UNIVERSITY OF TSUKUBA

DOCTORAL THESIS

Effects of Non-Verbal Emotion-Like Feedback on the Interaction Between People and a Single-Eyed Spherical Robot

---

Author:

Diego Eiji ONCHI SUGUIMITZU

Supervisor:

PhD. Seung Hee LEE

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in

Kansei Information

Graduate School of Comprehensive Human Sciences

Doctoral Program in Kansei, Behavioral, and Brain Sciences

2022

"Nothing has such power to broaden the mind as the ability to investigate systematically and truly all that comes under thy observation in life."

Marcus Aurelius, *Meditations (c. 161–180 CE), Chapter II.*

Abstract

Diego Eiji

ONCHI SUGUIMITZU

Effects of Non-Verbal Emotion-Like Feedback on the Interaction Between People and a Single-Eyed Spherical Robot

The dynamic between humans and robots is becoming an important part of our daily lives and many researchers in the area of Human-Robot Interaction are creating Social Robots that are pleasant to use and interact with. To that end, the goal of this research was to study how different elements of non-verbal communication between a robot and a human affect the interaction experience on an emotional level when doing collaborative tasks. Moreover, this work aims to develop several open source tools that might aid researchers and developers in future endeavors in Human-Robot Interaction research.

Several improvements were made to the robot developed by Onchi and Lee (2019) both in hardware and software, and two main investigations into non-verbal emotional expressions and human-robot interaction were conducted.

A within-subjects experiment with 8 design students (4F / median age: 20–32) from the University of Tsukuba was conducted between the robot showing only motion feedback (control) and motion with LED feedback (LED).

The results indicated that adding an LED changes the emotional impression of the robot. It is possible to use this stimulus to modulate the valence and arousal of the emotion being expressed and create a tool to dynamically change the emotion of a non-humanoid robot using movement, light, and animations. Having a variable output other than just movement may increase the animacy perceived, thus creating a more engaging experience.

The second part of the study focused on how the robot compared to other interacting agents, like humans or computers, using a cooperative game. A within-subjects experiment with 24 participants (12F / $\text{median}_{\text{age}}$ = 25–29) from the University of Tsukuba was conducted. The overall results evidenced that people prefer to interact with physical beings that can express some type of feedback during the interaction. The performance of the robot was comparatively similar to when people interacted with another person, while the interaction with the computer was categorized as emotionally neutral and no social bonding happened.

Finally, this research generated not only insights in the area of Human-Robot Interaction, but also useful tools available to the scientific and programming community. In particular, six open source modules optimized for the Raspberry Pi platform were created.

keywords: robot, interaction, minimalism, feedback, emotion

Acknowledgements

I would like to thank the following people, without whom I would not have been able to complete this research, and without whom I would not have made it through my doctoral degree.

First, I would like to thank the professors of the University of Tsukuba, especially to my supervisor Dr Seung Hee Lee, whose insight, knowledge, and passion regarding *Kansei* steered me through this research. Also, I would like to thank my colleagues at Lee Laboratory for helping me during the research with their valuable feedback. And special thanks to Ban, who went out of his way to help me in times of need!

I would also like to thank Natanya C., who help me during my experiments and took great videos and pictures of the spherical robot. Without her, the design of the animations would not have been as polished!

And my biggest thanks to my family for all the support you have shown me through this research, a hard work of three years and more away from them. Without them, it would have been hard to continue in this path and see the culmination of my studies. Thank you so much!

Contents

List of Figures

List of Tables

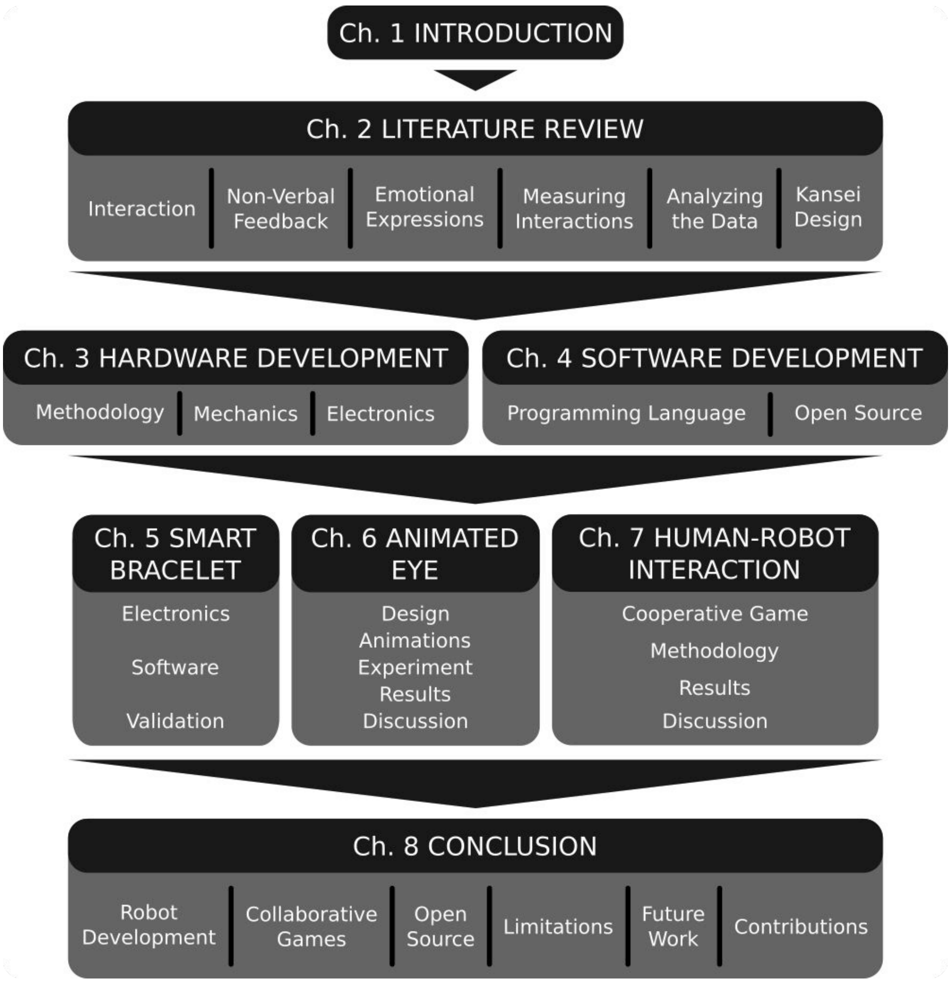
List of Abbreviations

ADC	**A**nalog-**D**igital **C**onverter
EDA	**E**lecto**d**ermal **A**ctivity
GSR	**G**alvanic **S**kin **R**esponse
I^{2}C	**I**nter-**I**ntegrated **C**ircuit
RoSAS	**Ro**botic **S**ocial **A**ttributes **S**cale
SAM	**S**elf-**A**ssessment **M**anikin
SPI	**S**erial **P**eripheral **I**nterface
WAI	**W**orking **A**lliance **I**nventory

Dedicated to my family, who
supported me even at a
distance.

Chapter 1

Introduction



The ***beginning of each chapter*** throughout this thesis presents a ***short overview*** of the topics in the chapter. In addition, the above ***flowchart*** connects the main ideas of the research and is used as a ***visual guide*** to aid the reader.

1.1 Purpose of the Study

People are social creatures that use collaboration to achieve greater goals. This collaboration is not limited between humans, but inter-species collaboration has been important throughout human history (Orozco and Parker-Starbuck, 2015, p. 19). For example, dogs were used to aid hunters when going after a prey, or horses were used as a transportation medium before the combustion engine was mainstream. With the advent of technology, non-living alternatives have been developed, either as virtual agents (Amazon.com Inc, 2021; Google Nest, 2021) or robots (Rethink Robotics, 2021; Pandey and Gelin, 2018; Boston Dynamics, 2021). Especially for situations where a person might be endangered, robots are beginning to be used.

This dynamic between humans and robots is becoming an important part of our daily lives. It has caught the interest of many researchers in the area of Human-Robot Interaction to create social robots that are pleasant to use and interact with. To that end, the goal of this research is to ***study how different elements of non-verbal communication between a robot and a human affect the interaction experience on an emotional level when doing collaborative tasks***. Moreover, this work ***aims to develop several open source tools*** that might aid researchers and developers in future endeavors in Human-Robot Interaction research.

Research questions Related to the above matters, the present study focuses on the following main research questions:

- *"Is it possible to use light animations to enhance emotional expression?"*
- *"Will people interact with an emotional robot on the same level as interacting with a person?"*
- *"Is feedback essential when doing collaborative tasks?"*

To answer those questions, the following research structure was designed:

1. Study the minimal means to express non-verbal emotion-like expressions in a spherical robot (Onchi and Lee, 2019).
2. Study if light can be used as an additional non-verbal mean to convey emotion-like expressions in the spherical robot and if it makes a significant difference (Chapter 6).
3. Study the interaction experience with the spherical robot using a cooperative game (Chapter 7).
4. Finally, draw conclusions on how to use these insights for the design of interactive non-humanoid robots.

1.2 Significance of the Study

This study is motivated by research in Social Robotics aimed to create intuitive robots (Breazeal, 2002; Thomaz and Breazeal, 2008; Chernova and Thomaz, 2014; Onchi and Lee, 2019). In particular, it follows the steps of Onchi and Lee (2019) to create a non-humanoid minimalistic robot that can express emotion-like feedback to increase the interaction experience of people. The design approach and research considerations follow a ***Kansei Design*** methodology, an approach introduced in Section 2.8.

The author of this research hopes that the results of this work serve as a foundation to create better social robots that people feel comfortable interacting with. Furthermore, the author looks forward to people using the open source tools developed during this research and encourages future collaborators to keep improving on them.

