up
- Wake fast

The emotional states are the states that allow for interaction with one or multiple people. When in a state, it can stay there as long as the inputs do not change. The expressions of these states are discussed in the next chapter. These states are:

- Sleep (starting state)
- Relaxed

- Interest / Alert
- Stressed
- Playful
- Fearful
- Defensive

Transitions

Now that the states and inputs are defined, the transitions can be determined. The transitions are shown in Table 4.6. The starting state is the sleep state, meaning on startup or reset, it starts there. All transitions can only happen after the minimum state duration T_s has elapsed.

Current State	Next State	Condition(s)
Sleep	Wake up	P (presence detected)
	Wake fast	S (sound above threshold)
	Sleep	Else
Wake up	Relaxed	Always
Wake fast	Stressed	$\#B = 1$ or $\#B = 2$
	Fearful	F or $\#B \geqslant 3$
	Relaxed	Else
Relaxed	Interest/Alert	F (face detected)
	Stressed	$\#B \geqslant 2$ or S
	Go to sleep	T_p (timeout: no people)
	Relaxed	Else
Interest/Alert	Playful	$H < h$
	Stressed	$\#B \geqslant 2$
	Relaxed	$!F$
	Interest/Alert	Else
Playful	Stressed	$\#B \geqslant 2$ or $H > h$
	Interest/Alert	$!F$
	Playful	Else
Stressed	Fearful	$\#B \geqslant 3$ or $H > h_2$ or S
	Defensive	$H < h_2$
	Relaxed	$\#B \leqslant 1$
	Stressed	Else
Fearful	Stressed	$\#B \leqslant 2$ and $!F$
	Fearful	Else
Defensive	Fearful	$\#B \geqslant 3$ or $H > h_2$
	Defensive	Else
Go to sleep	Sleep	Always

Table 4.6: Transition table for the finite state machine of the robot

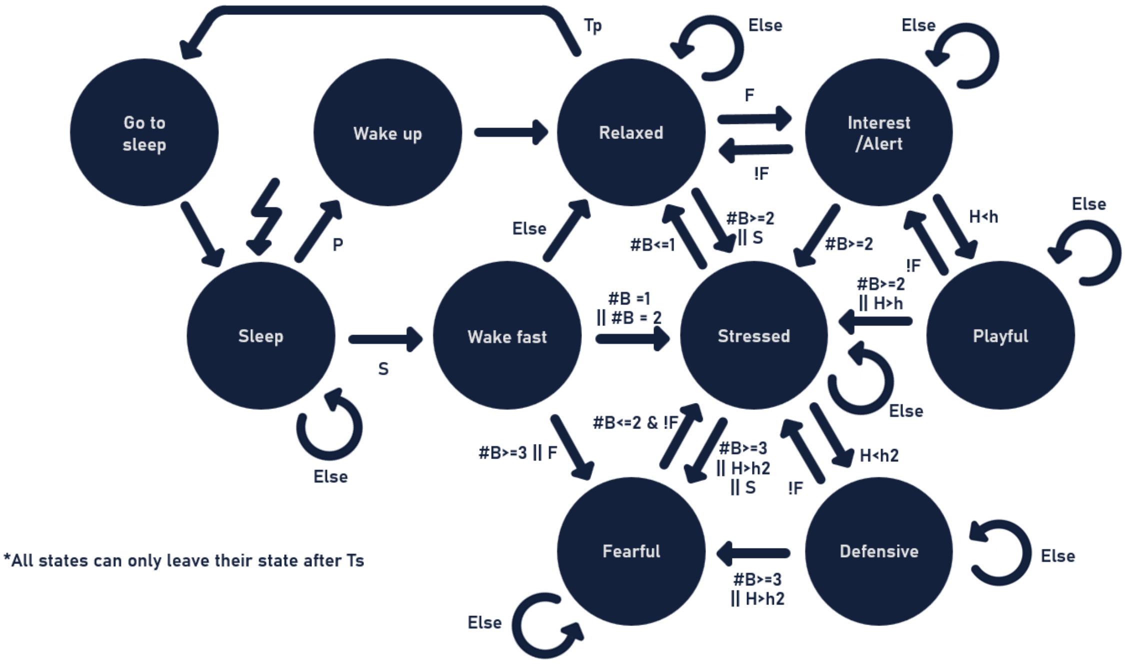


Figure 4.7: Finite State Machine (FSM) of the robot with all transitions and transitional rules

The State Machine

Now that everything is discussed, the FSM can be presented. It is shown in Figure 4.7. The implementation can be found in the function `void doState()` and `void goToState()` in Appendix C respectively. Let us discuss all the states and what they should represent.

- **Sleep:** The default idle state. Transition to active states occurs either upon detecting a person (P) and waking up slowly, mimicking how animals feel vibrations in the ground, or hearing a sound above a threshold (S) and waking up fast, mimicking being scared awake. Otherwise, the robot remains in sleep mode.

- **Wake up / Wake fast:** Transition states activation states. If a human is present (P), the robot wakes up calmly; if a sudden sound is detected (S), it enters a heightened state of alertness via *Wake fast*. From here, it can transition to *Fearful* if someone is close F or many people are there $\#B \geqslant 3$, *Stressed* if one or two people are detected or *Relaxed* if no one is seen.

- **Relaxed:** A calm state post-activation. It shifts to *Interest/Alert* if a face is detected (F) or escalates to *Stressed* if multiple bodies are detected ($\#B \geqslant 2$) or a loud sound is made. If no one is seen for the T_p duration, it goes back to sleep via *Go to sleep*.

- **Go to sleep:** shuts down the eyes slowly and goes to *Sleep*.

- **Interest / Alert:** An attentive state. If the closest person is perceived as non-threatening ($H < h$), the robot enters a *Playful* state. If the perceived height is large ($H > h$) or more bodies are detected, it goes to *Stressed*, but if no face and just one body are detected, it goes back to *Relaxed*.

- **Playful:** A socially receptive state. It returns to *Interested / Alert* if visual cues disappear ($!F$). If the situation becomes more crowded ($\#B \geqslant 2$) or if the person scares it ($H > h$), it goes to *Stressed*.

- **Stressed:** A central node representing elevated tension. Depending on contextual cues, the robot may escalate to *Fearful* if many bodies ($\#B \geqslant 3$), a face is high ($H > h_2$), or a loud sound is made(S). It goes to *Defensive* if the person closest is less dominant and stays low ($H < h_2$). It can also de-escalate to *Relaxed* when there are one or fewer people.
- **Fearful:** Reactive state where it is easy to arrive but only one way to leave. This is through the *Stressed state* when there is no one close $!F$ and at most two people.
- **Defensive:** Reactive state where the perceived threat level is low for the robot. It can go back to *Stressed* or if the person goes back $!F$ or go to *Fearful* if the person stands too tall or more people come.

4.5 Manual Control

For debugging, testing individual components, and performing user tests, a manual mode was created. This manual mode can be turned on and off with `#define CONTROLLERUSEFLAG true` in the code in Appendix D. When this is turned on, the state transitions are deactivated, and the robot remains in the state it is in until the Bluetooth controller is used (can be found in Appendix A). All code handling the controller is shown in Appendix C.

The controller shown in Figure 4.8 can be used in two ways. The first way is controlling the appendices; this is called manual control appendices. In this way, the robot does not perform any state or emotion; only three special states are used.

- **Antennae:** In this state, entered through the Y button, the joysticks control both antennae.
- **Eyes:** In this state, entered through the X button, the left joystick controls the eyes, and the right joystick controls the eyelids.
- **Wheels:** In this state, entered through the A button, the left joystick performs driving forward and backward with steering from right to left. The right joystick performs steering in place.
- **Home:** Homes all the appendices and stops motors when the B button is pressed.

In the other mode, manual control states can be selected. The selection of the states is shown in the figure. In this mode, the appendices can not be moved.

4.6 Towards Actuation

Now that the decision on the state is made, **"what to do?"** the last step is actuation, **"how to do it?"**. This is discussed in the next chapter.

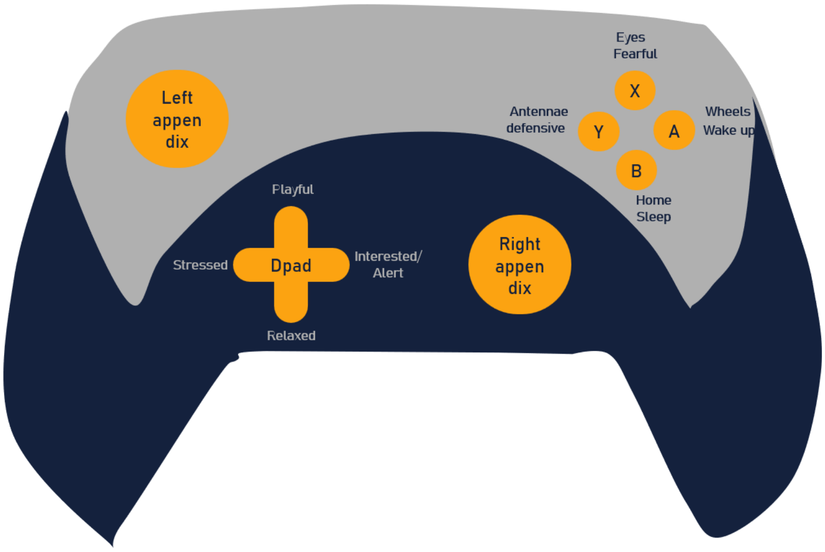


Figure 4.8: Controller inputs, with on top manual control appendices and the bottom manual control states

Chapter 5

Actuation

Now that the robot can see, hear, understand, and decide, it should be able to move. The robot should convert the desired emotions determined in the previous stage into a perceived emotion that is distinguishable and detectable, as in Reqs M.15 and M.16. **Detectable means a person should interpret it correctly when it is shown. Distinguishable means it is different from the others in that it is not falsely detected within another state.**

5.1 Overview

Figure 5.1 shows an overview of the actuator's stage. The input coming from the decision stage is the desired emotion. Using the expression subsystem, the angles or speed and direction for all motors are determined. These values are sent to the motor drivers, which power and signal the motors. All motor movements together are the behavioral output of the robot. This is the perceived emotion by the human.

5.2 Expression

A software structure is required to allow the robot to express itself. This approach is formalized in a hierarchical architecture, where each behavioral layer operates semi-independently,

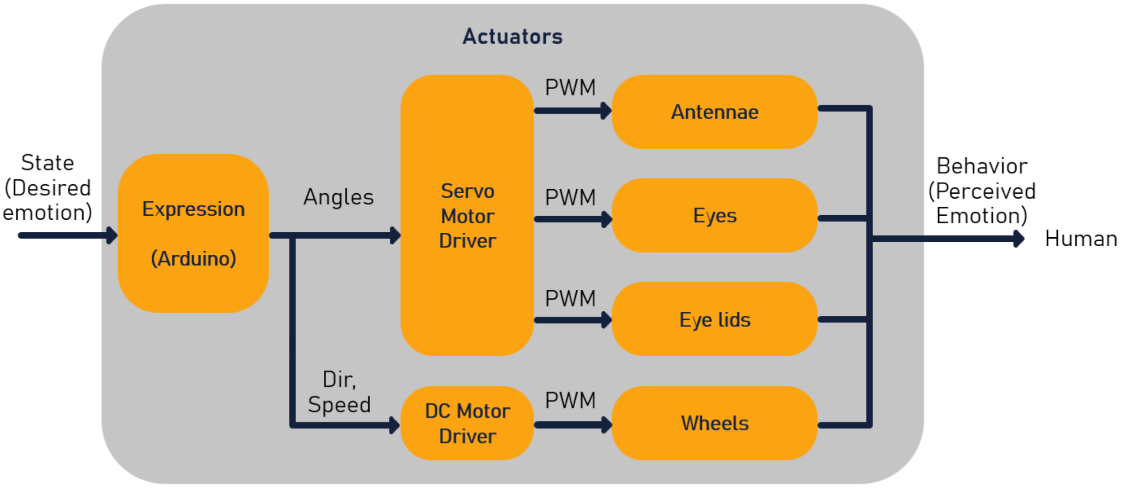


Figure 5.1: Schematic of the actuator stage of the robot showing the input (the state or desired emotion), the subsystems (in orange), and the output (behavior in the form of motion or the perceived emotion)

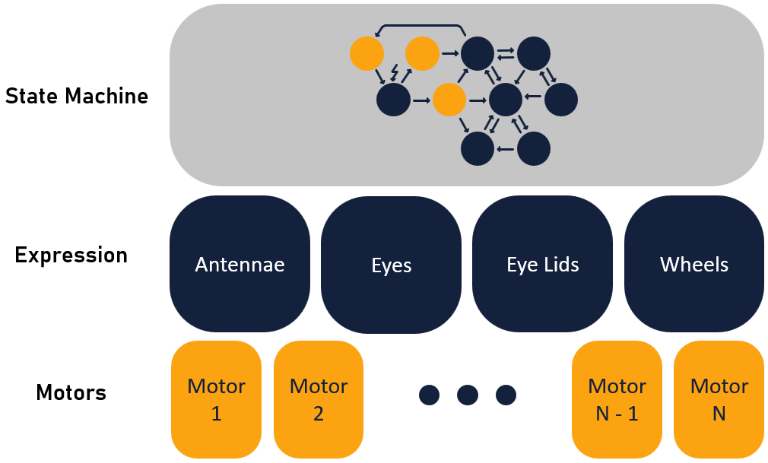


Figure 5.2: Hierarchical behavioral structure of the robot with the state machine as a top-level, motors as the bottom-level, and in dark blue the expression level.

with higher levels built atop lower reactive mechanisms. Lower layers—such as those responsible for perception or basic motor responses—function continuously and autonomously, ensuring that fundamental behaviors are always available and robust. Higher-level layers, such as those responsible for strategic decision-making or social interaction, can access and modulate the activity of lower layers but do not serve as prerequisites for their operation. This asymmetric dependency ensures system resilience: **even in the absence or failure of higher-level modules, the lower-level behaviors remain functional.** Such a structure promotes robustness, adaptability, and real-time responsiveness in dynamic and uncertain environments, aligning closely with the demands of embedded and autonomous robotic systems.

Applying this to the robot helps in structuring the expressive functions, building from lower-level actuation signals, i.e., angles, speed, and direction, that function autonomously to higher-level functions that respond to state changes. In the context of this robot, it could be shown schematically as in Figure 5.2. Each component in the blue layer of expression will have a similar hierarchical structure. These four actuation subsystems are explained and shown below. However, before diving into the actuation, here is a quick remark on two things.

5.2.1 Switching States

As mentioned in the previous chapter, when a new state is entered, the state timer T_s is reset using the internal clock. A new state is always entered using the `void goToState ()` function in Appendix C. This handles several things which will be used in the expressive functions discussed later.

- **Timer Handling:** It resets all timer values to the current time using `millis()`.
- **Enter state flag reset:** The enter state flags are used to perform actions upon entering a state.
- **Angle reset:** It sets all start and end angles to the current angle. This is done to recalculate movements using the speeds and acceleration of the new state. It also resets the start time values that are used for angle calculations.

5.2.2 Global Variables

Some of the values used are set globally so that every function can use them. These are divided into constants and variables, shown in Appendix D and C respectively. The constants are:

- **Threshold values:** Used to alter behavior.
- **Angle modes:** These include minimum, maximum, and homing angles for all servos in all states
- **Delay and minimum movement:** These values are used for the randomization in movements that are described later. The delay gives the range of delays possible between movements, and the minimum movement removes small movements.
- **Speed and Acceleration:** Both these values are set for all states and used in the movement calculations.

The variables are:

- **States and timers:** Both the current state and the timer values are accessible by all functions.
- **Angle values:** All current angles can be accessed, as well as the start angle and end angle for every movement.
- **Serial communication values:** All variables received from the serial communication are stored globally, as well as the values directly calculated from them.

5.2.3 Structure

All four subsystems, antennae, eyes, eyelids, and wheels, are in the same hierarchical tier. They can also be subdivided into the same four hierarchical sub-tiers. These tiers are from bottom to top: Motor control, Calculating movement, Basic movement, and Expressive movement. These tiers handle:

- **Motor control:** Handles the interface with motors. When given an angle, speed, or direction from the tier above, it converts it to motor instruction.
- **Calculating movement:** Calculates a smooth path from A to B for the servos. It uses variables such as speed, acceleration, and start time. The DC motors perform the steering and driving instructions. This uses variables such as direction, steering, and throttle.
- **Basic movement:** This tier creates basic movement sets by providing the timing and location variables that the calculation tier uses.
- **Expressive movement:** Lastly, this tier combines basic movements and movement sequences to create expressive movements. These movement sets can be selected by the state machine using the state variable.

5.2.4 Antennae

Starting with the expression of the antennae as shown in Figure 5.3. The blue and grey squares represent functions on that tier. They are named according to their respective names in the code. Because lower-level functions are independent of top-level functions, they are discussed bottom up. Lastly, the blue squares are implemented in the final design, and the grey squares are not but are suggestions to be added.

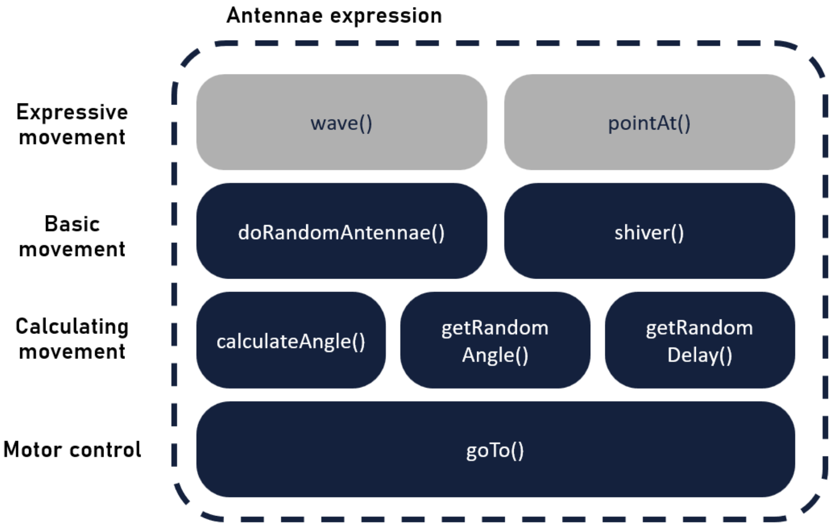


Figure 5.3: Hierarchical structure of the antennae expression with the top level being Expressive movement and the bottom level being motor control, with the blue (implemented) and grey (not implemented) squares showing the functions on that tier

`void servoGoTo(uint16_t angle, uint16_t servoNumber)`

The function `servoGoTo` in Appendix C is designed to control the angular position of a servo motor. It receives as input the target angle and the identifier of the servo motor that should perform the motion. The angle must fall within the servo's operating range.

To avoid redundant operations, the function first checks whether the desired angle differs from the previously set value, stored in a global array named `servoAngle[]`. This is done as the communication with the motor driver is limited by the clock speed of the I^2C protocol (100kHz), which is magnitudes slower than the Arduino clock speed (16MHz). No action is taken if the requested angle is the same as the current one. Otherwise, the array is updated. The conversion from angle to motor control signal is further discussed in the section on the motor driver.

Using just this function to control the motors would result in unnatural behavior. The distance and speed profiles are shown in Figure 5.4. This shows a non-continuous speed function and a non-differentiable function for the distance or angle.