1.3 Structure of this Thesis

Each chapter begins with a visual schematic of the thesis and a short description to guide the reader throughout the document. The current thesis follows the structure presented below:

- **Chapter 2: Literature Review** presents the context of this research, as well as the theoretical background in which the development and validation of the robot are based. It introduces the ***state of the art*** in collaborative social robots and emotional expression. At the end of this chapter, the details of the measuring techniques used in this thesis are presented.
- **Chapter 3: Hardware Development** presents the ***physical design*** considerations of the robot, and highlights the improvements done over the initial robot developed by Onchi and Lee (2019). Then, the ***electronic and mechanical design*** of the robot is explained.
- **Chapter 4: Software Development** presents the ***software design*** of the robot's program and all the ***open source tools*** developed to make it possible. This chapter also includes an explanation of each module and their possible use outside this thesis.
- **Chapter 5: Smart Bracelet** presents the development of a custom made ***smart bracelet*** to measure the physiological information of participants, as well as the open source tools created for this device.
- **Chapter 6: Animated Eye** presents the ***design process*** of the animated eye of the robot, as well as its ***validation test***. This chapter includes the ***reason behind each animation*** created and concludes with the ***results*** collected in the study.

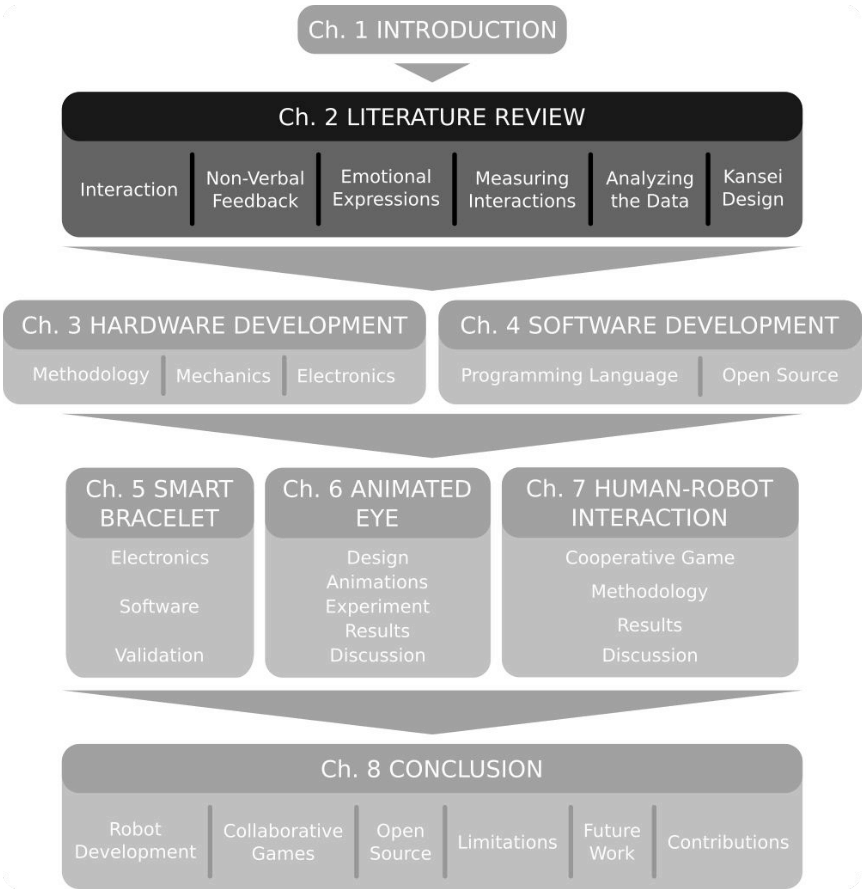
- **Chapter 7: Human-Robot Interaction** presents the design of the ***collaborative game*** use to test the level of trust that a person has depending on their interacting partner. The ***methodology*** and ***results*** are included. The chapter ends with the information collected during the interview with the participants.
- **Chapter 8: Conclusions** provides the ***discussion*** of the research as a whole, as well as focusing on the possible interpretations and implications of each study conducted. It concludes with the ***limitations*** of the study, its ***contributions***, and ***future work***.
- **Appendix** is a collection of additional material relevant to this thesis including, but not limited to, surveys, source code, and relevant documents.

1.4 Ethical Considerations

This research was conducted with the approval of the Research Ethics Committee of Art and Design, University of Tsukuba (Appendix B.1). All participants took part in the research voluntarily after being informed of the details of the research in their preferred language (Appendix B.2 for Japanese and Appendix B.3 for English) and signing proper ***Agreement Form*** (Appendix B.4 for Japanese and Appendix B.5 for English).

Chapter 2

Literature Review



This chapters presents the context of this thesis as well as the necessary literature review to understand the concepts researched. It starts with the ***state of the art on human-robot interactions***, followed by ways to ***express emotions in robots***. An introduction to several ***measuring instruments*** is presented, and a definition of ***Kansei Design*** is presented at the end.

2.1 Interaction

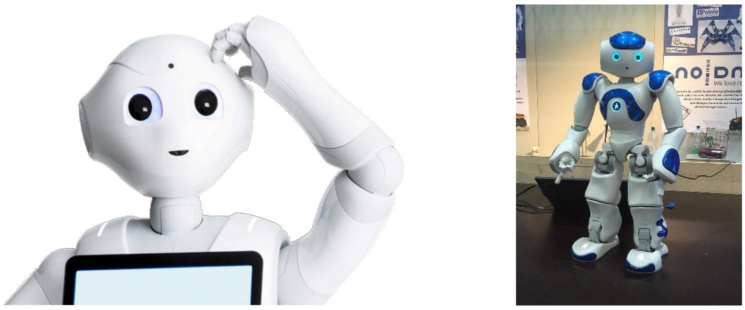
2.1.1 Human Interaction

People are social creatures that interact with their environment and attempt to give meaning to the information perceived. When people socialize with others, there is a conscious and unconscious effort to understand the intention behind the actions of the other party (Baldwin and Baird, 2001). People construct an idea of a person based on the motivations and goals that they can infer, either from experience or memory (Goldman, Graesser, and Broek, 1999).

2.1.2 Human-Robot Interaction

This social interaction not only happens between humans. We are able to interact with non-living agents (e.g. robots) or other species (e.g. pets) as well. In this sense, the concept of ***animacy*** is important in Human-Robot Interaction. ***Animacy***[1] is defined as the *"state of being alive and animate"* (Collins English Dictionary - Complete and Unabridged, 2014). Animacy might be good feature to evaluate if people consider an artificial agent to be *"alive"*, but seems to be difficult to quantify (Bartneck et al., 2009b). Some elements, like giving a physical body to an artificial agent, can increase the level of animacy of robots (Scholl and Tremoulet, 2000). In this regard, Bartneck et al. (2009a) have shown that the physical shape of a robot can influence the perceived intelligence and animacy of the agent.

[1] Animacy is also used in Linguistic to define the semantic feature of languages to express how alive something is.



(A) Pepper (B) NAO

FIGURE 2.1: Example of humanoid robots.
Source: (A) SoftBank Robotics (2021), (B) ubahnverleih (2016)

FIGURE 2.2: Spot from Boston Dynamics
Source: Boston Dynamics (2021)

Social robots that work with people can take a myriad of forms. For example, ***Pepper*** (Pandey and Gelin, 2018), shown in Figure 2.1a, is a humanoid robot from by SoftBank Robotics that is used in several areas including healthcare, education, entertainment, among others. Another representative robot is ***NAO*** (Gouaillier et al., 2009), a robot developed by the same company which is widely used in scientific research (Tapus et al., 2012; Han et al., 2012; Jokinen and Wilcock, 2013; Andreasson et al., 2017). As shown in Figure 2.1b, this robot has a full humanoid build.

On the other hand, non-humanoid robots are also used for tasks that require collaboration with people. For example, ***Spot*** (Boston Dynamics, 2021), presented in Figure 2.2, is a four-legged robot developed by Boston Dynamics that resembles a dog. This robot is used in construction or mining to go to places that would otherwise be dangerous for people.

Apart from the physical shape of a robot, Holroyd et al. (2011) identified four social connection events that can enhance Human-Robot Interaction: ***directed gaze***, which indicates where the robot is looking at; ***mutual facial gaze***, in which both parties look at each other when interacting; ***adjacency pair***, which refers to the minimal overlap between the communication between the parties; and ***backchannel***, which are brief behavioral cues done by the listener.

2.2 Non-Verbal Feedback

Feedback is an essential part of communication in human beings. Without feedback, important information might be lost and affect interactions. Feedback provides a second system of communication that can regulate the subtleties of communication, but this system must be shared by all interacting parties to be understood (Bateson, 1975). This second system is not limited by words, but relies on other cues to indicate pacing and roles during communication.

In other words, ***feedback*** is not just bounded to verbal cues: human communication is aided by non-linguistic elements, like gestures or sounds, that provide that layer of information (Payrato, 2009). They can be used conscious or unconsciously to express emotional states or affective signals (Krauss, Chen, and Chawla, 1996).

2.2.1 Social Eye Gaze Feedback

Social eye gaze in human-robot interactions is an important tool to express non-verbal information (Admoni and Scassellati, 2017). This behavior has been defined as having a dual function (Cañigueral and Hamilton, 2019): one to perceive information, like deictic gaze (Johnson, Ok, and Luo, 2007) or emotional states (Baron-Cohen, Wheelwright, and Jolliffe, 1997); and the other to signal information, such as gaze cueing (Kuhn, Tatler, and Cole, 2009) or floor management (Kendon, 1967) at the same time. Another important aspect of gaze during communication is gaze aversion, a behavior that conveys three main functions: cognitive, intimacy modulation, and floor management (Andrist et al., 2014). This attentional focus can be simulated by having a consistent motion, regardless of the physical shape (Johnson and Ma, 2005). These are important behaviors during human-to-human communication as well as human-to-robot interactions, because people naturally tend to look for those behavioral cues by focusing on the eyes and mouth of the speaker (Vatikiotis-Bateson et al., 1998) to collect information on the mental state and attentional focus of the speaking part (Langton, Watt, and Bruce, 2000).