`float calculateAngle(float startAngle, float endAngle, float maxSpeed, float maxAccel, unsigned long servoStartTime)`

The function `calculateAngle` in Appendix C implements a time-dependent trapezoidal motion profile for smooth servo control. It ensures that movement from a given start angle to a target angle occurs within defined physical constraints, namely maximum speed and acceleration. It prevents sudden velocity changes that show unnatural movement. The input parameters are:

- `startAngle`: Initial angle (degrees)
- `endAngle`: Final desired angle (degrees)
- `maxSpeed`: Maximum angular velocity (degrees per second)

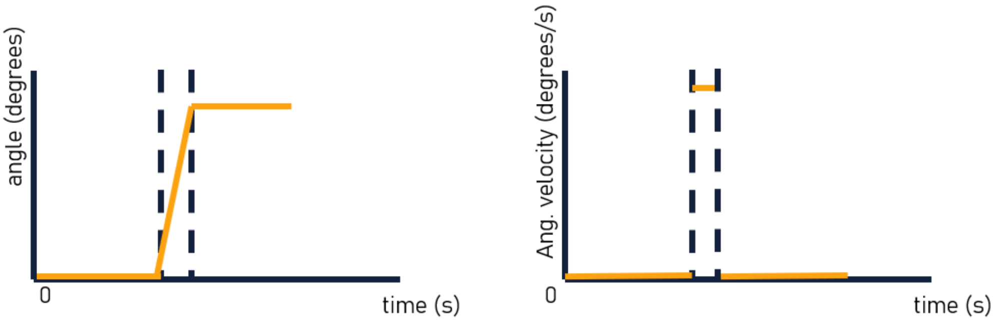


Figure 5.4: Motion profile of the `servoGoTo` function, with the distance over time on the left and velocity over time on the right

- `maxAccel`: Maximum angular acceleration (degrees per second squared)
- `servoStartTime`: Time at which the motion started (in milliseconds)

The mathematical derivation of this calculation is as follows:

1. Compute the elapsed time since motion started using:

$$t = \frac{\texttt{millis()} - \texttt{servoStartTime}}{1000.0}$$

2. Calculate the total angular distance to travel:

$$d = |\theta_{\text{end}} - \theta_{\text{start}}|$$

3. Compute the time required to accelerate to maximum speed:

$$t_{\text{accel}} = \frac{v_{\text{max}}}{a_{\text{max}}}$$

4. Calculate the distance covered during acceleration:

$$d_{\text{accel}} = \frac{1}{2} a_{\text{max}} t_{\text{accel}}^2$$

5. Check whether a constant-speed cruising phase is needed:

   - If $2d_{\text{accel}} < d$, then:

$$d_{\text{cruise}} = d - 2d_{\text{accel}}, \quad t_{\text{cruise}} = \frac{d_{\text{cruise}}}{v_{\text{max}}}$$

   - Otherwise, use a triangular profile (max speed not reached):

$$t_{\text{accel}} = \sqrt{\frac{d}{a_{\text{max}}}}, \quad d_{\text{accel}} = \frac{1}{2} a_{\text{max}} t_{\text{accel}}^2$$

6. Compute the total motion time:

$$t_{\text{total}} = 2t_{\text{accel}} + t_{\text{cruise}}$$

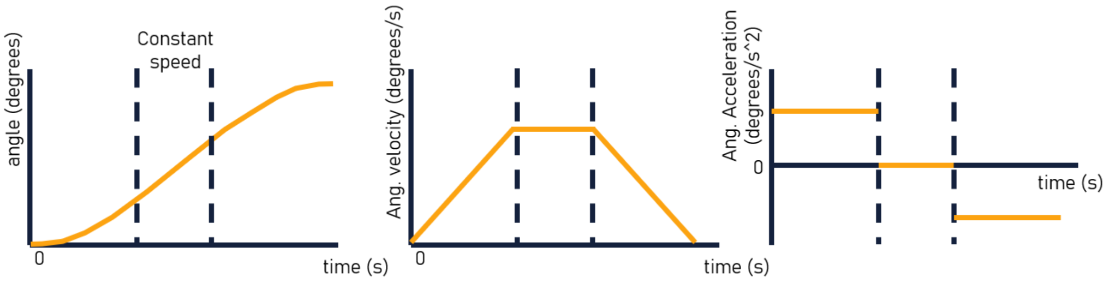


Figure 5.5: Time-dependent trapezoidal motion profile of the `calculateAngle` function, with the distance over time on the left, velocity over time in the middle, and the acceleration over time on the right

7. Determine the angle based on the current phase:
   - **Acceleration phase** $(t \leqslant t_{\text{accel}})$:

     $$\theta(t) = \theta_0 + \frac{1}{2}at^2$$

   - **Cruise phase** $(t_{\text{accel}} < t \leqslant t_{\text{accel}} + t_{\text{cruise}})$:

     $$\theta(t) = \theta_0 + d_{\text{accel}} + v(t - t_{\text{accel}})$$

   - **Deceleration phase** $(t > t_{\text{accel}} + t_{\text{cruise}})$:

     $$\theta(t) = \theta_0 + d_{\text{accel}} + d_{\text{cruise}} + vt - \frac{1}{2}at^2$$

8. Avoid overshoots:

   $$\theta(t) = \begin{cases} \theta_{\text{end}}, & \text{if exceeded target} \\ \theta(t), & \text{otherwise} \end{cases}$$

Following these steps can give two different speed curve outputs: one where the cruising speed or constant speed is reached and one where it is not reached. Figure 5.5 shows the case where the constant speed is reached, with the corresponding acceleration profile. This function results in a continuous speed profile. When the cruising speed is not reached, the acceleration phase is followed directly by the deceleration phase.

`float getRandomAngle(float currentAngle, int servoNumber)`

The function `getRandomAngle()` in Appendix C is used to generate a randomized target angle for a servo motor, starting from its current position. This function ensures that the newly generated angle falls within predefined mechanical limits and state limits while also guaranteeing a minimum angular deviation from the original position.

The function uses the current angle and the servo number. Internally, the function references state-dependent arrays `SERVOMINANGLE`, `SERVOMAXANGLE`, and `RANDOMANGLE`, which define the lower and upper angular bounds and the minimum change threshold, respectively, for each servo in a given state.

The function operates by repeatedly generating a random offset (`randomDelta`) and applying it to the `currentAngle`. The offset is selected uniformly from the interval:

$$[-(angle_{\text{max}} - angle_{\text{min}}),\ (angle_{\text{max}} - angle_{\text{min}})]$$

where $angle_{\text{min}}$ and $angle_{\text{max}}$ are constant values given in Appendix D. The resulting candidate angle is then clamped within the defined bounds to prevent exceeding mechanical limits:

$$angle_{\text{new}} = \min\left(\max\left(angle_{\text{candidate}}, angle_{\text{min}}\right), angle_{\text{max}}\right)$$

This process continues iteratively until the absolute difference between the `newAngle` and the `currentAngle` exceeds the predefined threshold:

$$|angle_{\text{new}} - angle_{\text{current}}| \geqslant \texttt{RANDOMANGLE[state][servoNumber]}$$

The final value of `newAngle` is returned and will always be within the defined maximum and minimum angles. These limits simulate the physical constraints of the robot. Other random distributions could be implemented to create more natural behavior.

`unsigned long getRandomDelay(int servoNumber)`

The function `getRandomDelay()` in Appendix C is responsible for introducing a random delay before initiating the following action of a specified servo motor. It takes the `servoNumber` and computes a pseudo-random delay. It returns an `unsigned long` integer representing the delay time in milliseconds.

`void doRandomAntennae()`

The function `doRandomAntennae()` in Appendix C implements randomized, bounded motion for the antennae. This basic motion uses the three functions from the lower tier to provide randomized natural idle movement for a given state.

The function iterates over a predefined range of servo indices that represent the antennae, corresponding to the antennae actuators. For each servo i, it goes through a logic loop. If the current angle equals the end angle (`servoAngle[i] == servoEndAngle[i]`), it has finished its motion. If this is the case, it sets the start angle and calculates a new target position using `getRandomAngle()` and a new start time using `getRandomDelay()`. However, if the motion is not completed, it performs the motion using the `calculateAngle()` function.

One tire higher, the motion profile looks like the graph in Figure 5.6. It shows a more natural behavior with movements following one another after a random delay within the boundaries. Adjusting the values for the angles and delay per state provides the different expressions of the antennae.

`void shiver(int degrees)`

The `shiver()` function in Appendix C simulates a trembling or jittery motion in the robot's antennae, giving the impression of a shiver. It achieves this by applying small, rapid angular deviations. The degrees parameter determines the range of motion of the shiver.

It uses a normal distribution to find a new angle that is at most the given degrees away from the homing angle. It uses the `goTo()` function to utilize the maximum servo speed. It is used in the fearful state.

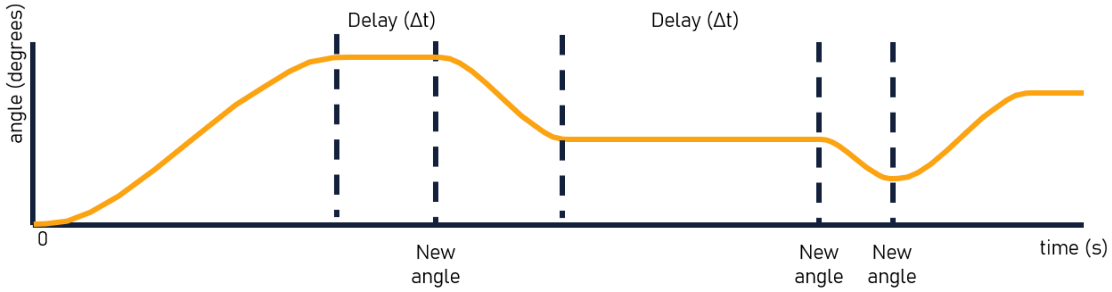


Figure 5.6: Motion profile of the `doRandomAntennae` function, with the distance over time, marked with random delays and random new angles

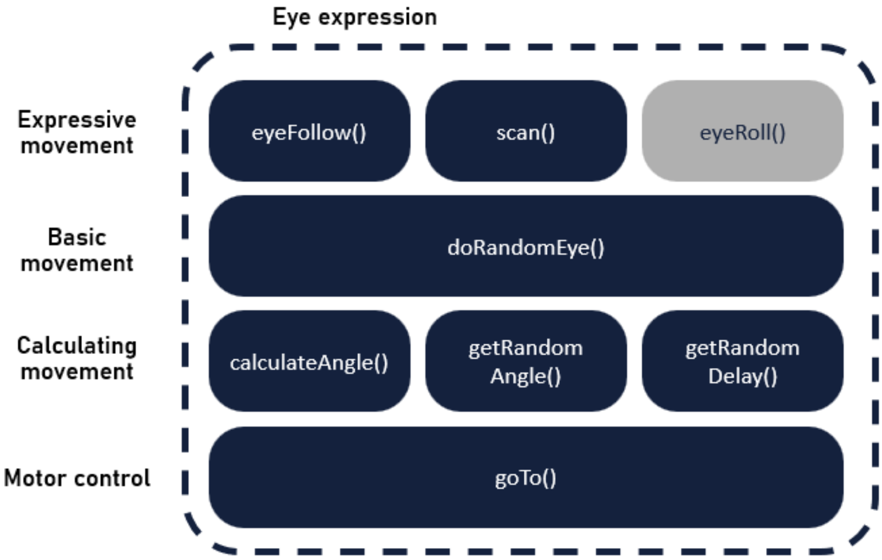


Figure 5.7: Hierarchical structure of the eye expression with the top level being Expressive movement and the bottom level being motor control, with the blue (implemented) and grey (not implemented) squares showing the functions on that tier

`wave()` and `pointAt()`

The two functions `wave()` and `pointAt()` were not implemented but are suggested as the next steps. The wave function could be used to attract attention. Whereas the point of function could be used to divert attention to another point of interest.

5.2.5 Eyes

The expression hierarchy of the eyes is shown in Figure 5.7. The lower two tiers are the same as those of the antennae. This is the beauty of using hierarchical structures. These lower-tier levels are the basis for many more complex movements.

The basic movement tier is also similar. However, the `doRandomEye()` function in Appendix C selects different servos and thus uses different boundary conditions and delays.

void eyeFollow()

The function `eyeFollow()` in Appendix C is responsible for looking at the person to show 'attention.' It has three actions it can perform based on priority. The function begins by checking whether any faces have been detected, as indicated by `numFaces > 0`. If this is the case, it computes the target angles by mapping the screen coordinates of the detected face `(faceX, faceY)` to the output angle:

$$\texttt{angleX} = \texttt{map}(\texttt{faceX}, \texttt{SCREENX}, 0, \texttt{SERVOMINANGLE}[0][\texttt{SERVO5}], \texttt{SERVOMAXANGLE}[0][\texttt{SERVO5}])$$

$$\texttt{angleY} = \texttt{map}(\texttt{faceY}, 0, \texttt{SCREENY}, \texttt{SERVOMINANGLE}[0][\texttt{SERVO6}], \texttt{SERVOMAXANGLE}[0][\texttt{SERVO6}])$$

This mapping is effective because the camera and the eyes are located at the same physical location. Before using the angles, they are checked so as not to exceed the eyelid locations to prevent the pupils from disappearing behind them and creating a 'demonic' look.

If there is no face detected, the robot looks for visible bodies (`numBodies > 0`), and a similar mapping process is executed using the body coordinates `(bodyX, bodyY)`.

If neither faces nor bodies are detected for a duration exceeding the (`eyeFollowTimerThreshold`), the function calls `doRandomEye()`, which induces randomized movement that mimics 'looking around' or 'looking for someone.' This ensures that the robot's gaze remains dynamic and lifelike, even in the absence of a tracked target.

bool eyeScanRoom(unsigned long startTime,int delayBetweenMovement)

The function `eyeScanRoom()` implements a predefined scanning pattern for the robot's eyes, simulating scanning a room. It is only used in the fast wake-up when a loud noise wakes it up. It takes two parameters: `startTime`, which marks the moment the scan began, and `delayBetweenMovement`, which sets the interval in milliseconds between the scanning movements.

Internally, the function compares the elapsed time since `startTime` with a sequence of thresholds derived from a constant base delay `standardDelay` and the user-supplied `delayBetweenMovement`. The eyes look in all four corners, which means all maximum, minimum, x, and y combinations.

This temporal sequencing of servo positions allows the robot to perform a sweeping visual gesture, mimicking the natural scanning of a room by a pair of eyes. The function returns `false` during all intermediate stages and only returns `true` once the full scanning cycle is complete.

5.2.6 Eyelids

The eyelids also use the `goTo()` and `calculateAngle()` functions as is shown in Figure 5.8. It does not need random angles or delays and performs its basic movements differently. The eyelids are mechanically constructed in such a way that the lids of both eyes are controlled separately. However, the lids in a single eye close and open together. Thus, when the top lid goes down, the bottom lid goes up by the same amount, and if fully closed, they meet in the middle.

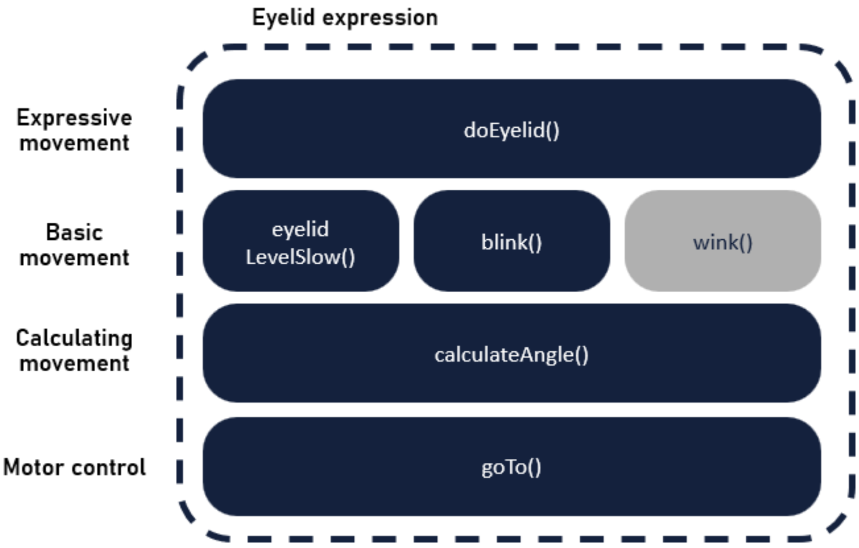


Figure 5.8: Hierarchical structure of the eyelids expression with the top level being Expressive movement and the bottom level being motor control, with the blue (implemented) and grey (not implemented) squares showing the functions on that tier

`eyeLidLevelSlow (int level, int speed, int accel)`

The `eyeLidLevelSlow()` function in Appendix C provides smooth and coordinated eyelid movement from one position to another. It uses the desired eyelid level, speed, and acceleration to perform the movement. When done, the function returns true. It utilizes the `calculateAngle` for the smooth motion.

`bool blink(int speed, int accel)`

The `blink()` function in Appendix C is a special case call for the `eyeLidLevelSlow()`. It calls this function with level 0. The reason for having a separate function is, firstly, for clarity and, secondly, to allow for different speeds for blinking and moving the eyelids. The `wink()` function would be similar, but only for one set of eyelids.

`void doEyelid(int level, int speed = MAXSPEED[state], int accel = MAXACCEL[state])`

The `doEyelid()` function in Appendix C coordinates the eyelid movement and blink routine. Switching between not blinking and blinking using random intervals. It allows for different speeds and accelerations, but as a default, it uses the maximum values. It does this by maintaining a global `blinkFlag`, which toggles between two behaviors:

1. **Normal eyelid movement** (when `blinkFlag == 0`): The eyelids are moved smoothly to the given `level` using the `eyeLidLevelSlow()` function. This function returns `true` once the desired position is reached. At this point, a random blink interval is generated between `MINBLINKINTERVAL` and `MAXBLINKINTERVAL`, and a timer is initialized. The `blinkFlag` is then set to `true`.

2. **Blinking** (when `blinkFlag == 1`): The `blink()` function is executed to temporarily close and reopen the eyelids at the given speed and acceleration. Once the blink is complete, the timer is reset, and `blinkFlag` is cleared, resuming normal eyelid positioning.

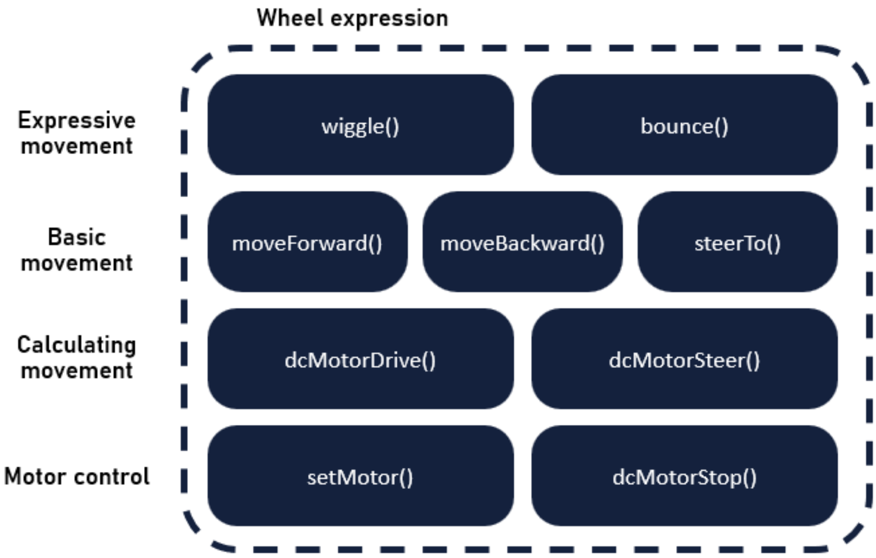


Figure 5.9: Hierarchical structure of the wheels expression with the top level being Expressive movement and the bottom level being motor control, with the blue (implemented) and grey (not implemented) squares showing the functions on that tier

5.2.7 Wheels

The wheel hierarchy is distinct from the other three with its own lower levels, as shown in Figure 5.9. Although the robot in this project has its locomotion system, it could also be attached to a different robot. This would let the other robot take control of the motor control or calculate movement tiers. However, for the sake of completeness, they are mentioned below.

`void setMotor()` and `void dcMotorStop()`

Both functions in Appendix C are simple. `setMotor()` sets the direction for the wheels on the direction pins and the throttle using PWM signals. The values it uses are the global variables `directionLeft`, `directionRight`, `throttleLeft`, and `throttleRight`. The `dcMotorStop()` function sets these values to zero and calls the `setMotor()` function.

`void dcMotorDrive(bool tempdirection, int throttle, int steering)`

The `dcMotorDrive()` function, as in Appendix C, takes the desired throttle, direction, and steering amount and maps that to the motors. When steering to the left, it scales the throttle of the left wheels down by the steering amount so that the robot drives in that direction, as shown on the right in Figure 5.10. Depending on its direction, it maps the throttle of the wheels to move backward or forward.

`void dcMotorSteer(int steeringInPlace)`

The `dcMotorSteer()` function, as in Appendix C, uses a steering value to steer in place. The sign determines the direction and the magnitude of the turning speed. It turns the motors of either side in the opposite direction with the same magnitude as in Figure 5.10 on the left. This results in steering in place.

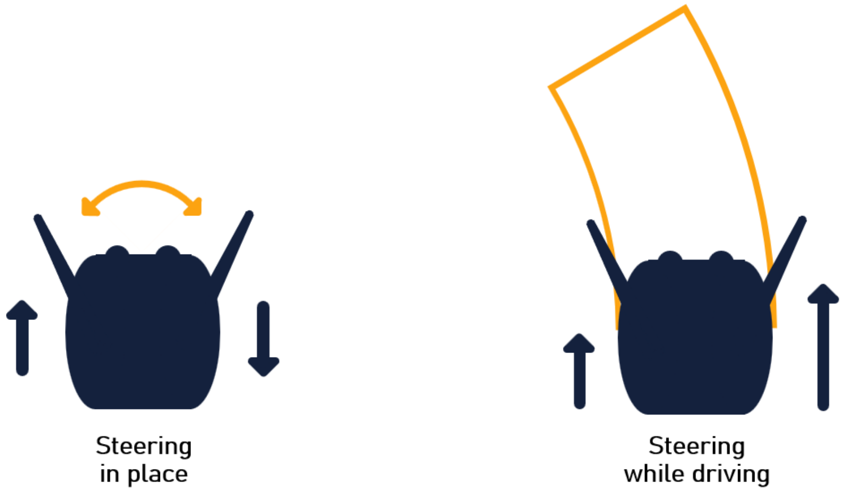


Figure 5.10: Two steering methods where the blue arrows show driving direction and throttle, with steering in place on the left and steering while driving on the right

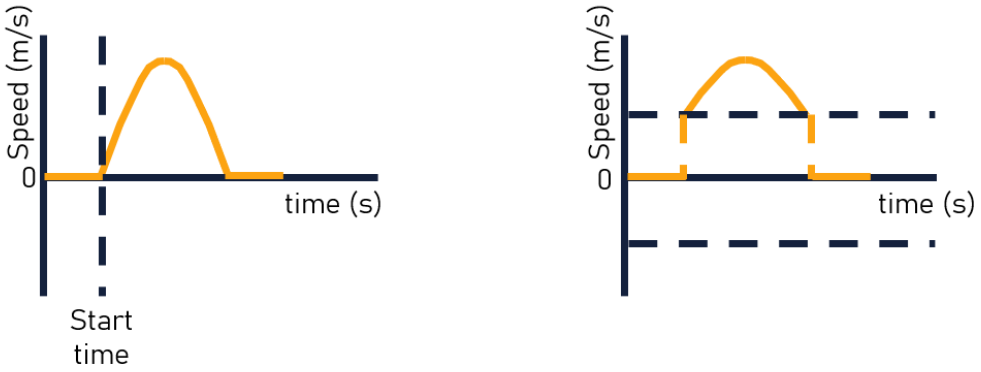


Figure 5.11: Half of the period of a sine wave as velocity for a forward motion on the left and the adjusted version that removes the deadband on the right

```
bool moveForward(unsigned long periodMs, unsigned long startTime, float amplitude)
```

The `moveForward()` function, as in Appendix C, uses the first half period of a sine wave for the throttle variable to drive a certain amount of time. Using a sine wave allows for speed up and slow down during movement instead of a jagged movement. However, because DC motors have a deadband, they cannot overcome the friction in the system, and this needs to be accounted for. Figure 5.11 shows how the deadband is removed to provide a smoother motion and prevent the high-frequency noise the motors make when not enough power is supplied. The deadband and amplitude can be determined in the integration stage, as it depends on the weight and friction in the design.

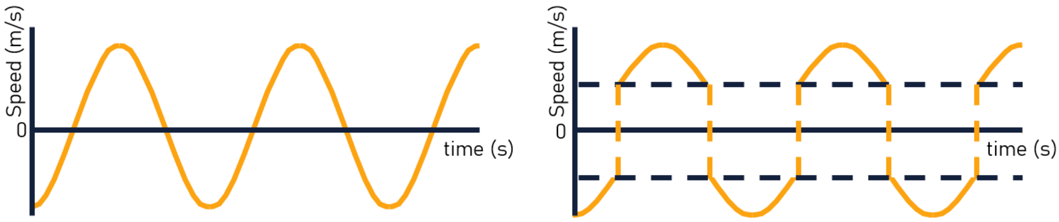


Figure 5.12: Bouncing motion on the left and the adjusted version that removes the deadband on the right

`bool movebackward(unsigned long periodMs, unsigned long startTime, float amplitude)` and `float steerTo(unsigned long periodMs, unsigned long startTime, float amplitude)`

These functions, as in Appendix C, are similar to `moveForward()`. The `movebackward()` function reverses the direction to drive backward. However, due to the motors' different capabilities in either direction, the amplitudes should be altered accordingly. The `steerTo()` function does the same but uses the `dcMotorSteer()` to perform its movement in place.

`float wiggle(unsigned long periodMs, unsigned long startTime, float amplitude)` and `void bounce(unsigned long periodMs, unsigned long startTime, float amplitude)`

These functions are very similar. The bounce function performs the `moveForward()` and `movebackward()` one after another. Providing a playful rocking motion. This results in the graph as shown in Figure 5.12. Similarly, the wiggle function also shows this graph but inputs it into the `dcMotorSteer()` function, showing a shaking motion.

5.2.8 Expressive Output

The behavioral layer on top then controls this expressive layer. This is the layer with the state machine. The state machine is implemented in `void doState ()` in Appendix C and uses the `goToState()` to handle state transitions, as discussed earlier. In Table 5.13, the expressive movements are shown. Within some states, different actions are performed upon entering.

A final overview can be found in Figure 5.14. It shows the four expressive layers that were discussed earlier. Furthermore, it shows the layers controlling those expressions and the layer on top of that – the behavior.

5.3 Motors

Two types of motors are required for the robot. The first type of motor is angle-controlled. A servo motor is a straightforward option, and because the robot performs restricted movements, it can be a non-continuous servo motor. For driving, DC motors were selected. With easy control and affordability, it is also a straightforward decision. Both motors can be found in Appendix A.

5.3.1 Servo Motors

The robot uses two types of servos. These are the Miuzei mf90 and Miuzei Servo Motor MS24. The first is a smaller, low-torque servo with a 0°-180° range. This is used for the eyes, eyelids, and the horizontal movement of the antennae. The latter is a larger, high-torque

State	Functions on Entry	Functions While in State

`Relaxed`	`steerDirection()`	`doEyelid()`
		`doRandomAntennae()`
		`eyeFollow()`
		`dcMotorStop()`

`Playful`	None	`doEyelid()`
		`doRandomAntennae()`
		`eyeFollow()`
		`bounce()`

`Alert/interested`	`moveForward()`	`doEyelid()`
		`doRandomAntennae()`
		`eyeFollow()`
		`dcMotorStop()`

`Stressed`	`moveBackward()`	`doEyelid()`
		`doRandomAntennae()`
		`eyeFollow()`
		`dcMotorStop()`

`Fearful`	`moveBackward()`	`doEyelid()`
		`shiver()`
		`eyeFollow()`

`Defensive`	`moveForward()`	`doEyelid()`
		`doRandomAntennae()`
		`eyeFollow()`

`Sleep`	`eyeLidLevel()`	`doRandomAntennae()`
		`eyeFollow()`
		`dcMotorStop()`

`Go to Sleep`	`eyeLidLevelSlow()`	`doRandomAntennae()`
		`doRandomEye()`
		`dcMotorStop()`

`Wake up`	`eyeLidLevelSlow()`	`doRandomAntennae()`
		`doRandomEye()`
		`dcMotorStop()`

`Wake up fast`	`eyeLidLevelSlow()`
	`eyeScanRoom()`
		`doRandomAntennae()`
		`dcMotorStop()`

Table 5.13: Function calls during state entry and state execution

servo with a 0°-270° range. These are used for the vertical movement of the antennae, as they need to be able to move them quickly up and down.

5.3.2 DC Motors

The DC motors are the Nidec MG16B-060-AA-00. They were selected by another student for another project. This also proves that the rest of the robot can function on top of another robot.

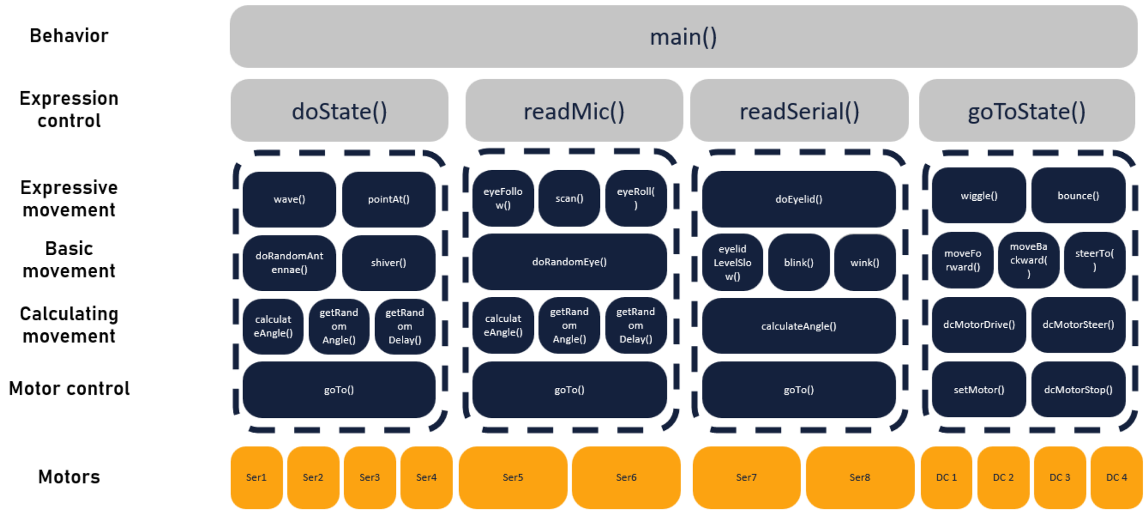


Figure 5.14: Complete functional hierarchy of the robot with all functions and all levels of behavior

5.4 Motor Drivers

Two motor drivers are required to handle eight servos and four DC motors. Both motor drivers can be found in Appendix A.

5.4.1 Servo Motor Driver

The selected motor driver for the servos is the Adafruit motor shield v2, which utilizes the PCA9685 for servo control. This chip can create 16 independent PWM signals. Although it could theoretically also handle the DC motors, the power requirements were too high.

A helper function handles the conversion from angle to PWM signal width, `servoDegree()`, which translates the given angle into a duration in microseconds. This value is then scaled to match the resolution of the PCA9685, which operates on a 12-bit timer at 50 Hz. The PWM register value is computed using the formula:

$$\texttt{PWM}_{value} = \frac{\texttt{servoDegree(angle)} * 1000.0}{\texttt{MICROSPERBIT}} \tag{5.1}$$

Where `MICROSPERBIT` is a constant defining the number of microseconds represented by one bit of PWM resolution (typically $4.88 \mu s$ for 12-bit resolution at 50 Hz). This result is passed to the `setPWM()` function of the Adafruit PWM driver library, which updates the corresponding PWM channel of the servo.

An alternative method using `writeMicroseconds()` is included in the code as a comment. **Furthermore, the frequency of the chip can be slightly off and should be checked using an oscilloscope.**

5.4.2 DC Motor Driver

The DC motors were connected to the HG7881, a widely used h-bridge module. It is based on the L9110 IC, which can handle 800mA per channel. The motors are rated for less than 400mA and could be coupled together. This meant that one channel controlled both motors on the left and one channel on both motors on the right.

The motor driver is directly controlled using two PWM pins on the Arduino and two regular pins for the direction. The board is powered separately, which will be discussed in the next chapter.

Chapter 6

Integration

Now that all four subsystems have been discussed, the integration can take place. This chapter discusses the integration of all four subsystems. It starts with some additional software and hardware challenges and is followed by the overall design with analysis and specifications overviews.

6.1 Software

Running all the code in Appendices B, C, and D is not sufficient for the robot. A few things need to be considered on the Arduino and RP. Furthermore, a small discussion is done on ROS 2 and its potential for this robot.

6.1.1 Arduino

The Arduino that is mentioned in this report is the Arduino WiFi rev 2.0, as shown in Appendix A. For programming the Arduino, the PlatformIO extension (version 6.1.18) on Visual Studio Code was used. This is a microcontroller board and library manager. The libraries that were used are shown in Appendix D.

Firmware

To use the controller, a firmware reflash of the Arduino is needed. By default, the NINA-W102 WiFi and Bluetooth module is configured only to enable the WiFi. **A reflash is required to allow for Bluetooth.** For the reflash version, 4.0 was used [45]. It is essential to use this version as later versions are incompatible with the controller.

Bluetooth

Bluetooth connection handling is not something a microcontroller is designed to do and takes up quite some processing power. All the Bluetooth and controller handling functions are shown in Appendix C. They are slightly modified example codes from the Bluepad32 library. When the controller is not used, it is essential to disable these functions to increase performance.

Modes

To be able to use the robot on different occasions and allow for better testing setups, three different modes have been created:

- `DEBUGFLAG`: Setting this to true enables debug mode. Debug mode prints many helpful messages and information to the terminal. These include the state, angles of the servos, DC motor throttles, and controller connections. Because the Arduino and RP are connected via serial, debug mode cannot be turned on while they are connected. This is because the serial connection is in use. Besides this, it is important to note that the debug mode is for the Arduino only; however, the RP can be debugged separately by connecting a screen to it.

- `DCMOTORUSEFLAG`: Setting this to true enables the DC motors, and conversely, setting it to false turns them off. This is especially useful when testing on an elevated level (a table), in a tight space, or when testing the servos.

- `CONTROLLERUSEFLAG`: Setting this to true enables controller-based control and disables the state machine transitions. This is useful for isolated testing and evaluation of the robot. When in operation, it should be turned off to improve performance.

6.1.2 Raspberry Pi

For the tests in this project, the Raspberry Pi 5 with 4GB RAM was used. However, the 16GB RAM version is recommended as in Appendix A. With the better RP, emotion recognition could potentially be added, and higher resolutions and, thus, higher performance can be achieved.

To use and program a RP, a screen, a mouse, and a keyboard are required. This is not desired for the robot to operate. The robot should function when the Pi boots, and it should be able to handle sudden power loss due to battery removal. These are two things that are not incorporated as default.

Run on boot

To run the RP on boot, several methods are possible. Four widely used methods are rc.local, autostart, systemd, and crontab. Of these, rc.local and crontab block the GUI, which means making changes afterward is difficult, and autostart requires a desktop login. **The added benefit of systemd is that it allows for retries on errors and allows the GUI to load, but it does not use it.** This means the system is fully operational, and later alterations can be made.

Using Systemd requires writing and enabling a service file. In the service file, the code path, restart conditions, and timers are set. For the robot, a restart on failure method is selected to prevent crashes when the serial connection is broken.

Power Off Safely

A RP should be powered down correctly to prevent SD card corruption. This happens when the power is lost mid-write. A hardware solution can be implemented to solve this. This would set a predetermined pin low that asks the RP to shut itself off. This has several issues, but the most pressing is that this is a software shutdown. For some applications, this might not be an issue, but a moving robot should always shut down safely when power is lost.

Another solution is to make the SD card read-only. This means no mid-write power loss can happen, as all writing is prohibited. Not only is this inconvenient as it cannot store any data, but it also removes the possibility of modifying the robot without getting a second SD card and reflashing it.

The third option, which was chosen, is an overlay system. This method makes all the essential files read-only and overlays a second writeable layer that is not essential for any functions. As a bonus, on the RP 5, this is a standard setting that can be turned on.

Debugging

Debugging the RP can be done by connecting the screen, mouse, and keyboard. This is possible because the systemd was used to perform the start-on-boot action. It can be tested while it is connected to the Arduino, which allows for testing with serial communication in place.

6.1.3 ROS 2

In future work, the software could be made ROS 2 (Robotic Operating System) compatible. ROS 2 is a modular node-based architecture with life cycle management, enabling robust and maintainable systems. This modular structure is ideal for a symbiotic robot that requires cross-platform compatibility. Furthermore, ROS 2 is under active development by Open Robotics and the community, with planned long-term support releases, making it future-proof. Due to time constraints, this project has not yet made ROS 2 compatible.

6.2 Hardware

Integrating the hardware requires a clear distinction between the host and the symbiote. Mainly because, for this project, a separate host is created to show the full capabilities of the symbiote. Although discussed more in-depth in the technical design section in [7], the power source, material choice, and tools are briefly mentioned.

6.2.1 Symbiote and Host Distinction

A symbiotic relationship is one where both the host and symbiote profit from the collaboration. For this project, the symbiote provides expressive capabilities, and the host provides movement and power, as shown in Figure 6.1. Together, they are capable of doing more than the individuals can on their own.

The host that is used in this project was constructed from the frame and wheels of the Mirte robot [18]. **This was done to show that the symbiote could be attached to virtually any host as long as the physical (i.e., weight and size) and power requirements were met.** The Mirte robot provides a wooden base, four wheels, and four motors. To this base, a 100W power bank was connected (components in Appendix A). Together, these simulated the minimum requirements for the host. In this case, the symbiote even had to take control as the host had no processing unit.

The symbiote consists of all other parts. These are the electronics, actuators, cables, structure, and housing, all placed on a flat mountable surface. The symbiote takes care of the sensing, interpretation, decision-making, and servo actuation. Using PWM signals, it communicates the expressive wheel movements. This shows typical symbiote behavior as it requires the host for movement, whereas the host would be unable to express itself without it.

6.2.2 Power Source

As mentioned above, the power source used for this project is a power bank. **More specifically, the INIU 100W 25000mAh power bank.** A power bank was chosen for several reasons.

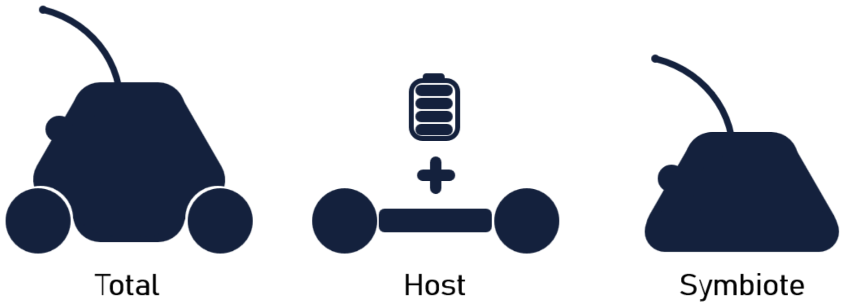


Figure 6.1: Difference between the host (locomotion and power source) and the symbiote

First of all, safety: a CE-certified power bank is safe for usage throughout this project. Secondly, it is easy to recharge using universal chargers. Lastly, it is widely available and easy to assemble. This 100W power bank can easily handle the power requirements, as shown in the following power section.