2.2.2 Blinking Information

Blinking is a natural behavior in human interactions, the average blinking rate of a resting person was measured to be 17 blinks per minute, while the rate increases to 26 when engaging in conversation (Bentivoglio et al., 1997). Additionally, the length of the blink can be modulated to convey non-verbal information during communication (Hömke, Holler, and Levinson, 2018). This visual feedback can help in establishing proper turn-taking (Levinson, 2016). Blinking patterns have been used in robots to create a smoother conversational interaction with people (Funakoshi et al., 2008). Therefore, it is possible to use blinking patterns in robots to increase their animacy.

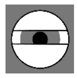
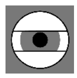
2.3 Emotional Expressions

2.3.1 Universally Recognized Emotions

Ekman, Friesen, and Ellsworth (1972) introduced seven universally recognized facial emotions. These emotions can be defined by how different parts of the human face move and look. While those emotions can be understood regardless of culture, their boundaries are not clearly defined. It is possible to blend several elements of the face to show mixed emotions (Ekman and Friesen, 2003).

Later research in neuroscience narrowed down these universally recognized emotions to four: ***happy***, ***sad***, ***fear/surprise***, ***disgust/anger*** (Jack, Garrod, and Schyns, 2014). By abstracting these movements and applying them in a non-humanoid robot, it was found that high-paced upward motions convey positive emotions while low-paced downward movements express negative emotions (Onchi and Lee, 2019). In other words, direction and speed are enough elements to express basic emotion-like features on a spherical robot. An additional visual way to display emotions is with color. Brightness and saturation in LED displays were shown to be positively correlated with emotional arousal (Wilms and Oberfeld, 2018). Blue hues were correlated with sadness (Terada, Yamauchi, and Ito, 2012). However, the emotional meaning of color depends on context and can be used to express positive and negative emotions (Kaya and Epps, 2004).

TABLE 2.1: Shape of eyelids to express emotions: *"Table 1: Set of emotions based on the physical characteristics of human beings."*

Emotion	Description	Picture	Avatar
Neutral	Upper eyelid touches the iris, lower eyelid is relaxed.		
Happiness	Cheeks raise, pushing the lower eyelid. Upper eyelid can be raised.		
Surprise	Eyes wide open. Sclera fully visible.		
Sadness	Eyes slightly squinted, upper eyelid drops due to the brows.		
Fear	Eyes open and tense. Lower eyelid contracted.		
Disgust	Eyes squinted due to the wrinkle of the nose.		
Anger	Eyes focused and wide open. Upper eyelid seems lower due to the brow.		

Source: Onchi, Saakes, and Lee (2020)

2.3.2 Emotional Eyelids

By focusing on just the eyes, Onchi, Saakes, and Lee (2020) studied if it was possible to express different emotions using rigid eyelids on a single-eyed 2D avatar (Table 2.1). Their results showed that people understand the difference in emotional states just by changing the positions of the eyelids. This was part of a study into minimalist elements to express emotions in consumer electronics to be able to add an emotional layer to the experience.

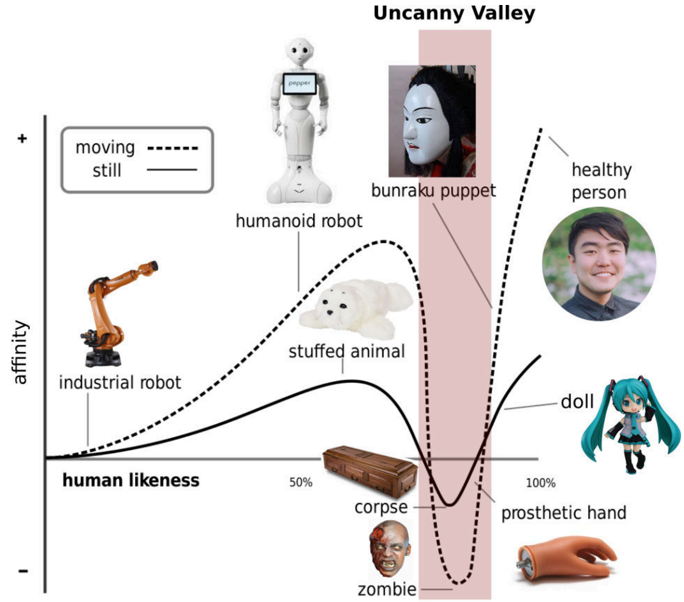


FIGURE 2.3: The Uncanny Valley: *"Figure 2. The presence of movement steepens the slopes of the uncanny valley. The arrow's path represents the sudden death of a healthy person."*
Source: Adapted from The Uncanny Valley (Mori, MacDorman, and Kageki, 2012, p. 99)

2.3.3 Emotional Expressions in Robots

Researching how to express and simulate emotions in robots has been of interest for the area of Human-Robot Interaction. In this regard, one approach to design robotic emotions is to closely recreate human expressions (Breazeal, 2002; Kishi et al., 2012). However, it is possible that these humanoid robots fall into the ***Uncanny Valley***, the point where people feel aversion toward a human-like robot (Mori, MacDorman, and Kageki, 2012).

2.3.3.1 Uncanny Valley

The closest reference people have to recreate emotional expressions are human beings. Therefore, it is not uncommon to imitate the physical characteristics of people with the goal to create emotional agents. However, the moment this physicality approaches a real human, but fails to complete grasp

FIGURE 2.4: Sophia from Hanson Robotics
Source: ITU Pictures (2018)

its essence, is the point where people start feeling uncomfortable with that agent. In Human-Robot Interaction, this inflection point is known as the ***Uncanny Valley***, the moment where *"[. . .] a person's response to a human-like robot would abruptly shift from empathy to revulsion as it approached, but failed to attain, a lifelike appearance"* (Mori, MacDorman, and Kageki, 2012).

To address this concern, Sumioka et al. (2013) proposed that a minimalistic design could avoid the ***Uncanny Valley*** and be used to create robots that are socially accepted. Studies using this concept have shown that minimal motions, colors, sounds, or a combination of them can be enough to express some basic emotions (Terada, Takeuchi, and Ito, 2013; Löffler, Schmidt, and Tscharn, 2018; Onchi and Lee, 2019). Thus, this research aims to continue exploring different minimalistic ways to express non-verbal emotional feedback.

FIGURE 2.5: Robot developed by Onchi and Lee (2019) showing different movements as non-verbal feedback.
Source: Onchi and Lee (2019)

2.3.4 Expressing Emotions as Movement

In the early work of Onchi and Lee (2019), the single-eyed spherical robot that could move its attentional focus, shown in Figure 2.5, was developed with the objective to improve training interactions. As part of their research, they studied how movement affected the emotional impression of people.

By controlling the ***speed*** and ***direction*** of the motion, different non-verbal emotion-like feedback were designed. As presented in Figure 2.6, their results evidenced that non-verbal movement feedback cannot be classified into an specific emotional state, but they can be used to convey emotional information about the robot within the emotional dimensions proposed by Russell (1980). In general, a ***high-paced upwards*** movement was associated with a ***positive*** emotional state, while a ***low-paced downwards*** movement expressed a ***negative*** emotional state.

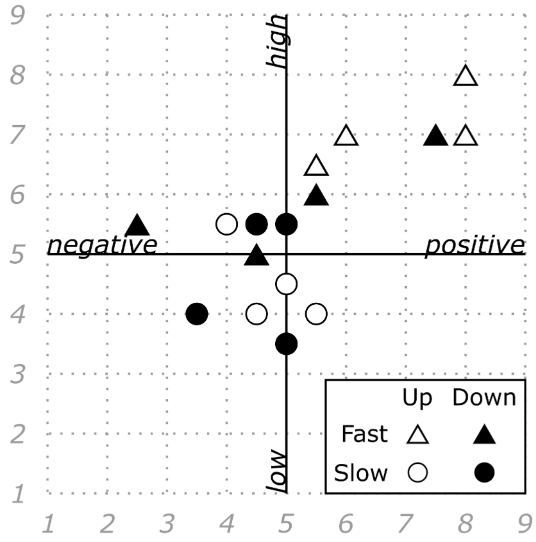


FIGURE 2.6: Correlation between movement speed and direction with the emotional impression.
Source: Onchi and Lee (2019)

2.4 Aim of this Study

Looking at the development of commercial assistants, it is possible to see that companies (Amazon.com Inc, 2021; Google Nest, 2021) tend to focus on creating abstract representations of their agents. Research in the area of Human-Robot Interaction has suggested that having a physical body might increase the animacy of the agent (Scholl and Tremoulet, 2000). In this regard, this section presented robots that follow humanoid shapes or have non-humanoid bodies that interact with people and are used for Social Robotic research. While it is technologically possible to create complex robots that imitate human expressions, everyday interactions may benefit by just adding simples gestures instead of creating a complex machine. This might make mass production easier and improve the overall acceptance of robots in the future. In this sense, more research is needed to find what are the minimal means to create engaging interactions.

On the other hand, studies have shown that gaze and eyes are an important source of non-verbal information (Andrist et al., 2014; Admoni and Scassellati, 2017; Hömke, Holler, and Levinson, 2018). Therefore, it bears the questions if we can use these elements in a minimalistic way to create simple movements that can convey information. The research done by Onchi and Lee (2019) evidenced that a minimalistic robot could express emotion-like feedback using only movement. However, the information conveyed was not complete clear and there was room for improvement in how engaging an interaction could be. In this case, it was necessary to test other means of non-verbal expression, like light animations.

For that reason, the current thesis focuses on researching ***what are the minimal means to create engagement*** in Human-Robot Interaction experiences by trying to answer the following research questions:

- *"Is it possible to use light animations to enhance emotional expression?"*
- *"Will people interact with an emotional robot on the same level as interacting with a person?"*
- *"Is feedback essential when doing collaborative tasks?"*

2.5 Measuring Emotions

Russell and Bullock (1986) suggested that the categorization of emotions using verbal concepts should be considered as fuzzy sets. In this regard, rather than fixed categories, emotions can be systematically placed along three orthogonal axes (valence, arousal, and dominance), and measured using non-verbal tools (Bradley and Lang, 1994).