6.2.3 Materials

All structural and cosmetic parts are 3D printed. Although not as strong as CNC-ed metal or molded plastics, the method was chosen for its effectiveness in rapid prototyping and low-scale manufacturability. Furthermore, it allows for modification, alterations, and improvements to the design in the following projects and uses widely available PLA. **This means everyone can make one at home, given they have a 3D printer.**

6.2.4 Tools and Building

Similar to the argument about choosing 3D printing, no custom-printed circuit boards or specific tools are used. Building the robot requires only a basic screwdriver set and a soldering iron. All the soldering jobs are connecting cables, which could also be done using screw cable connectors. The inserts used to create screw holes in the plastic can be easily inserted using a soldering iron or could be replaced by printed screw holes, removing the need for a soldering iron. All tools can also be found in Appendix A.

6.3 Design

The overall design of the robot is split into four parts. These are the physical design, electronics design, signals and data design, and Interaction design. The design choices for the physical and interaction design are discussed more in-depth in section ideation and technical design in [7].

6.3.1 Physical Design

The final physical design after all design considerations is shown in Figure 6.2. The left shows the render of the symbiote on top of the host, and the right shows a photograph of the end product. All appendices can be seen as part of the symbiote body. The wheels on either side belong to the host. The physical structure is discussed in the technical design section in [7], and the design choices for the appendices are discussed in the ideation and exploration sections.

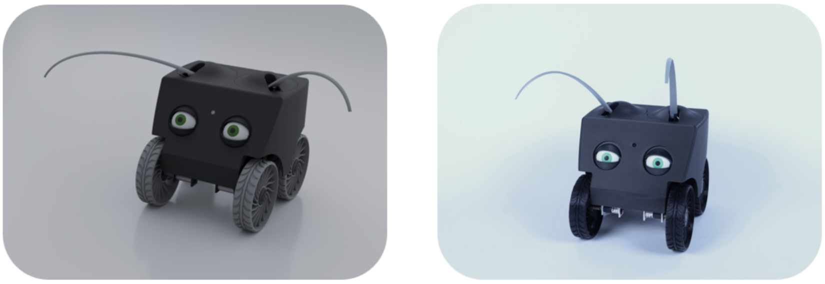


Figure 6.2: Render and photo of the symbiote on top of the host

Cost

The BoM can be found in Appendix A. This shows the cost, quantity, retailer, and purchasing link of all components used. Using the BoM, a purchase list can be created by entering the number of robots that are to be made.

The cost of one robot depends not only on the purchased components but also on the number of robots. Making more robots decreases the unit price, even without changing to production methods that scale. This is due to delivery costs, packaging quantities, one-time purchase items, and filament remainders. Using this, a calculation is made for the prices of one robot compared to the price per robot in bulk (incl. Dutch taxes of 21%). Table 6.3.1 shows the cost of one robot compared to the bulk price per robot for the number of robots going to infinity. It also splits up the cost for the symbiote and host and **shows that the symbiote benefits most from bulk purchases, making it perfectly suitable for swarming robots.**

Cost Metric	**Total**	**Symbiote**	**Host**
Price one robot (euros)	€ 967.22	€ 654.41	€ 312.81
Bulk Price/Robot (euros)	€ 658.15	€ 345.35	€ 312.80

Table 6.3: Cost of a single robot and the bulk price

Weight

The weight of the symbiote is important because the host should be able to carry it around. Next to this, for a larger number of symbiotes, the print time required is also important. If only a few printers are available, this could hamper production. No print optimizations were performed. However, analysis on print duration, filament usage, and filament loss was performed and shown in Table 6.4. **For the symbiote, it shows 42 hours of print time, 1.5 kg of filament, 0.2 kg of filament loss, and a total weight of 1.5 kg.** Filament weights and print times are retrieved from the Bambulab slicer, whereas the final weights are obtained through scale measurements.

6.3.2 Electronics Design

Figure 6.5 shows the wiring diagram of the robot. It shows several groups with different functionalities. In the top right is the power source or the host providing power. In the case of this project, it is the power bank using a USB-C trigger device that receives 20V at the output. The left top corner shows the power distribution consisting of three 5V/5A buck

Fabrication Metric	Total	Symbiote	Host
Print time (h)	69.6	42.2	27.5
Filament usage total (kg)	2.21	1.48	0.73
Filament usage robot (kg)	1.92	1.29	0.64
Filament loss (kg)	0.29	0.19	0.09
Number of prints	53	29	24
Weight (g)	2638	1508	1130

Table 6.4: Robot fabrication time, filament use, filament loss, and total weight for both symbiote and host

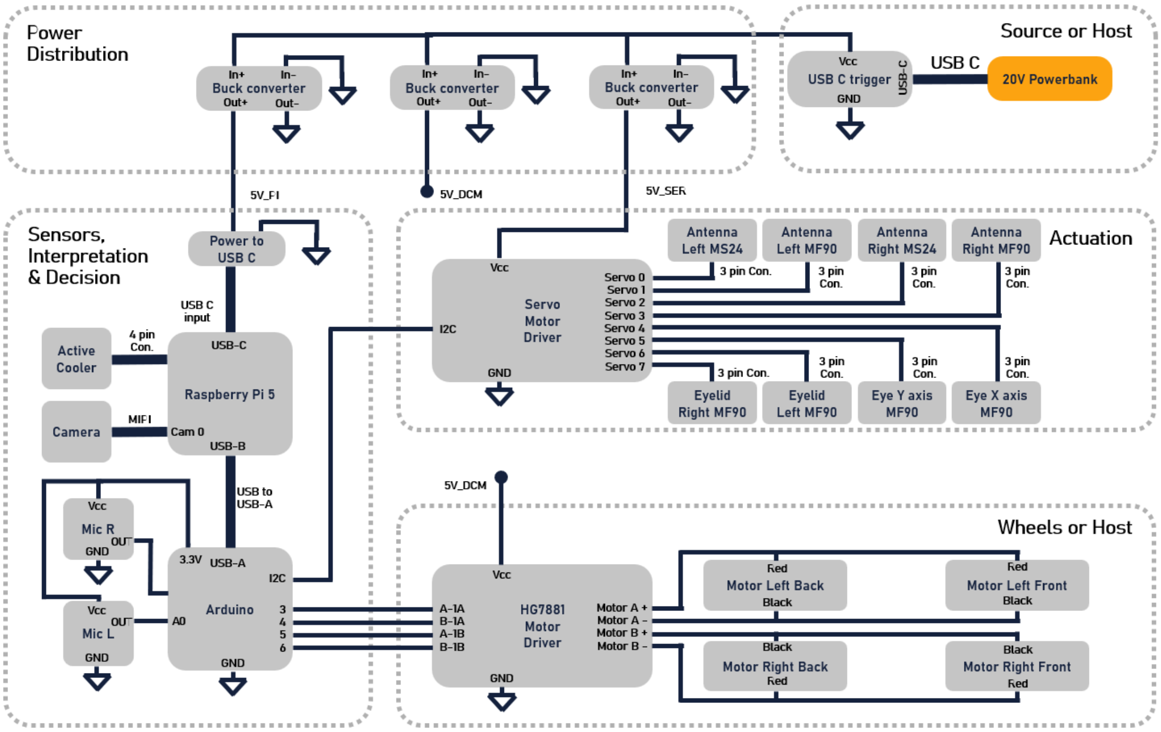


Figure 6.5: Schematic showing how to connect all electronics components categorized in several groups

converters, separating the RP and actuation systems. This is done because the RP requires a stable voltage and turns off when the voltage drops below 4.9V. The left bottom corner shows the system with the RP, Arduino, and sensors. This is connected via I2C to the servo actuators and via a PWM connection to the host. Both these systems have their buck converter.

Power Flow

To analyze the power flow, two power flow diagrams are made. Figure 6.6 shows the power flow through the robot with both the symbiote and the host. The numbers next to the nominal voltage and the maximum rated current. In this diagram, the sum of the maximum ratings at every level should not exceed that of the level on its left. This way, no maximum rating is ever exceeded. A similar schematic of only the symbiote is in Figure 6.7. Here, the host provides the power and takes care of the locomotive subsystem.

After describing the power flow and confirming electronics component safety, the power

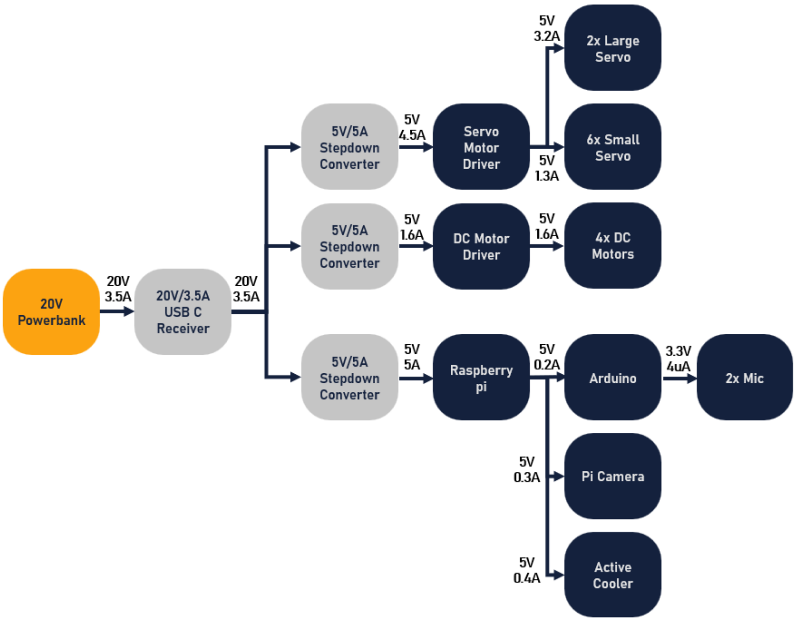


Figure 6.6: Power diagram for the symbiote and host together with numbers next to the arrows showing the nominal voltage and maximum ampere rating

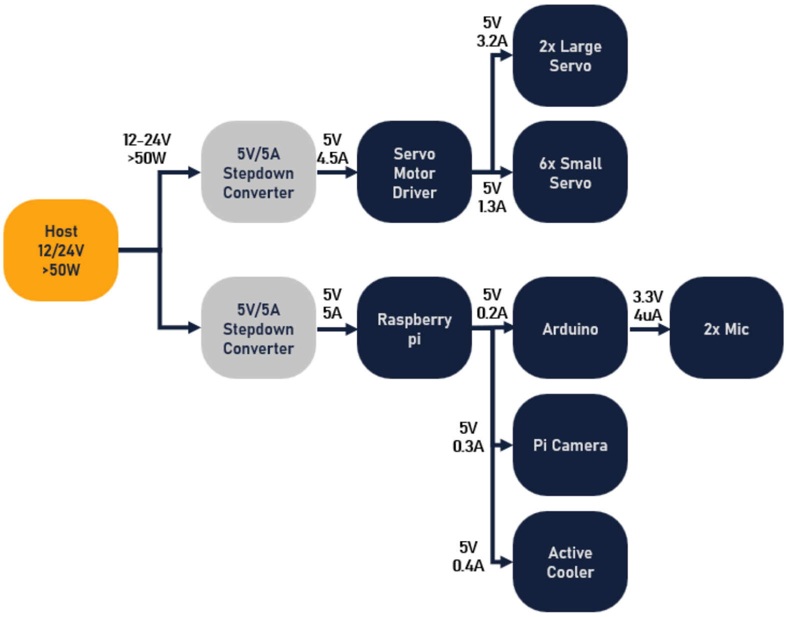


Figure 6.7: Power diagram for the symbiote with numbers next to the arrows showing the nominal voltage and maximum ampere rating

consumption can be tested. A USB-C power measurement tool was used for this measurement. This is inserted between the power bank and the buck converters. It measures the power over time and provides an average. The results were performed on the whole robot and on the robot where the DC motor driver was disconnected (Symbiote). Table 6.8 shows the measured power consumption per state. It is important to mention that, except for the happy state, all other states do not use the DC motors during the state but only upon entering it. This means the measurements do not consider this. This also explains the high power usage in the happy state compared to the others.

Using the average power consumption of 12.34 Watts and assuming every state had an equal presence, the battery life can be calculated. Let us use Eq. 6.1 with t_{batt} as the battery lifetime, $E_{eff} = 100Wh$ the effective capacity of the battery, and P_{avg} the average power. **This gives a battery life of 8.1 hours.**

$$t_{batt} = \frac{E_{eff}}{P_{avg}} \tag{6.1}$$

Power usage	**Total (W)**	**Symbiote (W)**	**Host (W)**
Sleep	10.9	10.7	0.2
Relaxed	11.5	11.2	0.3
Playful	16.7	11.9	4.8
Alert/Interested	11.3	11.2	0.1
Stressed	11.7	11.6	0.1
Fearful	12.5	12.3	0.2
Defensive	11.8	11.6	0.2
Average	12.34	11.50	0.84

Table 6.8: Power usage in different states for the robot, symbiote, and host

6.3.3 Signal and Data Design

The robot has three processors: the IMX708, ARM Cortex-A76, and Atmega4809 that belong to the camera, RP, and Arduino, respectively. All these processors perform actions in a loop. The number of loops they can perform in one second is called the Frames per Second (FPS). Each processor's average and minimum FPS is calculated, with the minimum being the worst-case scenario.

Processor Performance

The measurements for the IMX708 were performed in Chapter 3 with the results in Table 3.3. The FPS decreases with a higher resolution. However, as will be clear, this is not the bottleneck of the system. Furthermore, the resolution also has an impact on the next processor.

The ARM Cortex-A76 in the RP is used for body and face recognition and uses the camera images as input. With the methods chosen, the resolution is the leading indicator for FPS performance. Table 6.9 shows the loop times for different resolutions and the corresponding FPS, which is $1/t_{loop}$ with t_{loop} as the loop time. All measurements were performed over a thousand loops. **For the symbiote, the 1280x720 resolution was chosen.** This was done for several reasons. Firstly, with this resolution, all distance performance metrics are met. Secondly, it is the highest resolution where the worst-case scenario is still within the requirements. Lastly, as processors get hot – a thing they tend to do during operation – the performance decreases.

Resolution	Mean Time (s)	Max Time (s)	Min Time (s)	FPS	min FPS
1280x720	0.0705	0.0957	0.0650	14.19	10.45
1920x1080	0.0768	0.1118	0.0676	13.03	8.94
2560x1440	0.0927	0.1285	0.0836	10.79	7.78
3840x2160	0.1703	0.2695	0.1216	5.87	3.71

Table 6.9: Minimum, mean, and maximum loop time of the ARM Cortex-A76 in the RP for different resolutions with the corresponding FPS

Lastly, the Atmega4809 processor in the Arduino. For the Arduino, the main performance indicator is the state. The performance per state is shown in Table 6.10. Again, the measurements were performed over a thousand loops. The table clearly shows the playful state as the bottleneck when looking at the average FPS. **This is because the PWM generation in the Arduino takes approximately 50 ms, whereas the rest of the loop takes about 10 ms.** This results in the longest loop time for the happy state because it continuously uses the DC motors.

State	Mean Time (s)	Max Time (s)	Min Time (s)	FPS	Min FPS
Sleep	0.0071	0.0078	0.0058	140.85	128.21
Relaxed	0.0072	0.0081	0.0061	138.89	123.46
Playful	0.0602	0.0632	0.0553	16.61	15.87
Alert/Interested	0.0074	0.0638	0.0067	135.14	15.67
Stressed	0.0074	0.0631	0.0069	135.14	15.85
Fearful	0.0076	0.0628	0.0069	131.58	15.93
Defensive	0.0076	0.0632	0.0068	131.58	15.85

Table 6.10: Minimum, mean, and maximum loop time of the Atmega4809 processor in the Arduino for different states with the corresponding FPS

Data Flow

Now that all the FPS values are known for all the processors, the bottleneck of the total system can be determined. Figure 6.11 shows the flow of data through the robot. All three processors have their average and minimum FPS shown. The RP reads the camera values. The Arduino reads the serial port and microphone values and sends the control signals to the motor drivers. In this schematic, the values for the chosen 1280x720 resolution and the worst-case average and minimum of all states are used. This clearly shows that the RP is the bottleneck, both in the average case and the worst case.

To calculate the longest data path for the symbiote, a few things are important. In the calculations for the RP, the time it takes to get the images from the camera is included. This means that the camera's loop time is not added to the longest path. For the RP, the values as in Figure 6.11 are used. Lastly, for the Arduino, the symbiote only has an average FPS performance of around 140, as for the relaxed and sleeping state in Table 6.10. The worst-case performance is 123 FPS. Using Eqs. 6.2 and 6.3 the average and the minimum FPS of the symbiote can be calculated respectively. With $FPS_{PI,avg}$ and $FPS_{PI,min}$ being the average

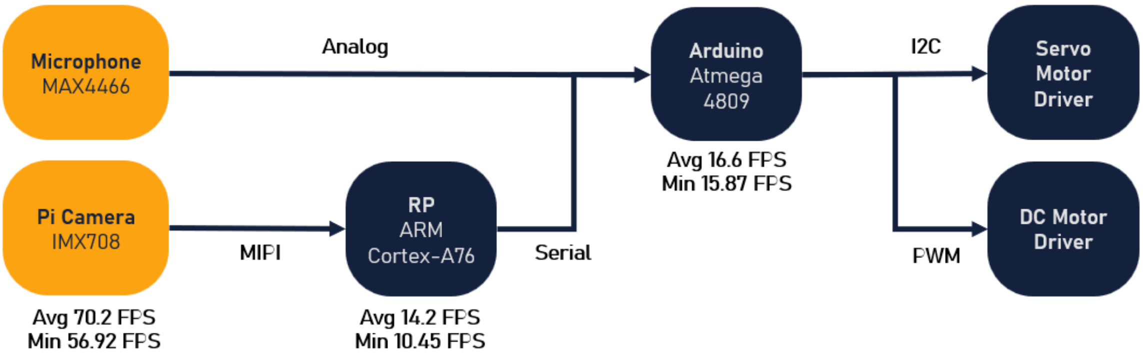


Figure 6.11: Data flow through the data processing components of the robot, with the numbers indicating the average FPS and minimum FPS at each stage, showing that the RP implementation is the bottleneck

and minimum for the RP and $FPS_{Arduino,avg}$ and $FPS_{Arduino,min}$ are the average and minimum for the Arduino. The reason that in Eq. 6.2, a factor of two is added to the Arduino FPS is because, on average, it is expected that the sampling of the serial port takes place after half the loop time. **Filling in the numbers gives an average of $FPS_{tot,avg} = 13.51$ and a worst-case scenario of $FPS_{tot,min} = 9.71$.**

$$FPS_{tot,avg} = \frac{1}{\frac{1}{FPS_{PI,avg}} + \frac{1}{2*FPS_{Arduino,avg}}} \tag{6.2}$$

$$FPS_{tot,min} = \frac{1}{\frac{1}{FPS_{PI,min}} + \frac{1}{FPS_{Arduino,min}}} \tag{6.3}$$

6.3.4 Interaction Design

The interaction design for this report is split up into the perception, movement, and emotional parts. With perception being visual and auditory.

Perception

Table 6.12 shows the visual performance metrics for all three detection types and the final design of the symbiote. The distances were measured with the robot on the floor and a caucasian male of average Dutch height (1.80 meters) standing up straight. This would be a realistic scenario for the robot in its use. The reason the minimum distance for face detection is higher than that of body detection is that the face is not on the screen, and body detection also works for the lower half of the body. For the number of detections, multiple people were put within the distance range, and a minimum of three people were tested. Emotion detection was only tested on the closest detected face.

Table 6.13 shows the microphone specifications, as used in the symbiote. It has two microphones, allowing for differences in time of arrival calculations. However, it must be stated that the current sampling frequency of the Arduino is insufficient for this purpose, and performing this calculation requires an extra dedicated processor. Furthermore, the full range of human speech is within the detectable band. The trigger value for loud sounds is set around 70 dB, which is above a loud conversation close to the robot. **This means the robot will not trigger due to regular conversations but will trigger from a loud clap or shout.**

Recognition	Face	Body	Emotion	Symbiote
Distance max (m)	1.16	4.95	1.16	4.95
Distance min (m)	0.55	0.33	0.55	0.33
Number of detections	>3	>3	1	>3
FPS	88.1	15.4	11.64	13.51
Viewing angle (degrees)	60	60	60	60

Table 6.12: Performance metrics for the face, body, and emotion detection and the metrics for the symbiote (face and body detection)

Specifications	Value
Number of microphones	2
Trigger value (dB)	70
Frequency min (Hz)	20
Frequency max (Hz)	20000
Sampling frequency (Hz)	250

Table 6.13: Microphone specifications used for the symbiote

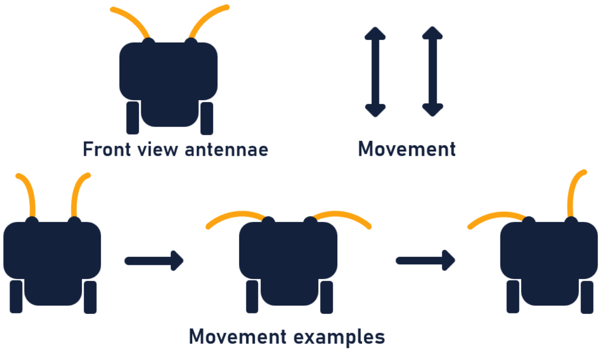


Figure 6.14: Possible movements with an example for the antennae (in orange) as seen from the front, being asynchronous up and down movement

Movement

The movements of the symbiote can be split into three. The first is the antennae. The antennae can move independently (asynchronously) from each other. The vertical movements, as seen from the front, are shown in Figure 6.14. It shows an example of possible positions of the antennae, where one can stand up straight while the other is down. This is very similar to how insects use their antennae. The vertical position of both antennae has a range of 90°. The top view shows the horizontal movement. Again, these movements are independent. This movement is shown schematically, as seen from above, in Figure 6.15. These movements have a range of 180°.

The eye and eyelid movements are synchronous. This means both eyes have the same angle, and so do the eyelids. However, the eyelids are not physically linked, meaning they could move asynchronously. The movements of the eyes with an example are shown in Figure 6.16 and for the eyelids in Figure 6.17.

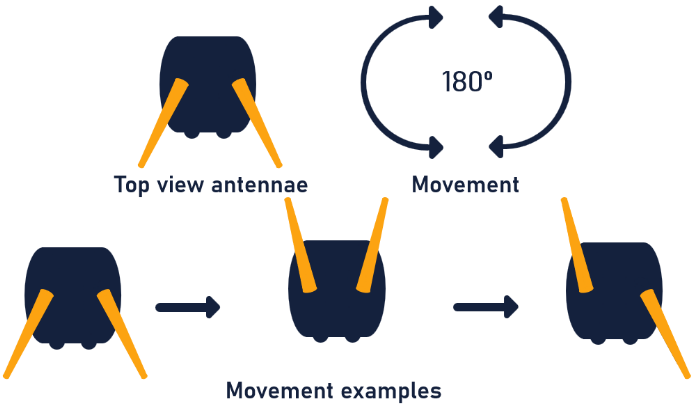


Figure 6.15: Possible movements with an example for the antennae (in orange) as seen from the top, being 180° asynchronous rotation

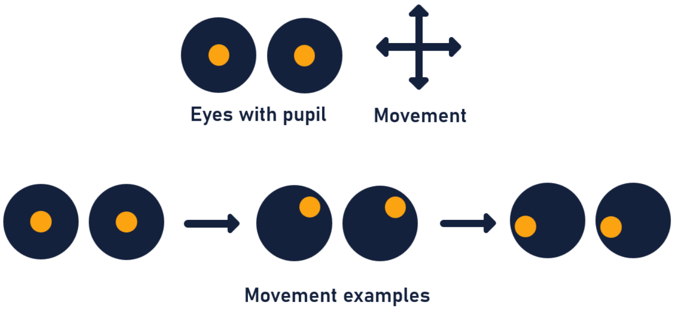


Figure 6.16: Possible movements with an example for the eyes with pupils (in orange) being up and down and left and right synchronously

A summarisation of these movements is shown in Table 6.18. It shows the degrees of freedom and the angle range per degree of freedom for all three appendices. These movements are then used to express the emotions.

Appendix	**Antennae**	**Eyes**	**Eyelids**
Degrees of freedom	2	2	1
Angle Range vertical (degrees)	90	60	60
Angle Range horizontal (degrees)	180	60	-

Table 6.18: Degrees of freedom and movement range for the three appendices of the symbiote

Emotions

Figure 6.19 shows renders of the six emotional states of the robot and the sleep state. To adhere to the remarks of Levillain in [46], the robot's performance movements in every emotional state. No state is stationary, not even the sleep state, where the symbiote ever so slightly

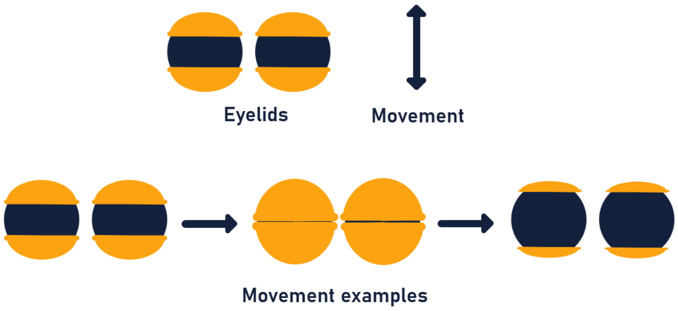


Figure 6.17: Possible movements with an example for the eyelids (in orange) being up and down synchronously

moves the antennae. Furthermore, in all other states, the robot blinks, looks at the closest person, and performs antennae movement to conform to the state.

6.4 FLIP: Friendly Logic-based Interactive Presence

FLIP is a friendly logic-based interactive presence, as shown in Figure 6.20. It is a robotic symbiote created for swarm robots to aid in sensing humans and expressing themselves towards them. It is specifically designed for use in the Cyber Zoo at DUT [17]. Furthermore, it aims to provide a novel educational framework for other students to test software, create hardware, and study interactions.

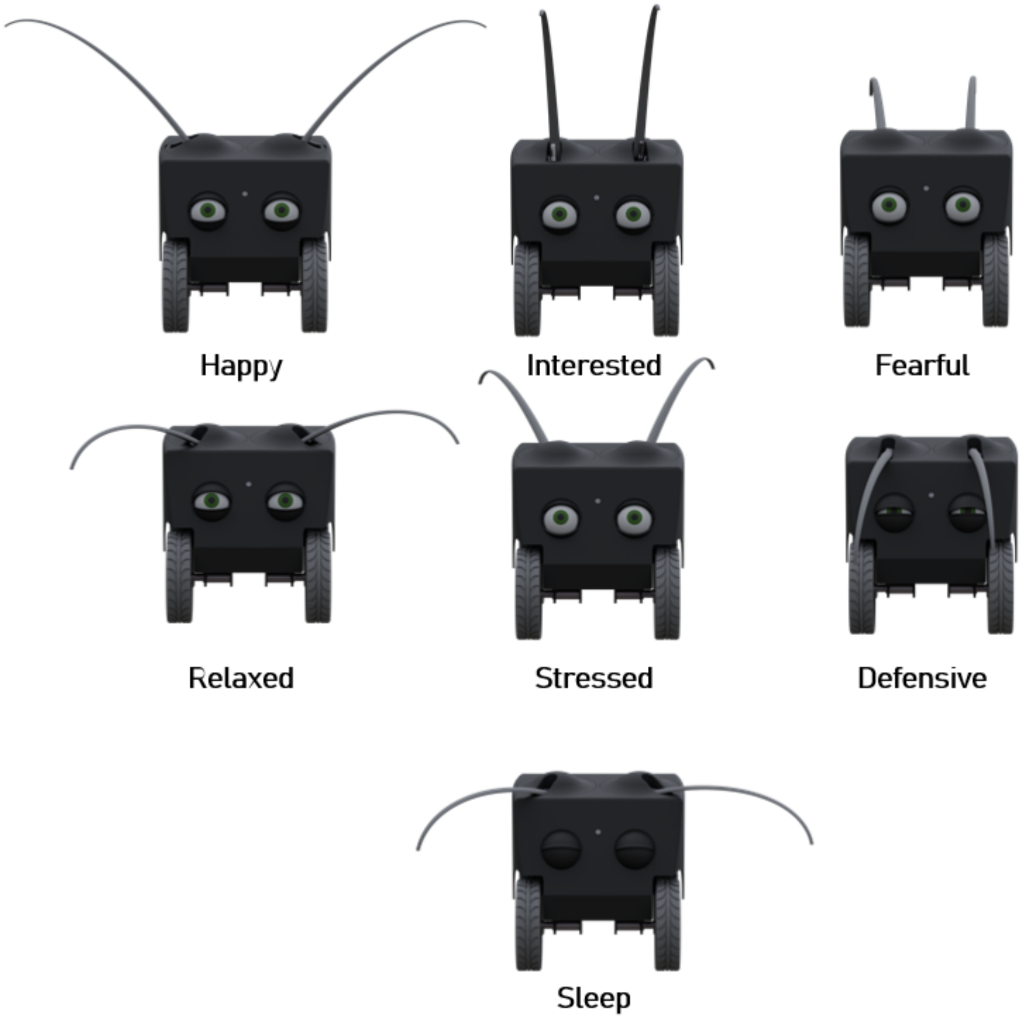


Figure 6.19: All six emotional states and the sleep state of the robot

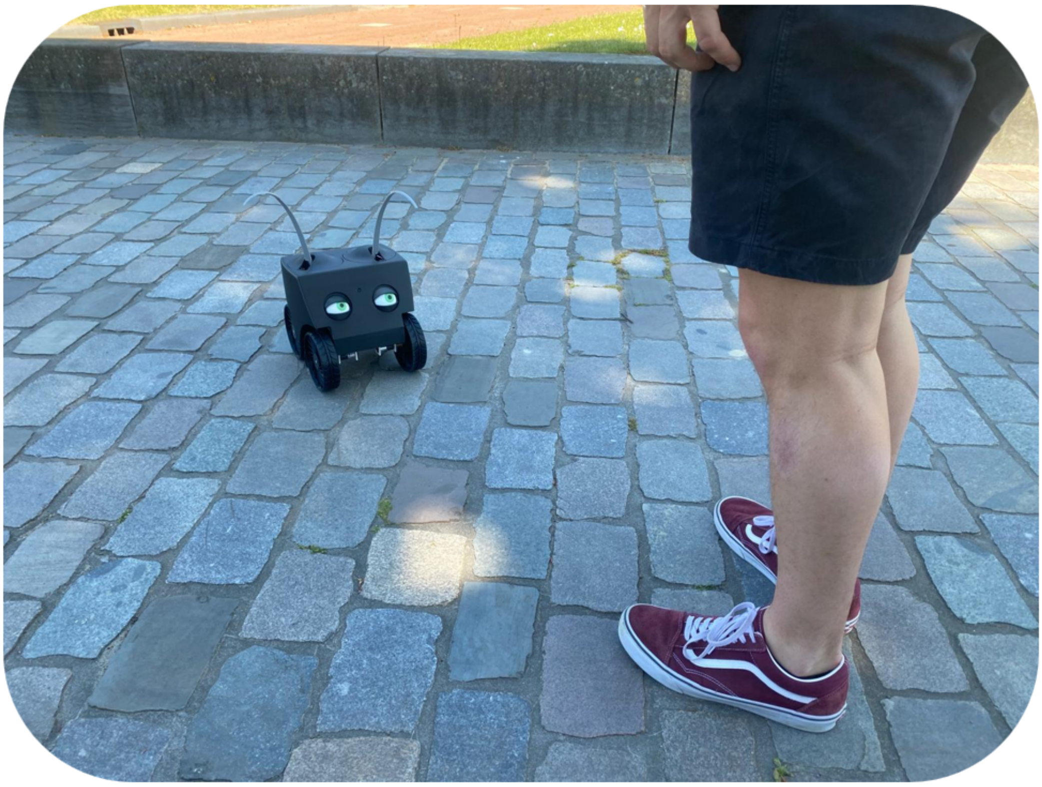


Figure 6.20: Photo of FLIP the robot on the DUT campus interacting with a person

Chapter 7

Evaluation

This chapter summarizes the completion of all the requirements. All requirements are either successfully met, partially met, or not met. All these answers are justified with references to the corresponding sections in brackets. These requirements, originating from the IPD report [7], are essential for the case study. After this, the research questions are answered and discussed. These research questions focused on the scientific gap, which will also be addressed in this chapter.

7.1 Must Have Requirements

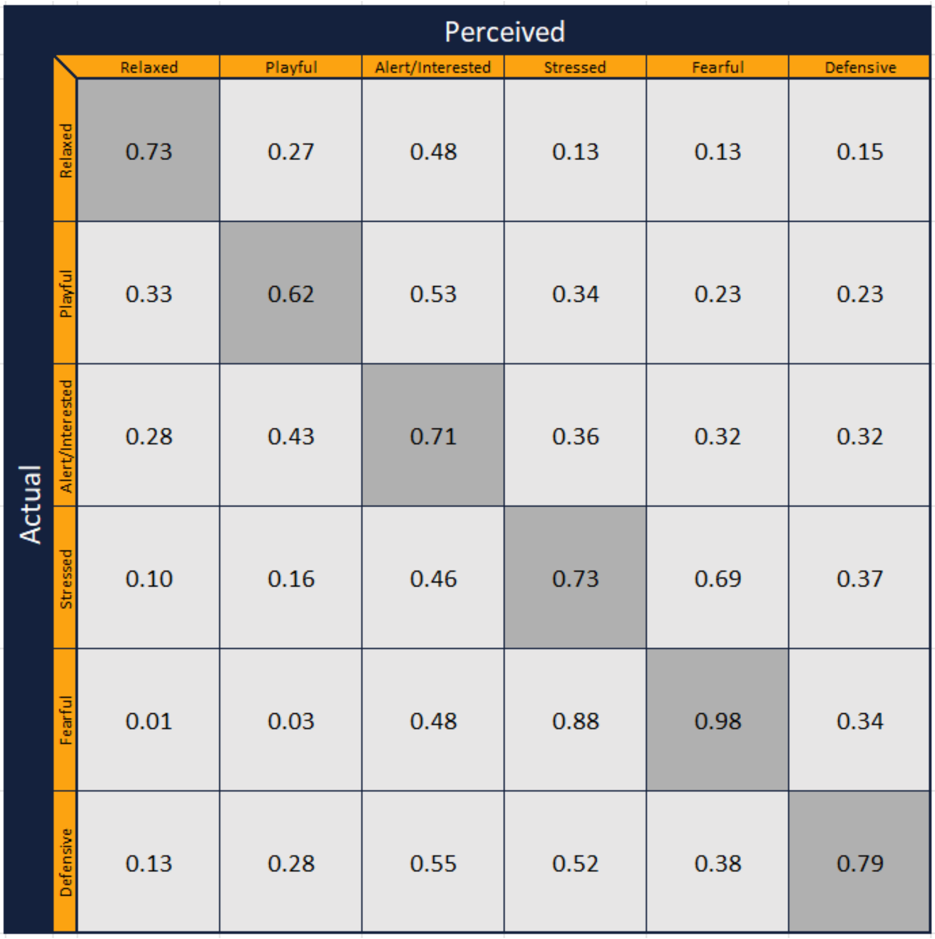
The must-have requirements are shown in Table 7.1. It shows the requirements as mentioned in Section 1.7, with a short description. It mentions if the requirements are fulfilled and also how this was established. This table shows a complete overview, but some requirements deserve more attention.

M.01 and M.09 through M.11 are all robot requirements. These are needed to perform the functionalities for the case study. Here, safety checks are critical as the robot will be close to humans and interact with them.

M.03 through M.7 are all on sensing and interpretation. These detection requirements were determined to be necessary for an intuitive interaction in the IPD report. All these requirements were tested up to the requirement limits and not further. This means the symbiote could detect more than three bodies and faces.

Requirement M.13 was one of the most limiting and challenging requirements in this design. The tested average fulfills the requirement with a 35% margin. However, the requirement formulation mentions that it should operate at least 10Hz. This means that it does not satisfy the requirement. However, in standard operation (over a longer time), it is not expected to go below 10Hz. Thus, it is partially fulfilled.

M.15 and M.16 are about the ability of people to detect the correct emotion. The test method, setup, results, and discussion are in [7]. Figure 7.2 shows the test results. The numbers in the boxes are the normalized score of each emotion received in that case. The horizontal bar shows how detectable the emotion is when it is shown. The vertical bars show how distinguishable the emotion is, i.e., how often it is guessed when it is not shown. Here, it is clear that for all rows, the diagonal is the highest number. This means that during any given test, the detected emotion was the same as the emotion the symbiote was showing. However, this is not the case for the columns. The stressed emotion is selected more often in the fearful state than in the stressed state. This means it is not distinguishable.



		Perceived					
		Relaxed	Playful	Alert/Interested	Stressed	Fearful	Defensive
Actual	Relaxed	0.73	0.27	0.48	0.13	0.13	0.15
	Playful	0.33	0.62	0.53	0.34	0.23	0.23
	Alert/Interested	0.28	0.43	0.71	0.36	0.32	0.32
	Stressed	0.10	0.16	0.46	0.73	0.69	0.37
	Fearful	0.01	0.03	0.48	0.88	0.98	0.34
	Defensive	0.13	0.28	0.55	0.52	0.38	0.79

Figure 7.2: Normalized test results for the emotion tests performed in [7]

7.2 Should Have Requirements

The fulfillment of the should-have requirements is shown in Table 7.3. All requirements are either successfully met, partially met, or not met. All these answers are justified with references to the corresponding sections in brackets. Let us discuss some of these requirements.

S.1 through S.03 are about detection and error correction. This report implemented elementary filters and error handling. These fulfilled the requirements but might fail in real-life implementations. This would require more research and could be the basis for another graduation project.

S.05 through S.08 are about the ease of building and modification. These are important not only because they are designed for a swarm robot. They also allow for educational value. These requirements help people who would like to continue the project. It is acknowledged that these requirements are not "very scientific" and are open to interpretation; nonetheless, they are key to the continuation of the project.

S.10 and S.11 are about emotion detection. S.10 is met, but S.11 is not. To test the requirements, emotion detection is fully implemented. However, because it did not meet the requirements, it was not added to the final design. This is also done because it would break Requirement M.13. Additional processing power and a better model would allow for successful implementation.

7.3 Could Have Requirements

The fulfillment of the could-have requirements is shown in Table 7.3. All requirements are either successfully met, partially met, or not met. All these answers are justified with references to the corresponding sections in brackets. Let us discuss some of these requirements.

C.01 is about the ability to determine the direction of loud sounds. Two microphones are implemented in the design. This means the possibility of this feature is available. However, it was not implemented. To successfully perform it, it might be necessary to add another processor.

Lastly, C.03 is about ROS 2. The software could be converted into ROS 2-compatible software. However, no ROS 2 notes were created during the project. This was mainly due to time constraints.

7.4 Discussion on the Requirements

The final evaluation of the requirements, as summarized in Table 7.5, highlights the overall success of the project in meeting its design objectives as given in [7].

Out of the 31 total requirements defined across all categories, 25 were fulfilled, five were partially fulfilled, and only one was not fulfilled. This high fulfillment rate reflects a design process strongly aligned with the initial project goals.

The must-have requirements, which define the essential functionality of the robot, were almost all successful. First, M.13 was partially fulfilled. This concerned the 10Hz minimum operation rate. It reflects the computational limitations encountered with the current hardware. Although the average operation rate was comfortably above the threshold (13.5Hz), edge cases dropped slightly below (9.7Hz). **Given the consistency in performance under normal conditions, this partial fulfillment still provides confidence in the system's usability, and upgrading from a 4GB RAM RP to a 16GB RAM RP would most likely solve this issue**. Second, S.10 was partially met due to issues in distinguishing between the stressed and fearful states. **From a biological point of view, these are very similar states, and this could be seen as a reasonable overlap. This nuance suggests that while the implementation is functional, there remains potential for refinement in expression**.