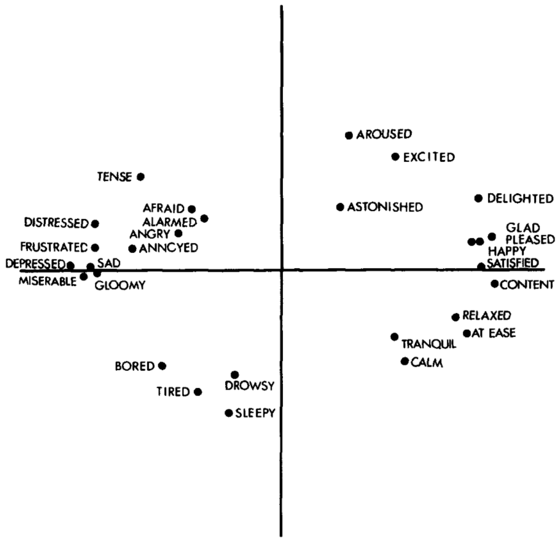


FIGURE 2.7: Circumplex Model of Affection: *"Figure 5. Regression weights for 28 affect words as a function of pleasure-displeasure (horizontal axis) and degree of arousal (vertical axis)."*
Source: A Circumplex Model of Affect (Russell, 1980, p. 1173)

2.5.1 Russell's Circumplex Model of Affection

The ***Circumplex Model of Affection*** initially presented by Russell (1980), proposes that emotion can be defined as a two-dimensional state of ***valence*** and ***arousal***. ***Valence*** refers to how positive or negative an emotion is. On the other hand, ***arousal*** refers to the level of emotional energy or activation. It is important to note that ***emotional arousal*** is not the same as ***emotional intensity***. While it is easy to confuse intense emotional moments as high arousal (e.g. feeling excited about winning the lottery), intense emotional states could happen with low arousal (e.g. intense depression is a state of low arousal). Therefore, arousal and intensity are considered as separate concepts when scoring emotions.

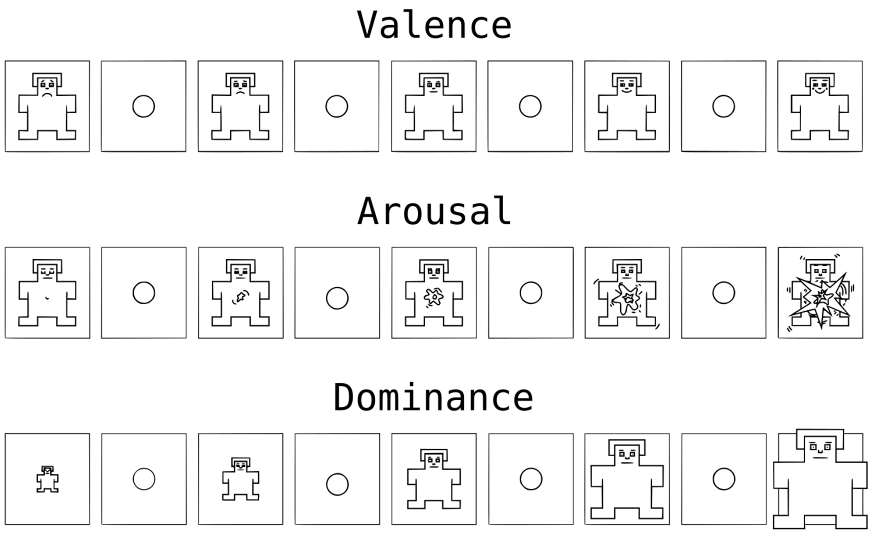


FIGURE 2.8: Self-Assessment Manikin. Each dimension consists of **five figures** and **four in-between values**, for a total of nine points in the scale.
Source: adapted and vectorized from Betella and Vershure (2016)

2.5.2 Self-Assessment Manikin (SAM)

The following thesis uses the ***Self-Assessment Manikin (SAM)***, a 9-points scale that measures emotion using pictographs (Figure 2.8). This tool evaluates the emotional impression of people using three dimensions: ***valence***, ***arousal***, and ***dominance*** (Bradley and Lang, 1994; Bynion and Feldner, 2017; Geethanjali et al., 2017). This tool has the advantage to be language agnostic (Bradley and Lang, 1994, p. 50) and does not rely on pre-defined labels to categorize an emotion. Its simplicity and dimensionality has made this a popular tool among studies in Affective Computing (Morris, Dontcheva, and Gerber, 2012; Onchi and Lee, 2019; Jaeger et al., 2019).

The specific figures used were vectorized from the pictures presented by Betella and Verschure (2016), which are based on Bradley and Lang (1994). An English (Appendix A.1) and Japanese (Appendix A.2) version of the survey were prepared and administered using Google Forms.

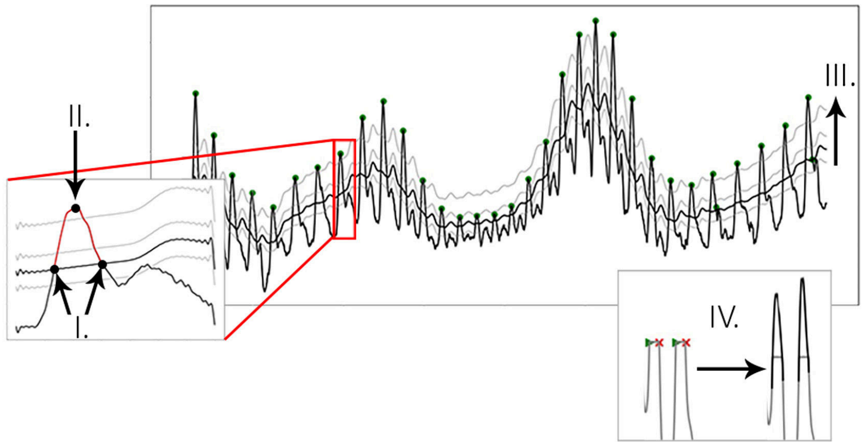


FIGURE 2.9: Sample heart-rate signal: *"Fig. 2. Process of peak extraction: a moving average is used as an intersection threshold (I). Candidate peaks are marked at the maximum between intersections (II), with optional spline interpolation available to improve position accuracy. The moving average is raised stepwise (III). IV. shows the detection of the onset and end of clipping, and the result after interpolating the clipping segment."*

Source: Gent et al. (2019).

2.5.3 Physiological Data

Recording physiological data can provide valuable information that could be hard to detect with surveys or interviews. Especially for unconscious reactions, physiological data might provide more insights into the emotional reactions of people.

2.5.3.1 Heart-Rate

Heart-rate is the measurement of how many cardiac cycles are performed by the heart in the span of one minute. The resting heart-rate range of a healthy person has been defined as 60 to 100 beats per minute (Kossmann, 1953), although Spodick et al. (1992) have lowered this range to 50 to 90 beats per minute (Spodick et al., 1992), while Graybiel et al. (1944) stated that young people have a normal range of 38 to 110 beats per minute.

Heart-rate variability is related to the regulation of emotional responses, which is connected to the sympathetic and parasympathetic reactions of the body (Appelhans and Luecken, 2006). Azarbarzin et al. (2014) have shown that there is a positive correlation between the change in heart-rate and the level of arousal. In particular, when people feel more aroused from a given event, their heart-rate variation increases. Nevertheless, heart-rate varies from person to person (Mathias and Stanford, 2003), which means it is more reliable to compare the changes in heart-rate rather than its actual value when analyzing a group of people. This thesis uses the analysis method provided by Gent et al. (2019) with the `HeartPy` Python module, which can filter and find heart beats in noisy data (Figure 2.9).

2.5.3.2 Electrodermal Activity (EDA)

Electrodermal Activity (EDA), also known as skin conductance, galvanic skin response, or electrodermal response, refers to the change in the electrical properties of the human skin given some stimuli (Boucsein, 2012, p. 2). EDA is mostly measured as the electrical resistance between two contact points in the skin, which is altered by sweat secretions (Boucsein, 2012, p. 104). This is important because sweat regulation is partially managed by the ***Autonomous Nervous System (ANS)*** (Hu et al., 2018), which is also connected to emotional states in a person (Levenson, 2006). In particular, EDA seems to be correlated with the ***emotional arousal*** of the individual.

EDA is usually analyzed by extracting the phasic[2] and tonic[3] components of the signal (Boucsein, 2012, p. 150). Based on the Nyquist-Shannon sampling theorem (Shannon, 1949), to correctly reconstruct both phasic and tonic

[2]Fast physiological response to a stimulus.
[3]Slow physiological change measured over a period of time.

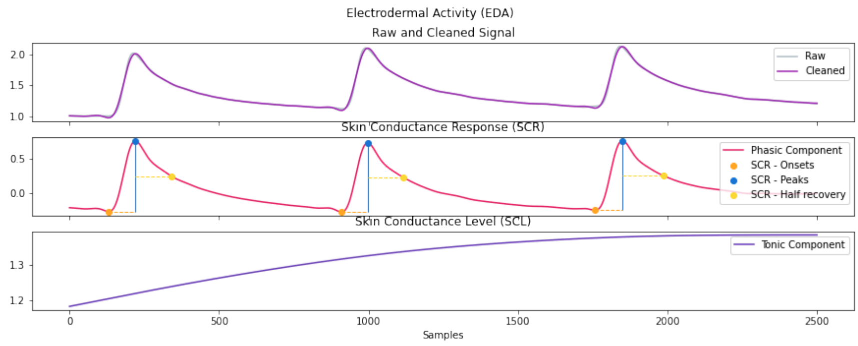


FIGURE 2.10: Sample EDA signal: plotting EDA signal using the `eda_plot()` function from `NeuroKit2`.
Source: Makowski et al. (2021).

frequencies of EDA, a sample rate between 200 to 400 Hz is required at minimum (Figner and Murphy, 2011).