The should-have requirements were similarly well addressed, with 12 out of 13 fulfilled. These requirements mainly focused on production, modularity, and emotional expression. One requirement (S.11) was not met due to hardware limitations and a lack of optimization in the emotion recognition model. Better hardware and more focus on this model could reduce the processing time and create a higher performance with respect to accuracy. Thus a future version could fulfill this requirement.

The could-have requirements highlight areas of potential expansion. None were fully met, though two were partially fulfilled. These included directional sound detection and ROS 2 compatibility. In all cases, foundational work was done (i.e., stereo microphones and ROS-friendly architecture), but integration was postponed due to resource or time constraints. The only unmet requirement (C.03) was the budget constraint. Although the design exceeds €500, it remains within a reasonable cost range for research and development purposes. Furthermore, the bulk price is below €500.

Overall, the distribution of fulfilled versus unfulfilled requirements shows a project that fulfilled all requirements on the critical fronts. **The partial and unmet "Could Have" requirements serve not as shortcomings but as clear indicators of directions for future work, particularly around software integration and expression**.

This thorough fulfillment of the functional, technical, and interactional goals makes the developed symbiote not only a compelling proof of concept but also a solid foundation for further development and deployment in educational and research settings.

7.5 Answers to the Research Questions

Now that the requirements have been discussed let us continue with the research questions. Before answering the research question, the sub-research questions are discussed. The four sub-questions are linked to Chapters 2 through 5.

sub-question 1: What sensors are needed to detect human presence, location, and emotions?

To detect human presence, location, and emotions, the robot is equipped with vision and audio sensing that mimics human and animal perception. **A wide-angle RP Camera 3** is used for visual sensing. It enables face, body, and emotion detection within a field of view of 67° vertically and 102° horizontally and detects faces up to 1.16 meters and human presence up to 4.95 meters. For auditory input, **two Adafruit MAX4466 microphones** are employed in a stereophonic configuration, allowing the robot to detect loud sounds and, in future work, estimate their direction based on time differences in sound wave arrival. These sensors were selected not only for their technical suitability but also for their alignment with human emotional communication modalities, as vision and hearing account for over 90% of how humans perceive emotion.

sub-question 2: What data should be extracted from sensors to capture human behavior relevant for interaction?

The camera is used for **face, body, and emotion detection**. Facial data includes coordinates and sizes of bounding boxes, allowing the robot to estimate the position and proximity of individuals. Body detection similarly provides positional data that aids in distinguishing the closest person. Emotion recognition is performed using the FER2013 dataset model, classifying faces into seven basic emotions (Anger, Disgust, Fear, Happiness, Sad, Surprise, and Neutral). From the microphones, two forms of auditory data are extracted: **loudness levels and, optionally, directional cues** based on the time delay between left and right microphone inputs. This data enables the robot to interpret both spatial and affective human cues, forming the foundation for responsive interaction.

sub-question 3: How can the robot process the interpreted data to decide on an appropriate emotional state?

The robot uses a structured decision-making pipeline that includes **data interpretation, filtering, and an FSM for state management**. Interpreted data from the camera and microphones are combined into serial data packets and transmitted to the robot's decision-making area. Here, filtering mechanisms such as majority voting and moving averages are applied to ensure stability and reduce the influence of noise or transient signals. The FSM uses predefined transitional rules based on detected features, such as the number of faces, proximity of the closest person, detected emotion, and loud sound presence, to select an appropriate emotional state. This state then governs the behavior of the robot. The decision-making process is intentionally kept interpretable and straightforward, mirroring the limited but robust intelligence seen in insects and ensuring that system responses remain predictable and comprehensible to users.

sub-question 4: How can actuators be used to express a range of different emotional states intuitively?

The robot expresses emotional states using **four bio-inspired appendices: antennae, eyes, eyelids, and wheels.** Each of these actuators contributes to an expressive framework. The antennae simulate twitching or exploratory movements, the eyes shift to simulate attention or gaze, and the eyelids modulate openness to reflect states. The wheels enable dynamic motion to convey urgency or playfulness. These actuators are controlled hierarchically: the FSM dictates the current emotional state, which in turn triggers specific motor behaviors using functions designed with natural motion profiles (e.g., trapezoidal or sinusoidal velocity curves). This hierarchy allows for smooth and lifelike transitions between states, reinforcing the robot's expressiveness. By replicating animal-like cues, especially those seen in dogs and insects, the robot becomes more relatable, and its emotional output becomes intuitively interpretable to human observers.

Research Question: How can bio-inspired design aid in intuitive interactions between humans and swarm robots?

Bio-Inspired Design (BID) plays a crucial role in enabling intuitive interactions between humans and swarm robots **by leveraging patterns, forms, and behaviors found in nature. Particularly those observed in insects and social animals.** By mimicking familiar biological traits, such as expressive appendices or eye-following mechanisms, the robot's behavior becomes more interpretable to humans without the need for explicit instruction or user training. In this project, bio-inspired principles were implemented both in the physical design, such as antennae, eyes, and body form, and in the behavioral logic, which prioritizes simplicity, emergence, and responsiveness. These elements contribute to a more natural and engaging interaction paradigm, making the robot's intentions and emotional states easily recognizable to humans, thereby enhancing the quality of Human-Swarm Interaction (HSI).

7.6 Discussion on the Research Gap

This thesis addresses a clear research gap in the field of Human-Swarm Interaction (HSI), specifically the lack of physical, expressive swarm robots designed for intuitive interaction with humans. While prior studies have explored HSI predominantly through simulations or focused on swarm control without attention to social interaction, this work takes a novel approach by developing a functional, bio-inspired robotic symbiote that is capable of perceiving, interpreting, and expressing emotional cues in real-world environments.

By integrating principles from Bio-Inspired Design (BID), Swarm Robotics (SR), and Human-Robot Interaction (HRI), the project introduces a comprehensive framework for emotional and behavioral communication between humans and swarm agents. The robot developed in this work embodies core HSI challenges: it detects human presence and affective states, makes decisions based on multiple sensor inputs, and expresses its internal state through interpretable, biologically motivated actuators.

The implementation of real-time face, body, and emotion detection, along with the design of intuitive, expressive outputs, demonstrates that HSI can go beyond control and command models, evolving into a socially meaningful interaction. It aims to lay a foundation for further research into swarm behavior that not only responds to environmental stimuli but also to human emotional and social cues.

Moreover, by emphasizing modularity, open-source software, and affordability, the project promotes accessibility and scalability, enabling other researchers to replicate, adapt, and expand upon the system. Future research can build upon this work by exploring more complex social behaviors in robot swarms, incorporating machine learning for adaptive interactions, and studying long-term user engagement in dynamic environments such as public spaces, classrooms, or collaborative work settings. The more pressing technical challenges are discussed in Chapter 9.

In bridging this research gap, the thesis not only validates the feasibility of bio-inspired expressive swarm robotics but also provides a framework for advancing the social intelligence and usability of future autonomous systems.

Requirement	Fulfilled	Justification
M.01: Pass the device safety check of TU Delft	Yes	Device safety check was passed on 10-4-2025
M.02: Show predictable behavior	Yes	Designed using a FSM to show predictable behavior (4.4)
M.03: Detect human from ≥ 3 meters	Yes	Designed and tested up to 4.95m (6.3.4)
M.04: Detect at least 3 bodies	Yes	Designed and tested up to 4 bodies (6.3.4)
M.05: Detect face from 1 meter	Yes	Designed and tested up to 1.16m (6.3.4)
M.06: Detect at least 3 faces	Yes	Designed and tested up to 4 faces (6.3.4)
M.07: Discriminate the closest person	Yes	The closest person is discriminated using the largest bounding box (3)
M.08: Fully autonomous except power	Yes	Designed and tested to not need I/O devices and only require external power (6.3.2)
M.09: Be rechargeable	Yes	Robot is rechargeable through the rechargeable power bank (6.2.2)
M.10: Run for at least 3 hours	Yes	Designed and tested to operate for 8.1 hours (6.3.2)
M.11: Move around on a flat surface	Yes	Designed and tested to drive, forward, backward, and steer (5.2.7)
M.12: Startup without I/O devices	Yes	Designed and tested to start and shut-off safely without I/O devices using systemd and overlay (6.1.2)
M.13: Operate at 10Hz	Partially	tested average is 13.5Hz and the tested worst case is 9.7Hz (6.3.3)
M.14: Camera angle $\geq 60°$ vertically and horizontally	Yes	Selected camera has 67° vertical, 102° horizontal (2.1.2)
M.15: Show distinguishable emotions	Partially	tested, 5 out of 6 emotions were distinguishable (E)
M.16: Show detectable emotions	Yes	tested, all emotions were detectable (E)

Table 7.1: Evaluation of must-have requirements fulfillment, with section reference in brackets

Requirement	Fulfilled	Justification
S.01: Detect loud noises > 70 dB	Yes	Tested to detect sound louder than 70 dB (6.2.2)
S.02: Not triggered by conversation < 65 dB	Yes	Tested not to detect sound less loud than 65 dB (6.2.2)
S.03: Error handling on false positives/negatives	Yes	Moving average and majority voting filters implemented (4.2.2)
S.04: Rechargeable using a universal charger	Yes	Power bank is rechargeable through a USB-C cable (6.2.2)
S.05: Allow for modifications	Yes	Uses Arduino and RP to allow for modifications (3.8)
S.06: Allow options for additional sensors	Yes	Arduino and RP allow for adding sensors (3.8)
S.07: Use standard protocols	Yes	Uses standard I2C, Serial and CSV formatting (3.8)
S.08: Use basic tools to construct	Yes	Constructed using screwdriver and soldering iron (6.2.4)
S.09: Use materials with nationwide availability	Yes	Design uses materials from large companies (A)
S.10: Detect emotion within 200 ms	Yes	tested, emotion detection within 200 ms (E)
S.11: Detect emotion with 90% accuracy	No	accuracy was not above 66% (3.6)
S.12: Move naturally with no perceived jumps in speed	Yes	Implemented continuous speed profile (5.2.4)
S.13: Less than 500 euros when bought in bulk	Yes	Bill of Materials shows the cost of € 345.35 in bulk(6.3.1)

Table 7.3: Evaluation of should-have requirements fulfillment, with section reference in brackets

Requirement	Fulfilled	Justification
C.01: Detect the direction of sounds	Partially	Hardware setup supports stereo sound, but not implemented (3.4)
C.02: ROS 2 compatible	Partially	Uses ROS 2-compatible architecture but no explicit ROS 2 integration (6.1.3)
C.03: Cost less than 500 euros	No	Bill of Materials shows cost of € 654.41 in bulk (6.3.1)

Table 7.4: Evaluation of could-have requirements fulfillment, with section reference in brackets

Requirement Type	Fulfilled	Partially Fulfilled	Not Fulfilled	Total
Must Have	14	2	0	16
Should Have	12	0	1	13
Could Have	0	2	1	3
Total	**26**	**4**	**2**	**32**

Table 7.5: Summary of requirement fulfillment across categories

Chapter 8

Conclusion

This thesis explored the development of a bio-inspired symbiotic robot designed for expressive and intuitive interaction within human-swarm environments. The project successfully translated biologically motivated sensing and actuation into an embedded robotic platform, showing that simplicity in system design can lead to rich, interpretable behavior.

By mimicking sensory modalities common in nature, such as vision and hearing, the robot was able to interpret basic human cues like presence, movement, and loud vocalizations. These inputs, processed through a lightweight and predictable embedded architecture, enabled the robot to express a limited but meaningful emotional range through actuated appendices. The integration of sensors, interpretation, decision-making, and actuation show the feasibility of expressive swarm robots that prioritize real-world interaction over simulated intelligence.

Moreover, the design process emphasized expandability, reproducibility, and affordability, aiming to lower the entry barrier for future research and educational use. The symbiote, FLIP, serves not only as a proof-of-concept but also as a framework upon which further advancements in emotional expressivity, behavioral complexity, and swarm coordination can be built.

Ultimately, this work contributes to the growing field of Human-Swarm Interaction by offering a physical, approachable system that bridges the gap between synthetic behavior and natural communication. It demonstrates that with deliberate design and system-wide coherence, even simple robots can engage meaningfully with the humans around them.

Chapter 9

Future Work

While the current implementation demonstrates the feasibility of an intuitive, bio-inspired swarming robot, several promising directions for future development could enhance both technical performance and interactive richness.

Sensors Adding touch sensors to the antennae would expand the robot's ability to respond to physical interactions. Touch sensing could enable more nuanced human-robot interactions, especially in crowded or dynamic environments. Next to this, stereo cameras could be used for better distance information, navigation, and object detection.

Interpretation Several improvements could be made to the sensory interpretation pipeline. Fine-tuning the existing parameters for detection thresholds and filtering could increase reliability and responsiveness. Implementing stereo sound directionality would allow the robot to localize human presence more accurately, especially when paired with voice-based inputs such as pitch, tone, or even voice recognition. Additionally, pose detection would enable a better understanding of human intent and body language. Emotion detection could also be revisited with a more powerful onboard processor, enabling richer affective interactions. Finally, incorporating parity bits and data validation would strengthen the reliability of the communication and processing pipeline.

Actuation Future iterations could benefit from quieter, more efficient motors to reduce noise pollution during interaction. New expressive features, such as a moving mouth or brows, could significantly broaden the range of emotional states that can be conveyed.

Software and Embedded Systems On the software side, expanding functionality and adding modular control options would improve scalability and maintainability. Introducing structured safety checks, better data handling, and advanced filtering methods would enhance both system robustness and longevity. Features such as stereo sound interpretation, voice recognition, and tactile input integration could also be further explored in this context.

Mechanical Engineering Mechanically, the robot could be improved by reinforcing structural components and using back-drivable motors for smoother and more compliant movement. Implementing autonomous charging and refining control systems for locomotion would allow for extended and more independent operation.

Industrial Design For an industrial designer, the focus on expressive behavior and emotional communication could be interesting. Furthermore, the production process and mechanical design can be improved with a focus on manufacturability and visual appeal.

Bibliography

[1] Z. Zhao, Q. Yang, R. Li, *et al.*, "A comprehensive review on the evolution of bio-inspired sensors from aquatic creatures," *Cell Reports Physical Science*, vol. 5, p. 102 064, 7 Jul. 2024, ISSN: 2666-3864. DOI: 10.1016/J.XCRP.2024.102064.

[2] M. J. Benton, P. C. Donoghue, R. J. Asher, M. Friedman, T. J. Near, and J. Vinther, "Constraints on the timescale of animal evolutionary history," *Palaeontologia Electronica*, vol. 18, 1 2015, ISSN: 10948074. DOI: 10.26879/424. [Online]. Available: https://www.researchgate.net/publication/272741232_Constraints_on_the_timescale_of_animal_evolutionary_history.

[3] M. Hrncir, F. R. Barth, and J. R. Tautz, *32 vibratory and airborne-sound signals in bee communication (hymenoptera)*, 2025.

[4] C. R. Smith, A. Dolezal, D. Eliyahu, C. T. Holbrook, and J. Gadau, "Ants (formicidae): Models for social complexity," *Cold Spring Harbor Protocols*, vol. 2009, pdb.emo125, 7 Jul. 2009, ISSN: 1940-3402. DOI: 10.1101/PDB.EMO125. [Online]. Available: http://cshprotocols.cshlp.org/content/2009/7/pdb.emo125.full%20http://cshprotocols.cshlp.org/content/2009/7/pdb.emo125%20http://cshprotocols.cshlp.org/content/2009/7/pdb.emo125.abstract.

[5] R. D. Arnold, H. Yamaguchi, and T. Tanaka, "Search and rescue with autonomous flying robots through behavior-based cooperative intelligence," *Journal of International Humanitarian Action 2018 3:1*, vol. 3, pp. 1–18, 1 Dec. 2018, ISSN: 2364-3404. DOI: 10.1186/S41018-018-0045-4. [Online]. Available: https://jhumanitarianaction.springeropen.com/articles/10.1186/s41018-018-0045-4.

[6] R. Arnold, K. Carey, B. Abruzzo, and C. Korpela, *What is a robot swarm: A definition for swarming robotics*, Feb. 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8993024https://ieeexplore.ieee.org/abstract/document/8993024.

[7] A. Rozendaal, *Design of a small swarming robot showing intuitive human interaction - integrated product design report*, 2025.

[8] J. Duan, Y. A. Zhu, and S. Huang, "Stigmergy agent and swarm-intelligence-based multiagent system," *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*, pp. 720–724, 2012. DOI: 10.1109/WCICA.2012.6357972.

[9] M. J. Mens, F. Klijn, K. M. de Bruijn, and E. van Beek, "The meaning of system robustness for flood risk management," *Environmental Science & Policy*, vol. 14, pp. 1121–1131, 8 Dec. 2011, ISSN: 1462-9011. DOI: 10.1016/J.ENVSCI.2011.08.003.

[10] T. Norman, "Access control system servers and workstations," *Electronic Access Control*, pp. 243–260, 2012. DOI: 10.1016/B978-0-12-382028-0.00018-1.

[11] G. Kapellmann-Zafra, N. Salomons, A. Kolling, and R. Groß, "Human-robot swarm interaction with limited situational awareness," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9882 LNCS, pp. 125–136, 2016, issn: 16113349. doi: 10.1007/978-3-319-44427-7_11/FIGURES/5. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-44427-7_11.

[12] S. Kerman, D. Brown, and M. A. Goodrich, "Supporting human interaction with robust robot swarms," *Proceedings - 2012 5th International Symposium on Resilient Control Systems, ISRCS 2012*, pp. 197–202, 2012. doi: 10.1109/ISRCS.2012.6309318.

[13] A. M. Naghsh, J. Gancet, A. Tanoto, and C. Roast, "Analysis and design of human-robot swarm interaction in firefighting," *Proceedings of the 17th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN*, pp. 255–260, 2008. doi: 10.1109/ROMAN.2008.4600675.

[14] A. Kolling, P. Walker, N. Chakraborty, K. Sycara, and M. Lewis, "Human interaction with robot swarms: A survey," *IEEE Transactions on Human-Machine Systems*, vol. 46, pp. 9–26, 1 Feb. 2016, issn: 21682291. doi: 10.1109/THMS.2015.2480801.

[15] B. Pendleton and M. A. Goodrich, *Scalable human interaction with robotic swarms*, 2013. doi: 10.2514/6.2013-4731.

[16] J. Alonso-Mora, S. H. Lohaus, P. Leemann, R. Siegwart, and P. Beardsley, "Gesture based human - multi-robot swarm interaction and its application to an interactive display," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2015-June, pp. 5948–5953, June Jun. 2015, issn: 10504729. doi: 10.1109/ICRA.2015.7140033.

[17] Delta, *Cyberzoo: Een dierentuin met robots - delta*, 2014. [Online]. Available: https://delta.tudelft.nl/article/cyberzoo-een-dierentuin-met-robots.

[18] T. Delft, *Mirte*, 2025. [Online]. Available: https://mirte.org/.

[19] E. Miranda, "Moscow rules: A quantitative exposé," *Lecture Notes in Business Information Processing*, vol. 445 LNBIP, pp. 19–34, 2022, issn: 18651356. doi: 10.1007/978-3-031-08169-9_2. [Online]. Available: https://www.researchgate.net/publication/356836488_MoSCoW_Rules_A_quantitative_expose_Accepted_for_presentation_at_XP2022.

[20] R. Brooks, "Intelligence without representation," *Artificial Intelligence*, 1991.

[21] A. Mehrabian and M. Wiener, "Decoding of inconsistent communications.," *Journal of Personality and Social Psychology*, vol. 6, pp. 109–114, 1 1967, issn: 1939-1315. doi: 10.1037/h0024532.

[22] R. Pi, *Camera - raspberry pi documentation*, 2023. [Online]. Available: https://www.raspberrypi.com/documentation/accessories/camera.html.

[23] Kaggle, *Fer-2013*, 2013. [Online]. Available: https://www.kaggle.com/datasets/msambare/fer2013.

[24] P. Ekman and W. Friesen, *Unmasking the face: A guide to recognizing emotions from facial clues*, 1975.

[25] Wikipedia, *16:9 aspect ratio - wikipedia*, 2025. [Online]. Available: https://en.wikipedia.org/wiki/16:9_aspect_ratio.

[26] openCVDNN, *Opencv: Deep neural networks (dnn module)*, 2025. [Online]. Available: https://docs.opencv.org/4.x/d2/d58/tutorial_table_of_content_dnn.html.

[27] DlibCNN, *Face-recognition.js-models/models at master · justadudewhohacks/face-recognition.js-models*, 2025. [Online]. Available: https://github.com/justadudewhohacks/face-recognition.js-models/tree/master/models.

[28] Google, *Hollance/blazeface-pytorch: The blazeface face detector model implemented in pytorch*, 2025. [Online]. Available: https://github.com/hollance/BlazeFace-PyTorch.

[29] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2001, ISSN: 10636919. DOI: 10.1109/CVPR.2001.990517. [Online]. Available: https://www.researchgate.net/publication/3940582_Rapid_Object_Detection_using_a_Boosted_Cascade_of_Simple_Features.

[30] openCVHaarcascades, *Opencv/data/haarcascades at master · opencv/opencv*, 2025. [Online]. Available: https://github.com/opencv/opencv/tree/master/data/haarcascades.

[31] M. Wasala and T. Kryjak, "Real-time hog+svm based object detection using soc fpga for a uhd video stream," Apr. 2022. DOI: 10.1109/MECO55406.2022.9797113. [Online]. Available: http://arxiv.org/abs/2204.10619%20http://dx.doi.org/10.1109/MECO55406.2022.9797113.

[32] DlibHOG, *Face detection with dlib (hog and cnn) - pyimagesearch*, 2025. [Online]. Available: https://pyimagesearch.com/2021/04/19/face-detection-with-dlib-hog-and-cnn/.

[33] DlibLandmarks, *Italojs/facial-landmarks-recognition*, 2025. [Online]. Available: https://github.com/italojs/facial-landmarks-recognition/tree/master.

[34] J. Singh, A. Singh, K. K. Singh, *et al.*, "Real-time convolutional neural networks for emotion and gender classification," *Procedia Computer Science*, vol. 235, pp. 1429–1435, 2024, ISSN: 18770509. DOI: 10.1016/j.procs.2024.04.134. [Online]. Available: https://arxiv.org/pdf/1710.07557.

[35] Z. Zhang, "Deep face emotion recognition," *Journal of Physics: Conference Series*, vol. 1087, 6 Oct. 2018, ISSN: 17426596. DOI: 10.1088/1742-6596/1087/6/062036. [Online]. Available: https://www.researchgate.net/publication/328035819_Deep_Face_Emotion_Recognition.

[36] ultralytics, *Explore ultralytics yolov8 - ultralytics yolo docs*, 2023. [Online]. Available: https://docs.ultralytics.com/models/yolov8/.

[37] Qiuqiu, *Dog-qiuqiu/yolo-fastest: :zap: Based on yolo's ultra-lightweight universal target detection algorithm, the calculation amount is only 250mflops, the ncnn model size is only 666kb, the raspberry pi 3b can run up to 15fps+, and the mobile terminal can run up to 178fps+*, 2021. [Online]. Available: https://github.com/dog-qiuqiu/Yolo-Fastest/tree/master.

[38] L. Prevost, P. Negri, X. Clady, and S. M. Hanif, "A cascade of boosted generative and discriminative classifiers for vehicle detection," *Eurasip Journal on Advances in Signal Processing*, vol. 2008, 2008, ISSN: 16876172. DOI: 10.1155/2008/782432.

[39] M. Soori, B. Arezoo, and R. Dastres, "Artificial intelligence, machine learning and deep learning in advanced robotics, a review," *Cognitive Robotics*, vol. 3, pp. 54–70, Jan. 2023, ISSN: 2667-2413. DOI: 10.1016/J.COGR.2023.04.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667241323000113?utm_source=chatgpt.com.

[40] T. Zhang and H. Mo, "Reinforcement learning for robot research: A comprehensive review and open issues," *International Journal of Advanced Robotic Systems*, vol. 18, 3 2021, ISSN: 17298814. DOI: 10.1177/17298814211007305/ASSET/C6D2CE9F-E560-4CD4-A406-91CFDC9CB62C/ASSETS/IMAGES/LARGE/10.1177_17298814211007305-FIG11.JPG. [Online]. Available: https://scholar.google.com/scholar_url?url=https://journals.sagepub.com/doi/pdf/10.1177/17298814211007305&hl=nl&sa=T&oi=ucasa&ct=ufr&ei=020baL2-CrOi6rQPvJGpyAE&scisig=AAZF9b_jgjZuZmMo_X7jf97gdKLG.

[41] E. R. M. Aleluya, A. D. Zamayla, and S. L. M. Tamula, "Decision-making system of soccer-playing robots using finite state machine based on skill hierarchy and path planning through bezier polynomials," *Procedia Computer Science*, vol. 135, pp. 230–237, Jan. 2018, issn: 1877-0509. doi: 10.1016/J.PROCS.2018.08.170. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050918314595?utm_source=chatgpt.com.

[42] Y. Hu, C. Liu, M. Zhang, Y. Lu, Y. Jia, and Y. Xu, "An ontology and rule-based method for human–robot collaborative disassembly planning in smart remanufacturing," *Robotics and Computer-Integrated Manufacturing*, vol. 89, p. 102 766, Oct. 2024, issn: 0736-5845. doi: 10.1016/J.RCIM.2024.102766. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0736584524000528?utm_source=chatgpt.com.

[43] R. Peddi and N. Bezzo, "A decision tree-based monitoring and recovery framework for autonomous robots with decision uncertainties," *IEEE International Conference on Intelligent Robots and Systems*, pp. 9722–9728, Aug. 2023, issn: 21530866. doi: 10.1109/IROS55552.2023.10342207. [Online]. Available: https://arxiv.org/pdf/2308.00944.

[44] B. Chen, A. Zhang, and L. Cao, "Autonomous intelligent decision-making system based on bayesian som neural network for robot soccer," *Neurocomputing*, vol. 128, pp. 447–458, Mar. 2014, issn: 0925-2312. doi: 10.1016/J.NEUCOM.2013.08.021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231213008977?utm_source=chatgpt.com.

[45] Quesada, *Release bluepad32 v4.0 · ricardoquesada/bluepad32*, 2024. [Online]. Available: https://github.com/ricardoquesada/bluepad32/releases/tag/4.0.

[46] F. Levillain and E. Zibetti, "Behavioral objects: The rise of the evocative machines," *Journal of Human-Robot Interaction*, vol. 6, pp. 4–24, 1 2017.

Acronyms

BID Bio-Inspired Design

BoM Bill of Materials

CSV Comma Separated Values

DUT Delft University of Technology

ES Embedded Systems

FLIP Friendly Logic-based Interactive Presence

FPS Frames per Second

FSM Finite State Machine

HRI Human-Robot Interaction

HSI Human-Swarm Interaction

IPD Integrated Product Design

LZ Lunar Zebro

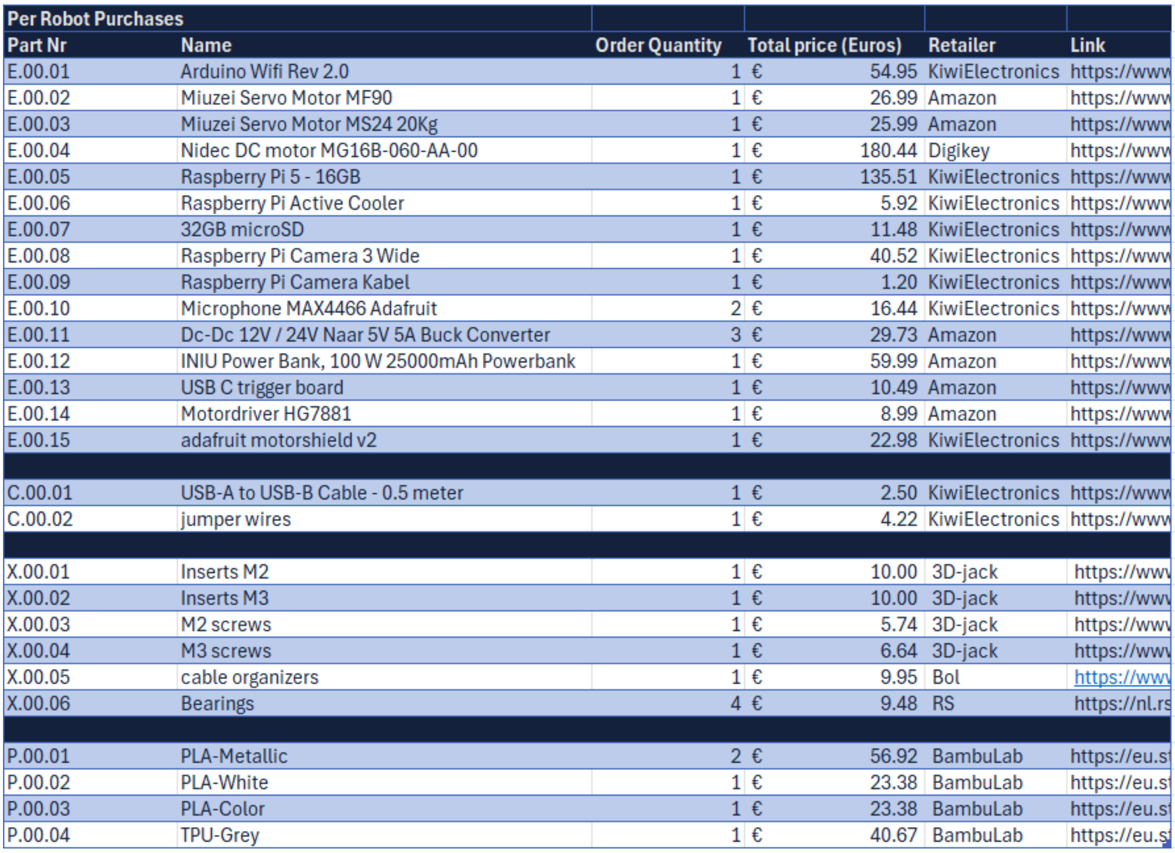
RP Raspberry Pi

SR Swarm Robotics

Appendix A

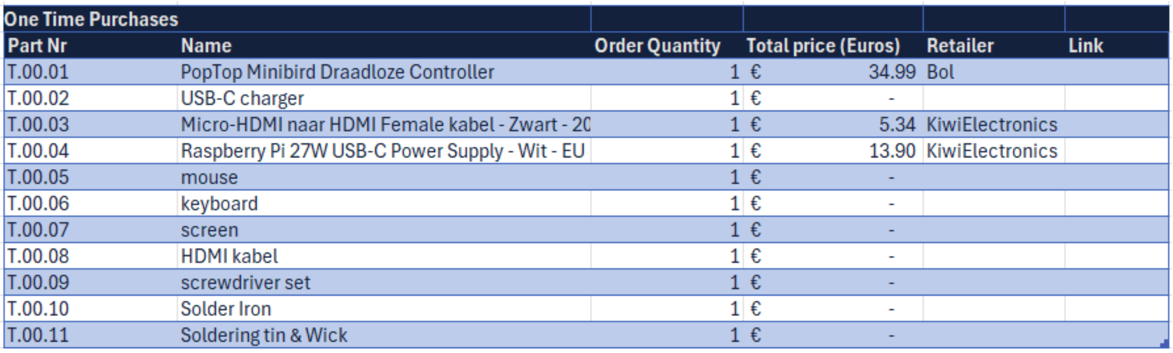
Bill of Materials

This Appendix contains the URL link to the GitHub page containing the up-to-date BoM. The link is: `https://github.com/aartrozendaal/flip/tree/main`. Furthermore, the BoM, as of the date of publishing, is shown in Figures A.1 with all the components in the robot, A.2 with all tools and one time purchases, and A.3 with a summary and delivery cost breakdown.



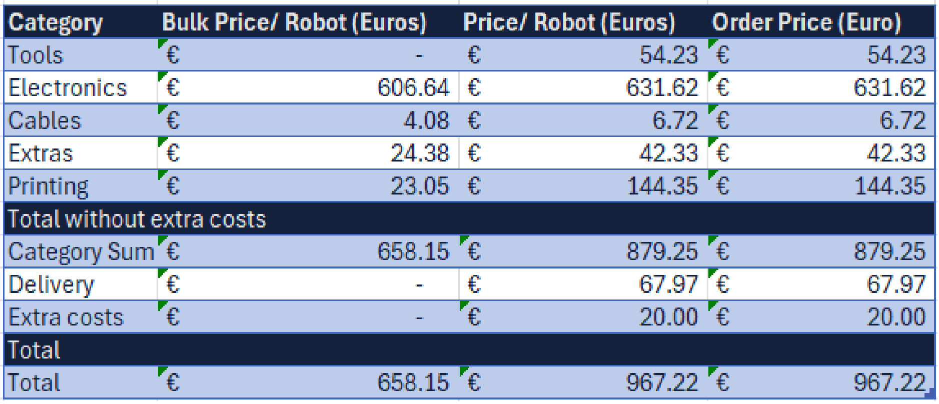
Per Robot Purchases						
Part Nr	Name	Order Quantity	Total price (Euros)		Retailer	Link
E.00.01	Arduino Wifi Rev 2.0	1	€	54.95	KiwiElectronics	https://www
E.00.02	Miuzei Servo Motor MF90	1	€	26.99	Amazon	https://www
E.00.03	Miuzei Servo Motor MS24 20Kg	1	€	25.99	Amazon	https://www
E.00.04	Nidec DC motor MG16B-060-AA-00	1	€	180.44	Digikey	https://www
E.00.05	Raspberry Pi 5 - 16GB	1	€	135.51	KiwiElectronics	https://www
E.00.06	Raspberry Pi Active Cooler	1	€	5.92	KiwiElectronics	https://www
E.00.07	32GB microSD	1	€	11.48	KiwiElectronics	https://www
E.00.08	Raspberry Pi Camera 3 Wide	1	€	40.52	KiwiElectronics	https://www
E.00.09	Raspberry Pi Camera Kabel	1	€	1.20	KiwiElectronics	https://www
E.00.10	Microphone MAX4466 Adafruit	2	€	16.44	KiwiElectronics	https://www
E.00.11	Dc-Dc 12V / 24V Naar 5V 5A Buck Converter	3	€	29.73	Amazon	https://www
E.00.12	INIU Power Bank, 100 W 25000mAh Powerbank	1	€	59.99	Amazon	https://www
E.00.13	USB C trigger board	1	€	10.49	Amazon	https://www
E.00.14	Motordriver HG7881	1	€	8.99	Amazon	https://www
E.00.15	adafruit motorshield v2	1	€	22.98	KiwiElectronics	https://www
C.00.01	USB-A to USB-B Cable - 0.5 meter	1	€	2.50	KiwiElectronics	https://www
C.00.02	jumper wires	1	€	4.22	KiwiElectronics	https://www
X.00.01	Inserts M2	1	€	10.00	3D-jack	https://www
X.00.02	Inserts M3	1	€	10.00	3D-jack	https://www
X.00.03	M2 screws	1	€	5.74	3D-jack	https://www
X.00.04	M3 screws	1	€	6.64	3D-jack	https://www
X.00.05	cable organizers	1	€	9.95	Bol	https://www
X.00.06	Bearings	4	€	9.48	RS	https://nl.rs
P.00.01	PLA-Metallic	2	€	56.92	BambuLab	https://eu.st
P.00.02	PLA-White	1	€	23.38	BambuLab	https://eu.st
P.00.03	PLA-Color	1	€	23.38	BambuLab	https://eu.st
P.00.04	TPU-Grey	1	€	40.67	BambuLab	https://eu.st

Figure A.1: Bill of Materials for all Robot components



One Time Purchases					
Part Nr	Name	Order Quantity	Total price (Euros)	Retailer	Link
T.00.01	PopTop Minibird Draadloze Controller	1	€ 34.99	Bol	
T.00.02	USB-C charger	1	€ -		
T.00.03	Micro-HDMI naar HDMI Female kabel - Zwart - 20	1	€ 5.34	KiwiElectronics	
T.00.04	Raspberry Pi 27W USB-C Power Supply - Wit - EU	1	€ 13.90	KiwiElectronics	
T.00.05	mouse	1	€ -		
T.00.06	keyboard	1	€ -		
T.00.07	screen	1	€ -		
T.00.08	HDMI kabel	1	€ -		
T.00.09	screwdriver set	1	€ -		
T.00.10	Solder Iron	1	€ -		
T.00.11	Soldering tin & Wick	1	€ -		

Figure A.2: Bill of Materials for one-time purchase components



Category	Bulk Price/ Robot (Euros)	Price/ Robot (Euros)	Order Price (Euro)
Tools	€ -	€ 54.23	€ 54.23
Electronics	€ 606.64	€ 631.62	€ 631.62
Cables	€ 4.08	€ 6.72	€ 6.72
Extras	€ 24.38	€ 42.33	€ 42.33
Printing	€ 23.05	€ 144.35	€ 144.35
Total without extra costs			
Category Sum	€ 658.15	€ 879.25	€ 879.25
Delivery	€ -	€ 67.97	€ 67.97
Extra costs	€ -	€ 20.00	€ 20.00
Total			
Total	€ 658.15	€ 967.22	€ 967.22

Figure A.3: Bill of Materials summary with delivery costs

Appendix B

Python Files

This Appendix contains the URL links to the GitHub page containing all the Python test files.