By separating the signal in this two components, it is possible to get the ***Skin Conductance Level (SCL)*** and the ***Skin Conductance Response (SCR)***. The SCR is a fast fluctuation on the skin conductance related to the somatic response of the body (Christopoulos, Uy, and Yap, 2016), which has an onset latency between 1 to 3 seconds, while SCL is the overall conductance over a long period of time (Figner and Murphy, 2011). In other words, SCR gives us the immediate response to a stimulus, while SCL gives us the change over the whole interaction.

In this regard, several automated algorithms have been developed to analyze this signal. For example, Greco et al. (2016) proposed a convex optimization approach to process EDA. On the other hand, Akash et al. (2018) have shown that there is a relationship between trust in intelligent machines and EDA. They developed a classification algorithm that aims to predict the level of trust based on the analysis of the phasic and tonic components of EDA. This thesis uses the analysis method provided by Makowski et al. (2021) with the `NeuroKit2` Python module, which can filter and split EDA signals into SCL and SCR (Figure 2.10).

2.6 Measuring Interactions

2.6.1 Working Alliance Inventory Short Revised (WAI-SR)

The ***Working Alliance Inventory*** (WAI) is a survey that measures the degree of collaboration between two people, usually a therapist and a patient, developed by Horvath and Greenberg (1989). This questionnaire scores the ***working alliance***[4] of an interaction based on the three dimensions proposed by Bordin (1979): ***task***, ***bond***, and ***goal***. Here, ***task*** is defined as the degree in which the person feels that the tasks done during the interaction are relevant, ***bond*** refers to the personal attachment between both parties, and ***goal*** points to the level of endorsement toward the goal of the activities.

While the original version of this test consisted of 36 items, Tracey and Kokotovic (1989) developed a short version of 12 items that could measure this relationship with the same reliability (Tracey and Kokotovic, 1989; Hatcher and Gillaspy, 2006). Each statement is scored using a 5-point Likert scale for frequency[5]. This test has been widely used in Social Robotics to measure the relationship between the user and the robotic agent (Bickmore and Picard, 2004; Kidd and Breazeal, 2008; Hoffman, 2019; Wilson et al., 2019).

This research implements the version developed by Hatcher and Gillaspy (2006). The list of items in English and Japanese are presented in Tables 2.2 and 2.3 respectively. The name of the interacting partner is mentally replaced at the underscore space of each statement. An English (Appendix A.5) and Japanese (Appendix A.6) version of the survey were prepared based on the official surveys (Society for Psychotherapy Research, 2016), and administered using Google Forms.

[4]**Working alliance**: change-inducing relationship (Horvath and Greenberg, 1989, p. 224).
[5]1. *Never* | 2. *Rarely* | 3. *Sometimes* | 4. *Often* | 5. *Always*

TABLE 2.2: Working Alliance Inventory Items (English Version)

Item	Question	Scale
1	_____ and I agree about the steps to be taken to improve his/her performance.	*task*
2	I am confident that what _____ was doing during training will help him/her perform better.	*task*
3	I believe _____ likes me.	*bond*
*4	I have doubts about what we are trying to accomplish in task.	*goal*
5	I am confident in my ability to help _____.	*bond*
6	We are working toward mutually agreed upon goals.	*goal*
7	I enjoy working with _____.	*bond*
8	_____ and I have a common perception of him/her goals.	*task*
9	_____ and I have built a mutual trust.	*bond*
*10	_____ and I have different ideas on what his/her real difficulties are.	*goal*
11	We agree about the kind of changes that would be good for _____.	*goal*
12	I think that _____ believes that the way we are working is useful.	*task*

*(*reverse item)*

TABLE 2.3: Working Alliance Inventory Items (日本語版)

Item	Question	Scale
1	____と私は、____の状況を改善するためにはどのようなステップを踏めばいいのか、意見が一致している。	*task*
2	課題の有用性について、____と私の意見が一致している。	*task*
3	____と私は、互いに好感を抱きあっている。	*bond*
*4	____と私の間で、課題から得られるものについての懸念がある。	*goal*
5	私は、私の援助能力を信頼しきっている。	*bond*
6	____と私は、目標を合意したうえで課題を行っている。	*goal*
7	私は____と一緒に課題するのが楽しいと思っている。	*bond*
8	課題における____の目標について、____と私の間で共通の認識がある。	*task*
9	____と私は、互いに信頼し合っている。	*bond*
*10	____と私は、____が抱えている問題についての考えが異なっている。	*goal*
11	____と私は、____がどのように変わればよいか、十分理解を深めることができた。	*goal*
12	私は、課題の方法は____にとって良い方法だと思っている。	*task*

*(*reverse item)*

2.6.2 Robotic Social Attributes Scale (RoSAS)

The ***Robotic Social Attributes Scale (RoSAS)*** is a validated survey developed by Carpinella et al. (2017) that measures how sociable a robot feels to a person. This is an 18-item scale based on the Godspeed Scale (Bartneck et al., 2009b). It measures three dimensions of robotic social perception: ***warmth***, ***competence***, and ***discomfort***. Each adjective is scored using a 5-point Likert scale for association[6]. In this scale, ***warmth*** is measured with the attributes *happy*, *feeling*, *social*, *organic*, *compassionate*, and *emotional*; ***competence*** is scored with the adjectives *capable*, *responsive*, *interactive*, *reliable*, *competent*, and *knowledgeable*; while ***discomfort*** is evaluated with the keywords *scary*, *strange*, *awkward*, *dangerous*, *awful*, and *aggressive*.

RoSAS is used in this research to create a baseline of the initial perception of the participant toward the robot. Then, it is possible to measure if that perception changes after interacting with the robot and analyze how each dimension is affected. The list of items in English and Japanese are presented in Table 2.4. An English (Appendix A.3) and Japanese (Appendix A.4) version of the survey were prepared and administered using Google Forms. The English version was based on the questionnaire published by Carpinella et al. (2017) and the Japanese version was based on the survey used by Noguchi, Kamide, and Tanaka (2018).

[6]1. *Definitely not associated* | 2. *Probably not associated* | 3. *Undecided* | 4. *Probably associated* | 5. *Definitely associated*

[7]*Based on the ranslation by Noguchi, Kamide, and Tanaka (2018, Table 2).*

TABLE 2.4: Robotic Social Attributes Scale Items

Dimension	**English**	**日本語**[7]
Warmth	Happy	幸せな
	Feeling	多感な
	Social	社会的な
	Organic	有機的な
	Compassionate	思いやりのある
	Emotional	情緒的な
Competence	Capable	有能な
	Responsive	敏感な
	Interactive	双方向的な
	Reliable	信頼できる
	Competent	適格な
	Knowledgeable	物知りな
Discomfort	Scary	おそろしい
	Strange	奇妙な
	Awkward	ぎこちない
	Dangerous	危険な
	Awful	ひどい
	Aggressive	攻撃的な

2.6.3 Cooperative Games

Cooperative games are activities where two or more participants work toward a defined goal. In this type of games, rather than competing with each other, the participants try to work together to complete the task at hand (Seif El-Nasr et al., 2010). These types of games have been used to study collaborative behavior in Human-Human Interaction (Hogan, Fisher, and Morrison, 1974; Creighton and Szymkowiak, 2014) and Human-Robot Interaction (Lee and Hwang, 2008; Wainer et al., 2013; Jeri et al., 2017; Paetzel, Perugia, and Castellano, 2020).

This thesis uses an implementation of a collaborative maze solving game in which the information about the maze is divided between two players and both players need to work together to exit the maze. A detailed explanation about the developed game is provided in Section 7.2.

2.7 Analyzing the Data

2.7.1 Linear Mixed-Effects Model (LMM)

Linear Mixed-Effects Models (LMMs) are a type of statistical analysis that takes into account random effects of the data and can accommodate missing information (Magezi, 2015). Compared to the more classical approach of using ***Analysis of Variance (ANOVA)***, LMMs are not limited by assumptions of normality of the data and variance-covariance matrix. Moreover, LMMs are ideal to analyze nested information with crossed groups.

In particular for this thesis, when studying the effects of interacting partners on ***emotional impression***, ***trust***, and ***physiological changes***, it is possible to specify random effects (e.g. participant) using LMM. Furthermore, LMM allows to explicitly partition the variance that is associated with these differences. On the other hand, contrary to ANOVA, missing information caused by unexpected behavior or noisy input when recording physiological information does not affect the analysis. For those reasons, a LMM analysis was most appropriate.

2.8 Kansei Design

The design methodology and research decisions in this study are based on the methodologies used in ***Kansei Design***. ***Kansei*** (感性) is a Japanese word with a complex meaning that is difficult to translate in one word. It might be used in English to refer to *"human sensitivity"*, *"subjectivity"*, *"emotions"*, and *"feelings"* (Lee and Stappers, 1999).

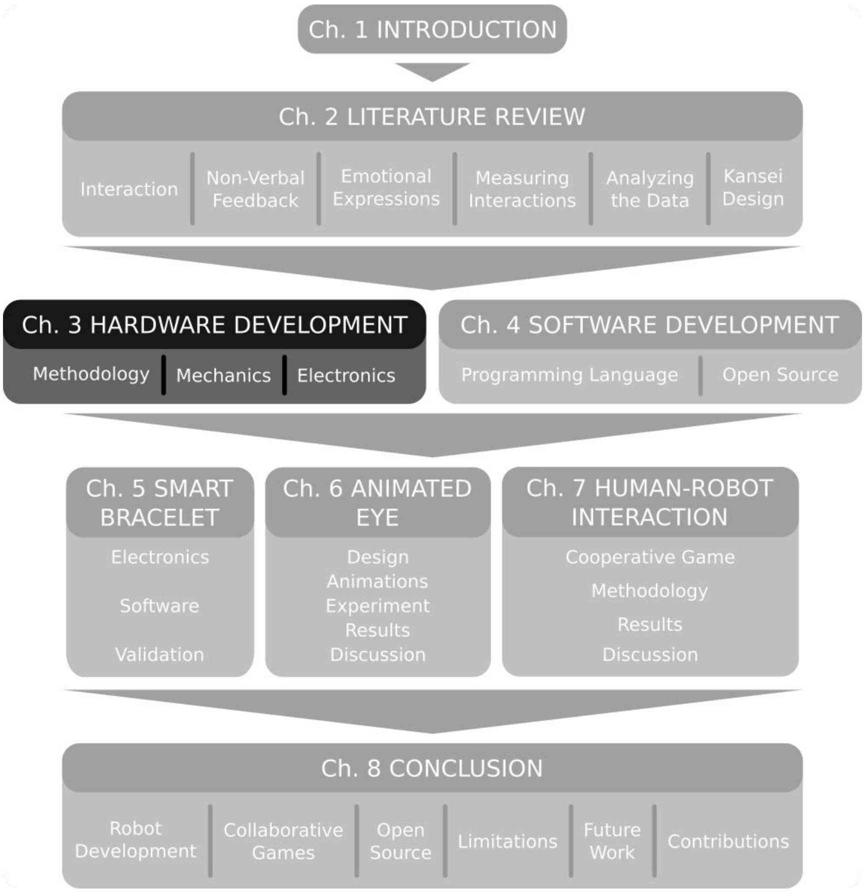
Based on Lee, Harada, and Stappers (2002, p. 213), *"Kansei has been regarded as a totally subjective phenomenon so that anyone in the world has their own* *individual way of absorbing and presenting. [...] But then in the history of product design, the emphasis on mass production caused a disregard for the individual's preferences and feelings.".*

Therefore, ***Kansei Design*** can be regarded as a type of ***user-centered*** design approach with the addition that the ***experience*** of both the designer and user are taken into account when making decisions. In this sense, ***Kansei Design*** place an importance on experiencing the world through physical means (Lee and Stappers, 1999; Lee, Stappers, and Harada, 2000) and using images and abstraction, instead of concrete words or definitions, to create new products (Lee, Kato, and Harada, 1997; Lee, Harada, and Stappers, 2002)

This approach was used by Onchi and Lee (2019) to design the initial prototype of the robot used in this research. This study builds upon their work to improve and further test the Human-Robot Interaction experience with non-humanoid robots.

Chapter 3

Hardware Development



This chapters presents the ***design methodology*** of the spherical robot as well as the ***mechanical*** and ***electronic*** improvements made to the robot. It also includes a detailed explanation of its inner structure.

3.1 Design Methodology

The spherical robot used is based on the robot developed by Onchi and Lee (2019). This robot was designed with three design considerations in mind: ***gaze direction***, ***transparency***, and ***simplicity***. Several design iterations converged into a single-eyed spherical robot that has a solid eye and can move in two-axes. There was no brow present, and the directional focus was coupled with a whole-body movement of the robot. To express more complex shapes without a brow, different arc lengths were displayed with LED rings. Moreover, subtle eye saccades were replaced by bigger movements and body jerks.

3.2 Mechanical Components

The following section details the mechanical components designed for the robot. All parts described below are visually presented in Figure 3.1.

The spherical robot of this study is modeled after the robot developed by Onchi and Lee (2019). The ***outer cover*** ⑦ ⑪ of the robot is made of two acrylic semi-spheres of 170 mm in diameter, with a thickness of 3.5 mm. Each hemisphere was spray-painted with white acrylic paint from the inside. Two pairs of 3D printed ***stumps*** ⑥ ⑫, which house $\phi 6{\times}3$ mm neodymium magnets, were glued to the inside of the ***outer cover*** ⑦ ⑪, and they are used to attach the inside components of the robot with its chassis.

The ***top beam*** ⑨ was remodeled to house the ***Raspberry Pi Zero W*** ⑧ directly on top, secured with four M2.5 screws. Its front has an indentation to slide a ***camera sensor*** and a place to glue two $\phi 13{\times}3$ mm neodymium magnets. This ensures that the computer and the camera move as one unit, eliminating the risk of damaging the connections. It has embossed the shape of the ***servo's horn*** ⑬, which attaches to the motor using two M2 screws. It also has two cable channels on the back to guide the cables inside the robot. Its lateral extremes house $\phi 6 \times 3$ mm neodymium magnets which mate with the ***top stumps*** ⑥.

The components of the ***eye*** are placed on the outside of the ***top cover*** ⑥. All parts are hold in place by the ***eye rim*** ①, which has two $\phi 13 \times 3$ mm neodymium magnets glued that mate with the magnets from the ***top beam*** ⑨. The ***cornea*** ② made out of an acrylic spherical sector of $\phi 100$ mm, the ***iris cover*** ③, the ***iris*** ④, and the ***LED ring*** ⑤ are clamped together between the ***eye rim*** ① and the ***top cover*** ⑦. A detailed explanation of the design considerations of the ***eye*** is presented in Chapter 6.

The ***bottom beam*** ⑩ is slanted 60° and has a hole in the middle aligned with the axle of the ***top servo motor*** ⑬. It is embossed with the shape of the ***bottom servo's horn*** ⑬ which attaches to the motor using two M2 screws. Its lateral extremes house $\phi 6 \times 3$ mm neodymium magnets which mate with the ***bottom stumps*** ⑫, transferring the motion of the ***bottom servo*** ⑬ to the ***bottom cover*** ⑪ of the robot.

The ***bottom servo*** ⑬ is placed inside a hole carved in the ***inner base*** ⑭. This ***inner base*** ⑭ has five $\phi 13 \times 3$ mm neodymium magnets glued with alternating polarity at its bottom. These magnets transfer the rotation of the ***inner base*** ⑭ onto the ***bottom base*** ⑯ through the ***bottom cover*** ⑪. Likewise, the ***bottom base*** ⑯ has five $\phi 13 \times 3$ mm neodymium magnets glued that match the orientation of the magnets from the ***inner base*** ⑭. This outer base is covered with a ***bottom base cover*** ⑮ to reduce friction between the acrylic and the base. This base as a 1.5 mm rubber film attached at the bottom to make sure that the robot does not slide when placed on a flat surface.

The following subsections give detailed information on the improvements done to the robot on each mechanical part.

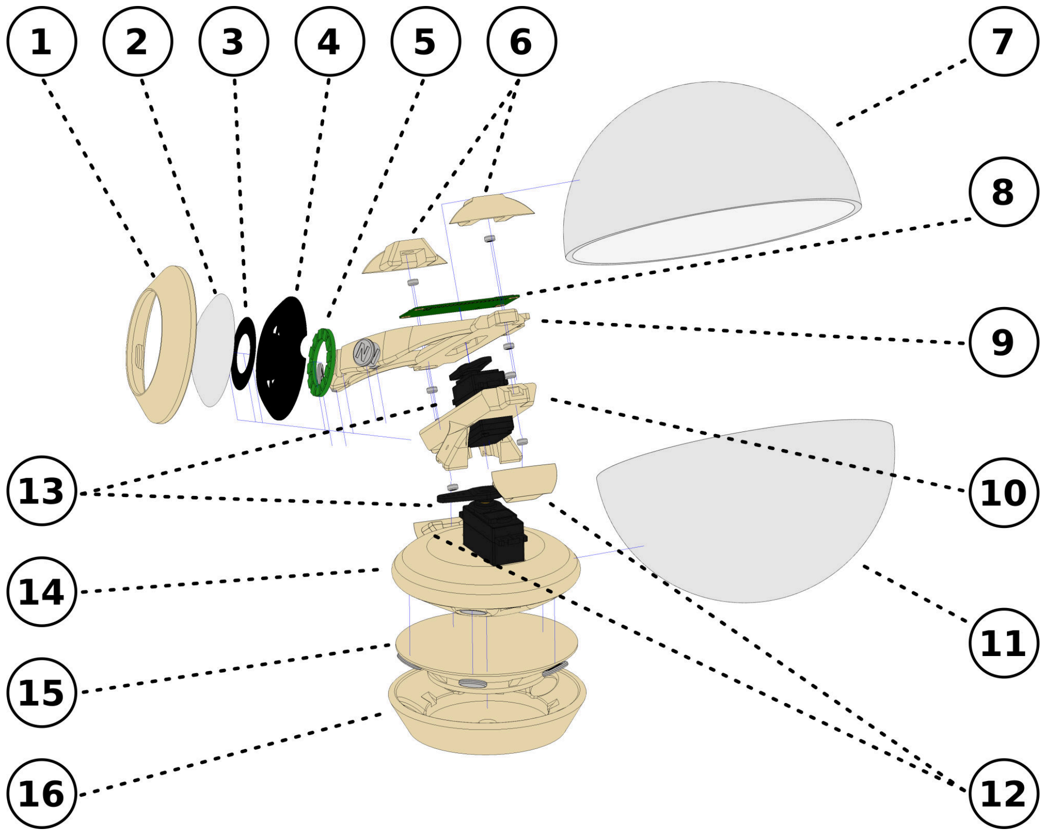


FIGURE 3.1: Exploded view of the robot's components. ① Eye Rim | ② Cornea | ③ Iris Cover | ④ Iris | ⑤ LED Ring | ⑥ Top Stump | ⑦ Top Cover | ⑧ Raspberry Pi Zero W | ⑨ Top Beam | ⑩ Bottom Beam | ⑪ Bottom Cover | ⑫ Bottom Stump | ⑬ Servo Motor | ⑭ Inner Base | ⑮ Bottom Base Cover | ⑯ Bottom Base |

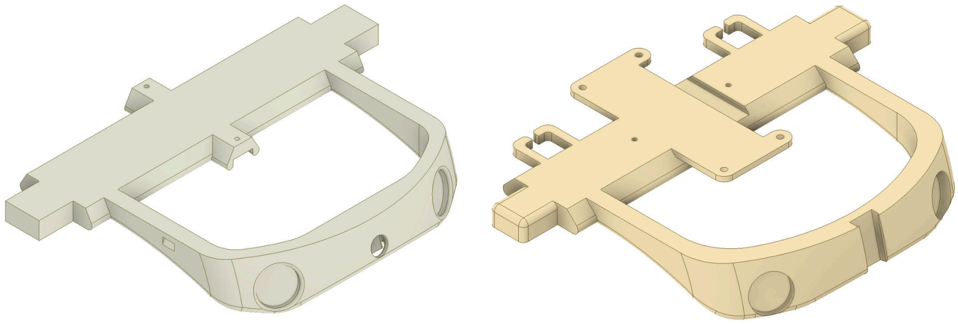
Source: own.