B.1 Face Detection

This Appendix contains the URL link to the GitHub page containing all the Python test files considering face recognition. The link is: `https://github.com/aartrozendaal/flip/tree/main/face_recognition`

B.2 Emotion Detection

This Appendix contains the URL link to the GitHub page containing all the Python test files considering emotion recognition. The link is: `https://github.com/aartrozendaal/flip/tree/main/emotion_recognition`

B.3 Body Detection

This Appendix contains the URL link to the GitHub page containing all the Python test files, considering body detection recognition. The link is: `https://github.com/aartrozendaal/flip/tree/main/body_recognition`

Appendix C

C++ Files

This Appendix contains the URL link to the GitHub page containing all the C++ test files. The link is: `https://github.com/aartrozendaal/flip/tree/main/arduino/src`

Appendix D

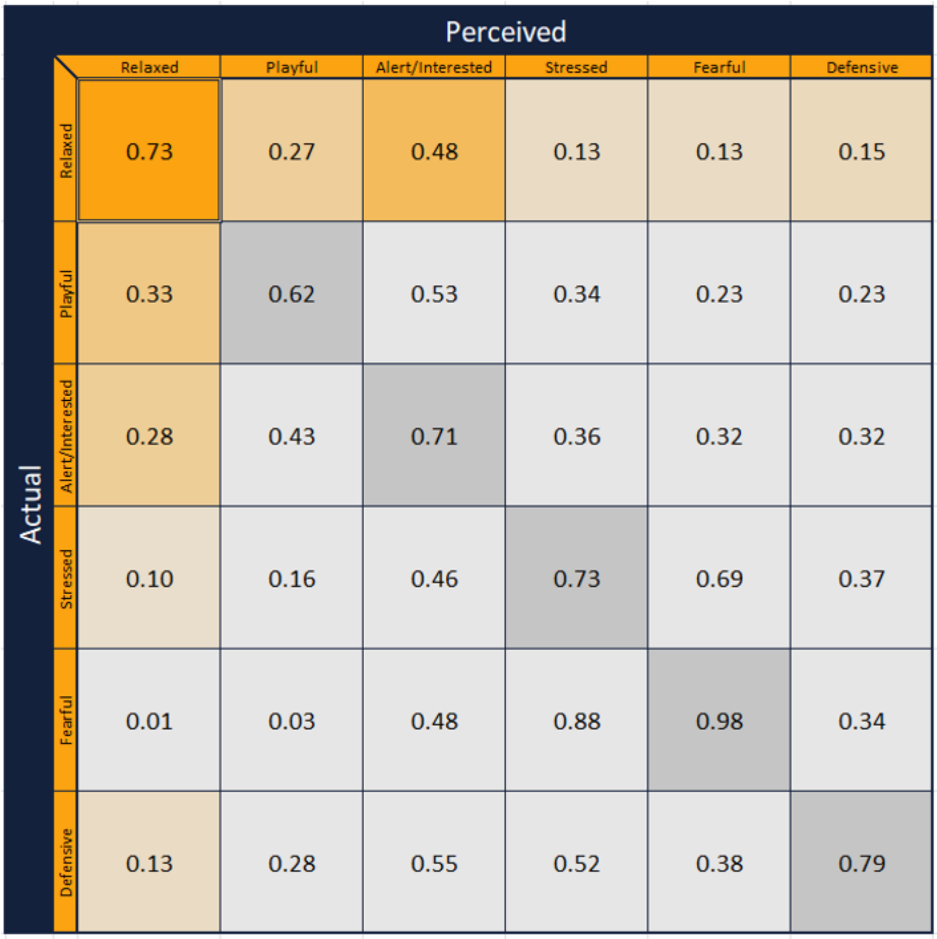
Header Files

This Appendix contains the URL link to the GitHub page containing all header files. The link is: `https://github.com/aartrozendaal/flip/tree/main/arduino/include`

Appendix E

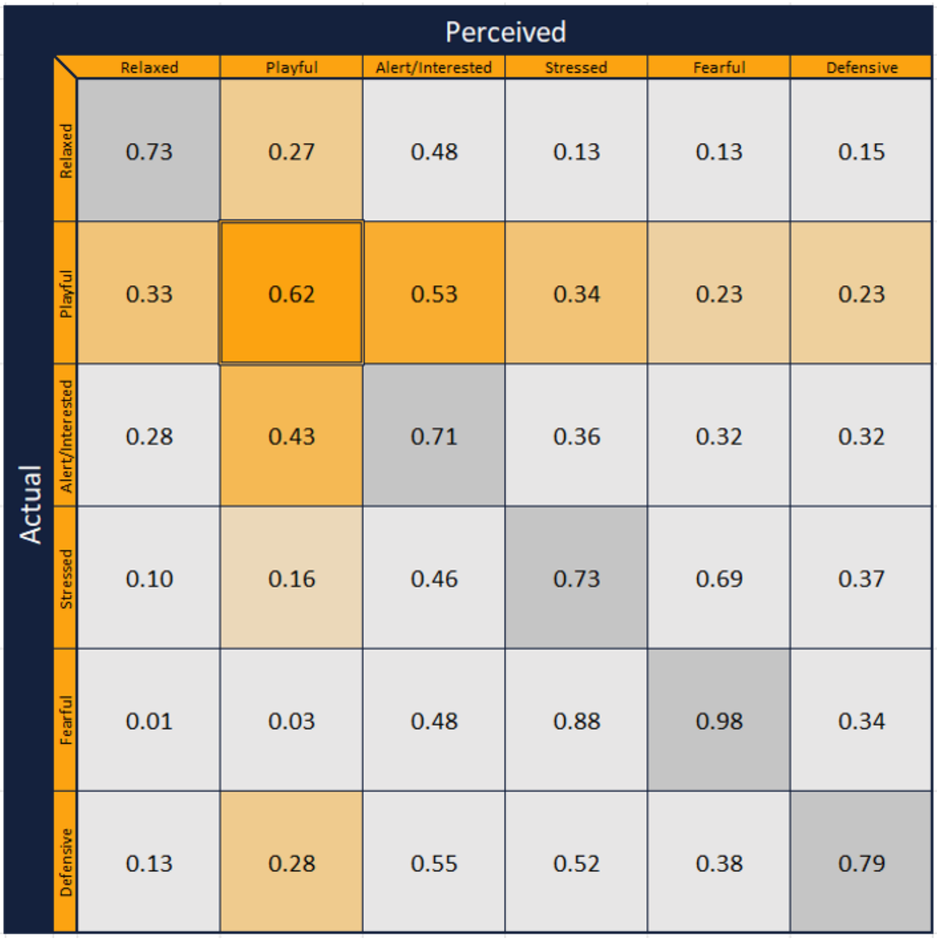
Emotion Test Results

Figure E.1 through E.6 shows the test results of the emotion test performed in [7]. Each figure highlights another emotion. The full method and test are mentioned in the other report, but the results are discussed here. The horizontal bar shows how detectable the emotion is when it is shown. The vertical bars show how distinguishable the emotion is, i.e. how often it is guessed when it is not shown.



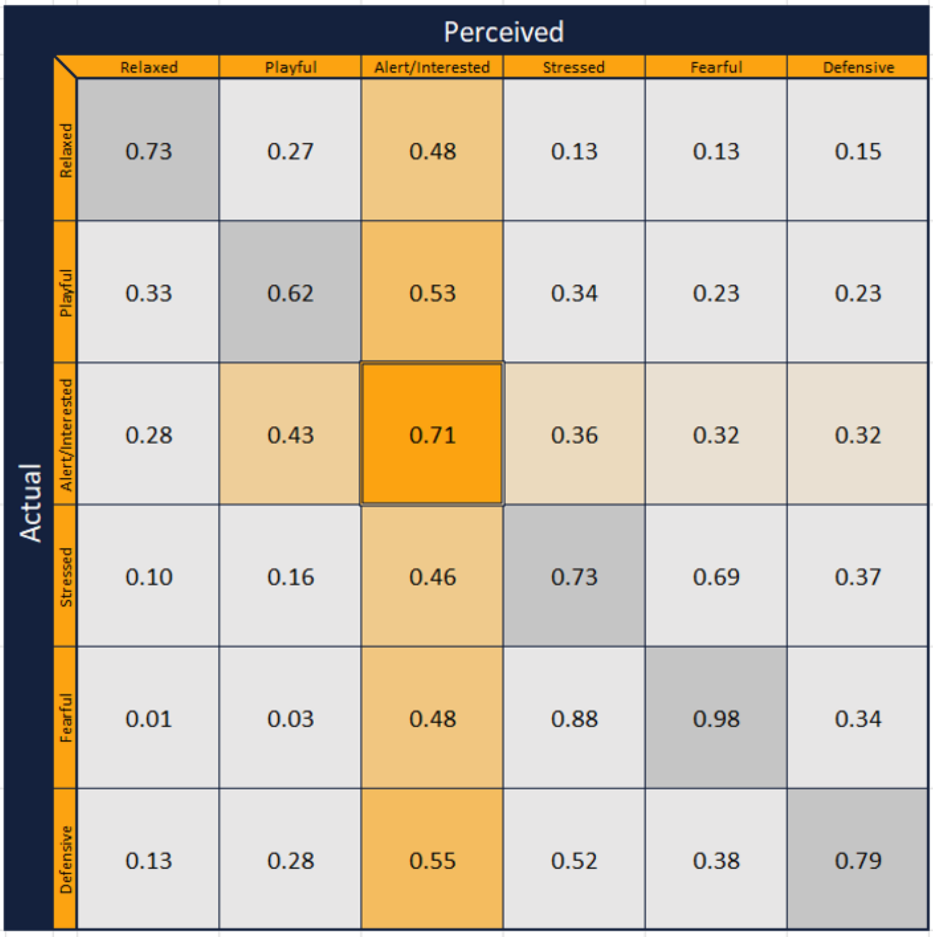
		Perceived					
		Relaxed	Playful	Alert/Interested	Stressed	Fearful	Defensive
Actual	Relaxed	0.73	0.27	0.48	0.13	0.13	0.15
	Playful	0.33	0.62	0.53	0.34	0.23	0.23
	Alert/Interested	0.28	0.43	0.71	0.36	0.32	0.32
	Stressed	0.10	0.16	0.46	0.73	0.69	0.37
	Fearful	0.01	0.03	0.48	0.88	0.98	0.34
	Defensive	0.13	0.28	0.55	0.52	0.38	0.79

Figure E.1: Test Results for the emotion tests performed in [7] highlighting the relaxed emotion



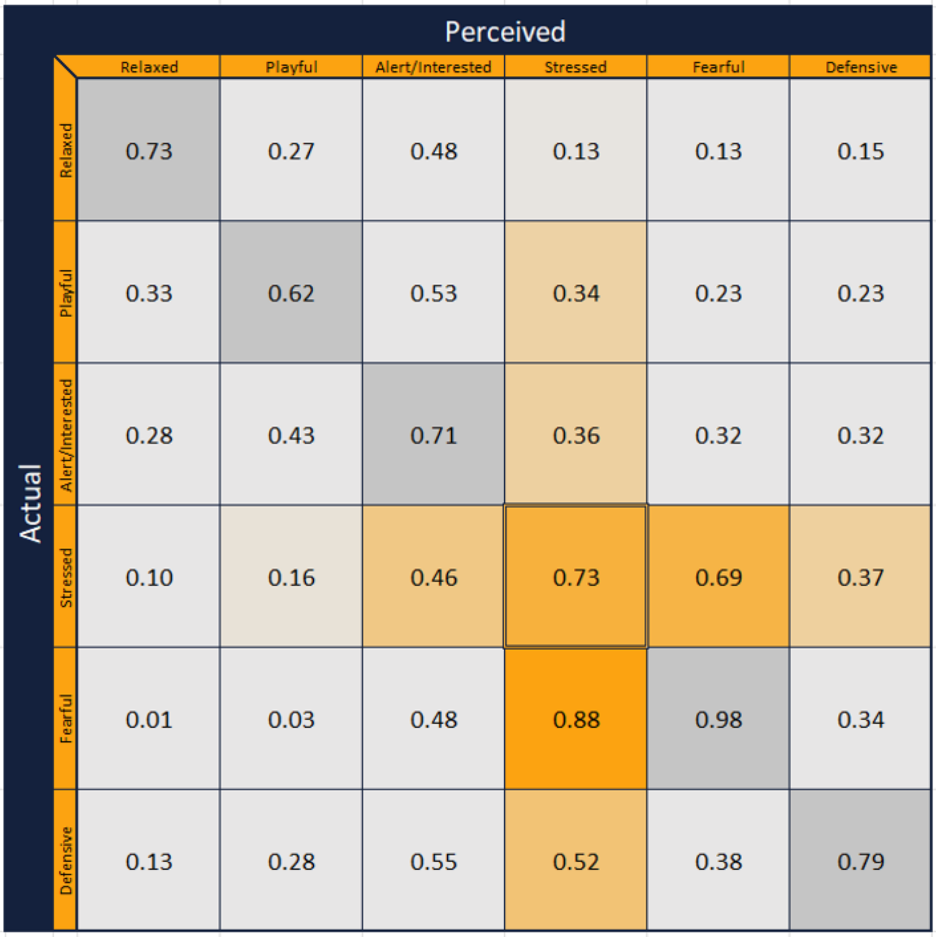
		Perceived					
		Relaxed	Playful	Alert/Interested	Stressed	Fearful	Defensive
Actual	Relaxed	0.73	0.27	0.48	0.13	0.13	0.15
	Playful	0.33	0.62	0.53	0.34	0.23	0.23
	Alert/Interested	0.28	0.43	0.71	0.36	0.32	0.32
	Stressed	0.10	0.16	0.46	0.73	0.69	0.37
	Fearful	0.01	0.03	0.48	0.88	0.98	0.34
	Defensive	0.13	0.28	0.55	0.52	0.38	0.79

Figure E.2: Test Results for the emotion tests performed in [7] highlighting the playful emotion



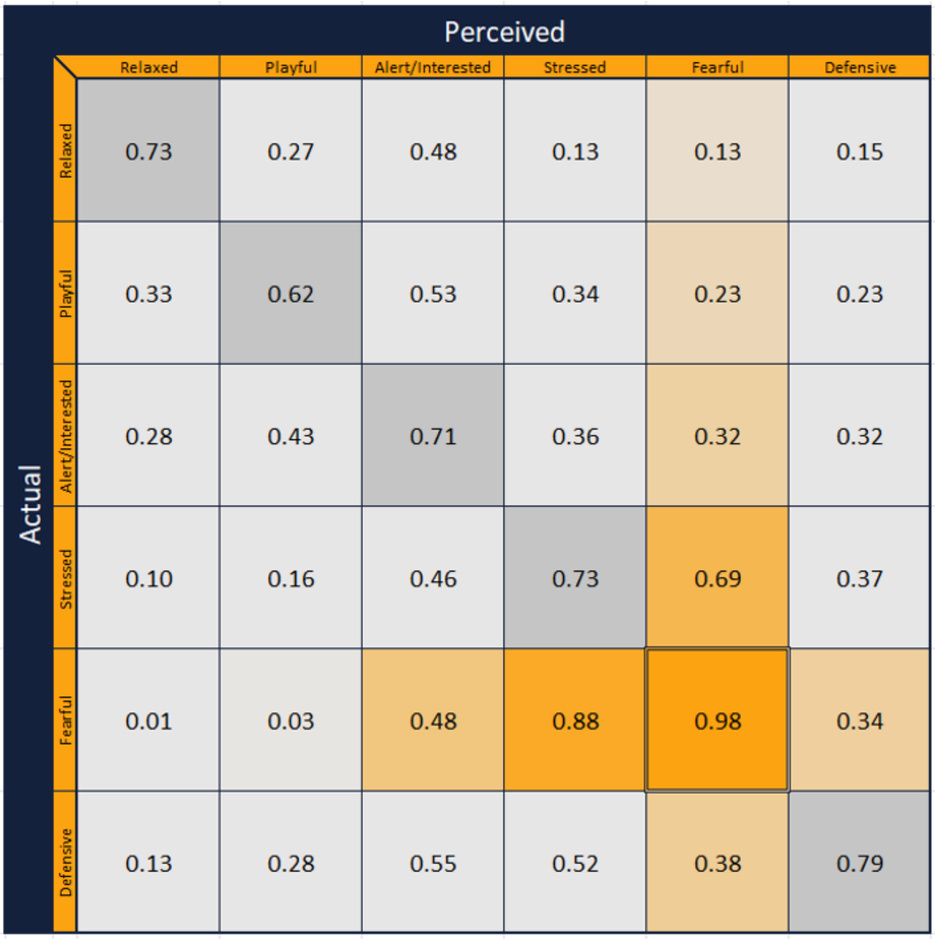
Actual \ Perceived	Relaxed	Playful	Alert/Interested	Stressed	Fearful	Defensive
Relaxed	0.73	0.27	0.48	0.13	0.13	0.15
Playful	0.33	0.62	0.53	0.34	0.23	0.23
Alert/Interested	0.28	0.43	0.71	0.36	0.32	0.32
Stressed	0.10	0.16	0.46	0.73	0.69	0.37
Fearful	0.01	0.03	0.48	0.88	0.98	0.34
Defensive	0.13	0.28	0.55	0.52	0.38	0.79

Figure E.3: Test Results for the emotion tests performed in [7] highlighting the alert emotion



	Perceived					
Actual	Relaxed	Playful	Alert/Interested	Stressed	Fearful	Defensive
Relaxed	0.73	0.27	0.48	0.13	0.13	0.15
Playful	0.33	0.62	0.53	0.34	0.23	0.23
Alert/Interested	0.28	0.43	0.71	0.36	0.32	0.32
Stressed	0.10	0.16	0.46	0.73	0.69	0.37
Fearful	0.01	0.03	0.48	0.88	0.98	0.34
Defensive	0.13	0.28	0.55	0.52	0.38	0.79

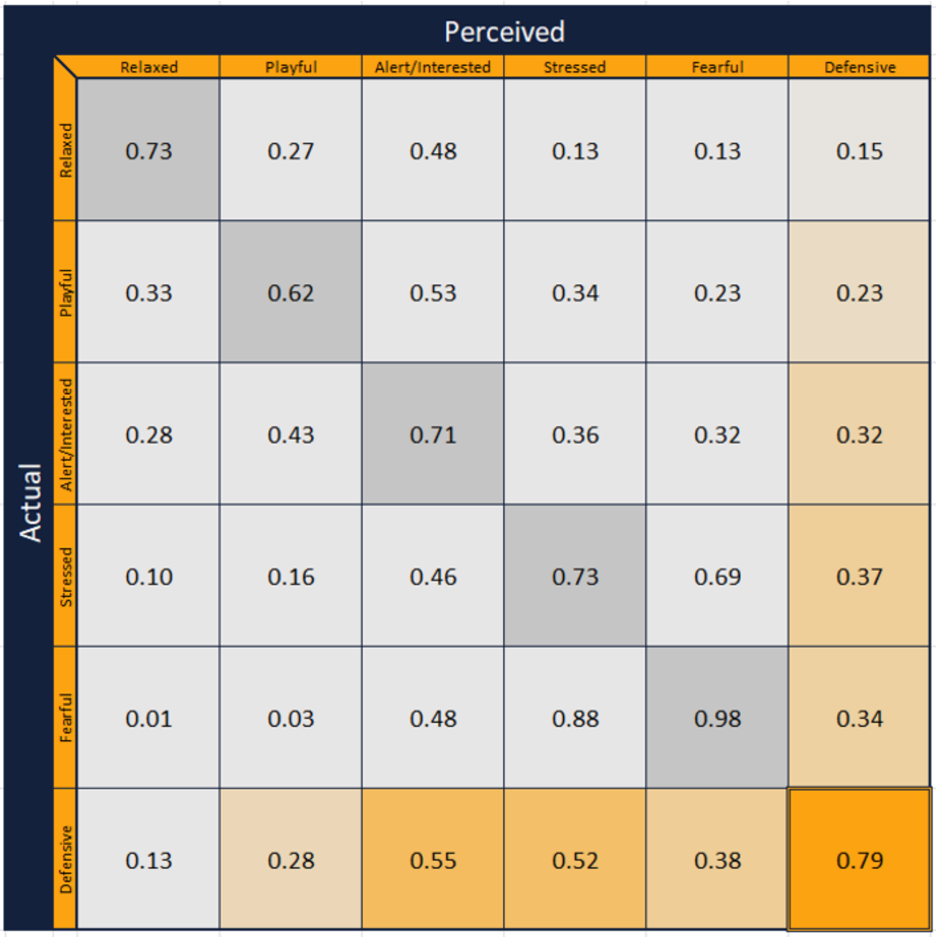
Figure E.4: Test Results for the emotion tests performed in [7] highlighting the stressed emotion



Perceived

Actual	Relaxed	Playful	Alert/Interested	Stressed	Fearful	Defensive
Relaxed	0.73	0.27	0.48	0.13	0.13	0.15
Playful	0.33	0.62	0.53	0.34	0.23	0.23
Alert/Interested	0.28	0.43	0.71	0.36	0.32	0.32
Stressed	0.10	0.16	0.46	0.73	0.69	0.37
Fearful	0.01	0.03	0.48	0.88	0.98	0.34
Defensive	0.13	0.28	0.55	0.52	0.38	0.79

Figure E.5: Test Results for the emotion tests performed in [7] highlighting the fearful emotion



Actual	Perceived					
	Relaxed	Playful	Alert/Interested	Stressed	Fearful	Defensive
Relaxed	0.73	0.27	0.48	0.13	0.13	0.15
Playful	0.33	0.62	0.53	0.34	0.23	0.23
Alert/Interested	0.28	0.43	0.71	0.36	0.32	0.32
Stressed	0.10	0.16	0.46	0.73	0.69	0.37
Fearful	0.01	0.03	0.48	0.88	0.98	0.34
Defensive	0.13	0.28	0.55	0.52	0.38	0.79

Figure E.6: Test Results for the emotion tests performed in [7] highlighting the defensive emotion