FIGURE 3.2: Original version of the robot with only dynamic motion.

Source: Onchi and Lee (2019).

FIGURE 3.3: Improved version of the robot with more fluid motion and an animated eye.

Source: own.



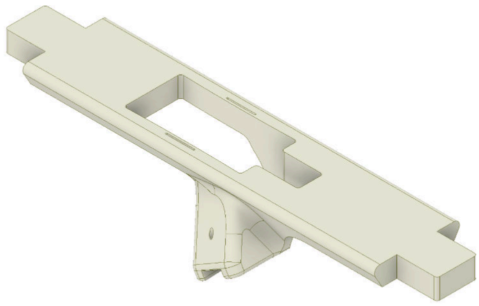
(A) Original Top Beam (B) Improved Top Beam

FIGURE 3.4: Comparison between the original version of the top beam *(A)* and its improved version *(B)*.
Source: own.

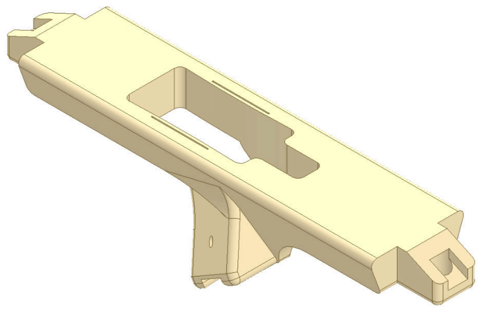
3.2.1 Top Beam

The first big change was the structure of the inner ***top beam***. In the original version (Figure 3.4a), the camera was placed inside a circular chamber which limited the flexibility to fine tune the position of the camera within the robot. Moreover, a separate piece was used to hold the Raspberry Pi Zero W in place, which led to unwanted movement inside the robot.

For the improved version (Figure 3.4b), the footprint of the Raspberry Pi Zero W was placed on top, securing the computer to the upper movement of the robot and ensuring that the cable connecting to the camera would stay static relative the this computer. Furthermore, the camera chamber was replaced by a slit that allowed the camera to be adjusted vertically. This greatly improved the quality of the image acquired by this sensor. On the other hand, two cable guides were placed on the back part of the beam to fix the cables connecting to the servo motors. This avoided inner tangling of the cables which improved the smoothness of the upper movement.



(A) Original Bottom Beam



(B) Improved Bottom Beam

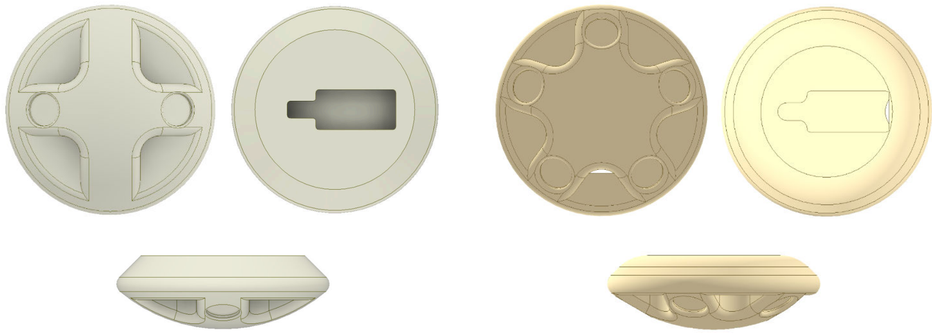
FIGURE 3.5: Comparison between the original version of the bottom beam *(A)* and its improved version *(B)*.

Source: own.

3.2.2 Bottom Beam

The original bottom beam (Figure 3.5a) worked great to transmit the movement of the bottom servo motor to the bottom case, as well as keeping in place the top servo motor. However, the placement of the magnets at the ends of the beam was from the bottom, making the contact point with the magnets in the stumps flushed. This meant that the magnets would fall after prolonged use of the robot due to wear of the glue between the plastic and the magnet, and because the connection between the magnets was stronger that the glue.

For the improved version (Figure 3.5b), the entrance to the chamber of the magnets was changed to the top. This added a 0.4 mm layer of plastic between the magnets and avoided unexpected falls of the magnets. Furthermore, an angle of 45° was chamfered on the ends of the bottom beam to avoid using support material during the 3D printing process.



(A) Original Inner Base (B) Improved Inner Base

FIGURE 3.6: Comparison between the original version of the inner base *(A)* and its improved version *(B)*.
Source: own.

3.2.3 Inner Base

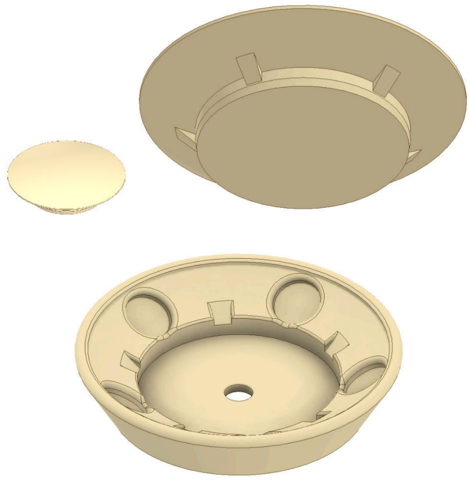
The original inner base (Figure 3.6a) had a cross-shaped base that slid inside the bottom case of the robot. It used two neodymium magnets in alternating polarities to transmit the movement to the outer base. Due to the limited amount of magnets, the robot tended to skid over the outer base on fast movements, causing the robot to loose traction and behave erratically.

For the improved version (Figure 3.6b), a star shaped base was designed that could hold an odd number of magnets. This configuration greatly improved the smoothness of the lower motion as well as making the robot more secure to the base. In addition, the alternating configuration gave the robot get a fixed orientation relative to the outer base and reduced the times when the robot lost traction.

3.2.4 Outer Base

Keeping in line with the pattern from the inner base, the original outer base (Figure 3.7a) had a concave shape with two chambers for the magnets. The contact point of the bottom case was throughout all the surface and the magnets. Unfortunately, this added extra friction between the flat surface of the

(A) Original Outer Base



(B) Improved Outer Base

FIGURE 3.7: Comparison between the original version of the outer base *(A)* and its improved version *(B)*.
Source: own.

magnets and the convex shape of the bottom case.

For the improved version (Figure 3.7b), a middle cover was designed that kept a smooth surface over all the bottom part of the case. What is more, all five magnets were enclosed between the cover and the base. The outer base had a hole in the middle to help disassembly of the base and magnets in case of maintenance.

3.3 Electronic Components

The spherical robot is controlled by a ***Raspberry Pi Zero W***[1] (Figure 3.9a), a computer with a 1 GHz ARM single-core CPU, 512 MB of RAM, 802.11 b/g/n wireless LAN, 40-pin header, a ***Camera Serial Interface*** (CSI) connector, and a micro USB power input (Raspberry Pi Foundation, 2018). The operating system on the robot is Raspbian GNU/Linux 10 Buster, a Debian GNU/Linux based distribution with kernel armv6l 5.4.51+.

[1]The schematic of the Raspberry Pi Zero W can be found at Stimson (2016).

For face tracking, a ***Zero Spy Camera*** (Figure 3.9c), a module with an OV5642[2] camera sensor (OmniVision Technologies Inc., 2009) attached to a CSI connector, is used. This camera is controlled with the custom made `PiCam` module (see Section 4.2.4), that allows access to raw YUV images at around 5 frames per second. Two ***TowerPro MG996R*** servo motors (Figure 3.9b), named ***servo motor*** ⑬ in Section 3.2, actuate the top and bottom sections of the robot (Torq Pro and Tower Pro, 2014). These motors are metal geared and have a torque of 9.40 kg cm and an angular speed of 5.51 rad s^{-1} at 4.8 V. They are connected directly to the `GND`, `5V`, `14`, and `15` pins[3] of the Raspberry Pi Zero W, and controlled with the custom made `servo` module (see Section 4.2.2), that can manage up to 100 servos asynchronously on a Raspberry Pi Zero W. Finally, a ***NeoPixel Ring — 12 × 5050 RGB LED with Integrated Drivers*** (Figure 3.9d) was used to create the iris of the robot (Adafruit Industries, 2021c). This ring uses 12 addressable RGB LEDs that can be controlled individually with one data line connected in series to the GPIO pin `18`. A wired USB-A to USB-micro connector is used to power the robot. The custom made `ring` module (see Section 4.2.3) was developed to control these diodes on a Linux based system with the correct timings. A basic schematic showing the wiring path of the electronics is shown is Figure 3.8, and pictures of the electronic components described above are presented in Figure 3.9.

[2]A 5 megapixel CMOS QSXGA camera sensor.
[3]The location of the pins are based on GPIO numbering.

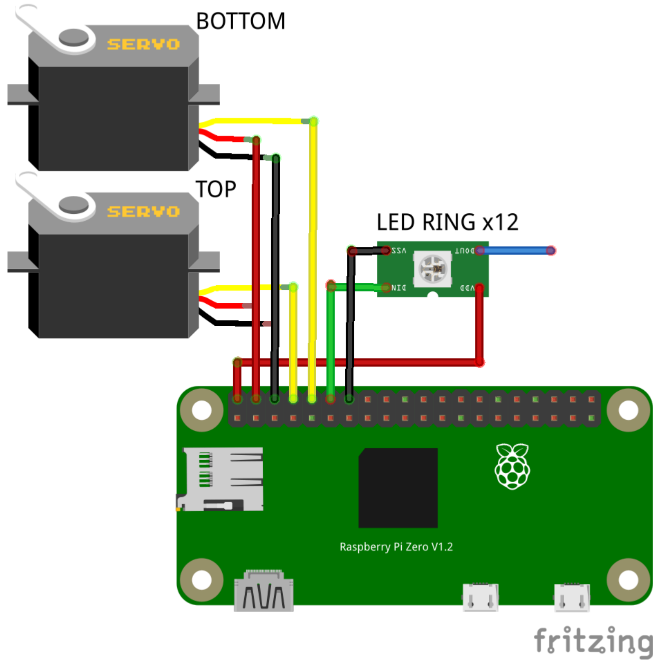
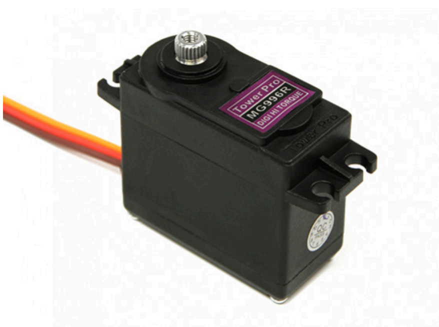
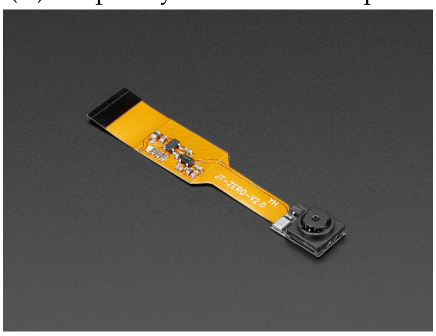


FIGURE 3.8: Basic electronic schematic of the robot.
Source: own, made with Fritzing (2021).

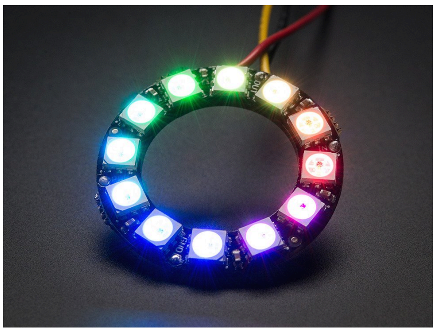
(A) Raspberry Pi Zero W Computer



(B) TowerPro MG996R Servo Motor



(C) Zero Spy Camera

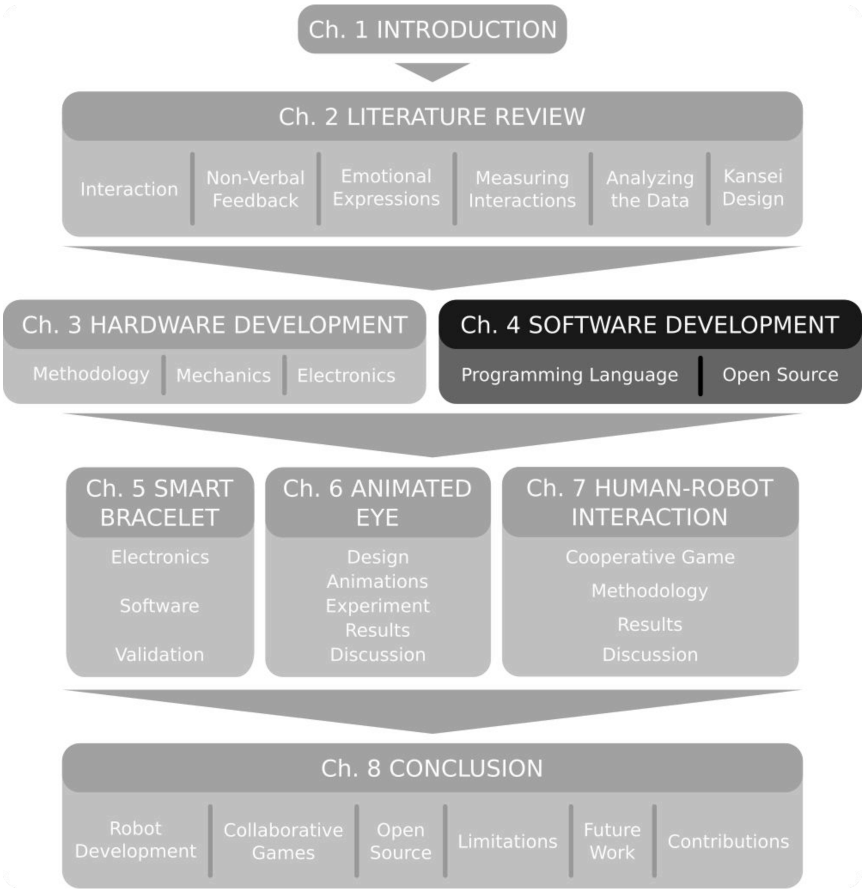


(D) NeoPixel Ring 12

FIGURE 3.9: Sample pictures of the electronic components.
Source: (A) Raspberry Pi Foundation (2017), (B) HobbyKing (2021), (C) Adafruit Industries (2021e), (D) Adafruit Industries (2021c)

Chapter 4

Software Development



This chapters presents the ***programming language*** used to program the robot and the ***open source modules*** developed to control it. An example code as well as explanation on how to use those libraries outside this thesis are included.

4.1 Programming Language

The original prototype developed by Onchi and Lee (2019) used Python 3 as the main programming language of the robot. While this interpreted language is effective for prototyping, the next version of the robot required a more robust program that could handle concurrent execution and be light weight at the same time. Therefore, the whole source code of the robot was re-designed and re-written using Go (Go, 2021), a statically programming language that was designed to be compiled, scalable, and handle concurrency.

4.1.1 The Go Programming Language

Go (2021) is an open source programming language, initially developed by Google, that is distributed under a BSD-style license[1]. Go was developed to be simple, scalable, and have fast compiling times. It also uses concurrency[2] and channels[3] as the pillars when creating programs. Because Go is a statically typed[4] and compiled language, it is ideal to be deployed into small systems and less prone to unexpected behaviors.

Listing 1 and Listing 2 show an example of the code required to update the position of different servo motors in Go and Python respectively. In the Python version (Listing 1), control of the servo motor is done as a self-contained thread using the `threading` module. In contrast, the version in Go (Listing 2) manages all servo motors inside a ***goroutine***[5] and uses ***channels*** to

[1]**BSD license:** Berkeley Software Distribution license. It is a type of license for free software that has minimal restrictions.

[2]**Concurrency:** the ability of a program to run different parts of its instructions at the same time or out-of-order, without changing the outcome.

[3]**Channels:** a type of variable that acts like a "pipe" to send and receive data.

[4]**Statically typed:** the variables used by the program are explicitly defined and check at compilation time.

[5]**Goroutine:** a function in Go that is run concurrently to the main thread.

```python
class _Update (threading.Thread):
    def __init__(self, servo):
        threading.Thread.__init__(self)
        self._servo = servo
        self._running = False

    def run(self):
        self._running = True
        self._previous_time = time.time()
        while self._running:
            delta_time = time.time() - self._previous_time
            self._previous_time = time.time()
            self._servo.update(delta_time)

    def stop(self):
        self._running = False
```

LISTING 1: Example of how to asynchronously update the position of a servo in Python 3. Taken from Onchi and Lee (2019).

```go
	updateCh := time.NewTicker(3 * time.Millisecond)

				case <-updateCh.C:
					for _, servo := range b._servos {
						if !servo.isIdle() {
							pin, pwm := servo.pwm()
							data[pin] = pwm
						}
					}
```

LISTING 2: Example of how to concurrently update the position of a servo in Go. Taken from `servo/blaster.go`. See Appendix D.2.5 for the complete source code.

control the update rate. This design allow for concurrent control of hundreds of servomotors reliably (Onchi, 2020).

4.2 Open Source Modules

For the sake of this research, and to contribute to the community of developers in the area of Robotics and Automation, several Go modules were programmed and made available as open source projects. Figure 4.1 shows a flowchart of how each module is interconnected in the robot. The main control of the robot is done at the top level `robot` package (Appendix D.1). This package depends upon the `anim` (Appendix D.5) and `tracker` (Appendix D.6) packages. To promote ***mutual facial gaze*** and increase the animacy of the robot (Holroyd et al., 2011), the module `pigo` from Simo (2020) version 1.4.2 was used. This module uses ***pixel intensity comparison-based object detection*** (Marku et al., 2014) to detect facial landmarks, such as eyes and mouth, and outputs regions of interest that might contain matching objects. The following subsections present all modules used in the robot, with notes on how they can be used outside of this research.

4.2.1 `anim`

The module `anim` is in charge of loading the animations that allow the robot to express emotion-like feedback. This package reads a `YAML`[6] that is easy to configure for a person (Appendix D.5.6). This format allows to place comments preceded by a `#` (hashtag).

This module was developed to control both the body and eyes of the robot using ***text-based animation key frames***. Each line from the YAML file represents one key frame of the animation. Each frame can be configured to move the robot at a certain direction with a customized speed, and changing the shape of the eye by closing the top and bottom eyelids. The frames from the body and the eye are loaded separately and using the keyword `sync` synchronizes two key frames from each region of the robot. This allows the robot to perform asynchronous behaviors and finish correctly at the end of the animation. Moreover, each animation can be named, and a ***playlist*** can be defined at the beginning of the file to specify which animations to use without having

[6]**YAML:** YAML Ain't Markup Language is a human readable data serialization scheme.

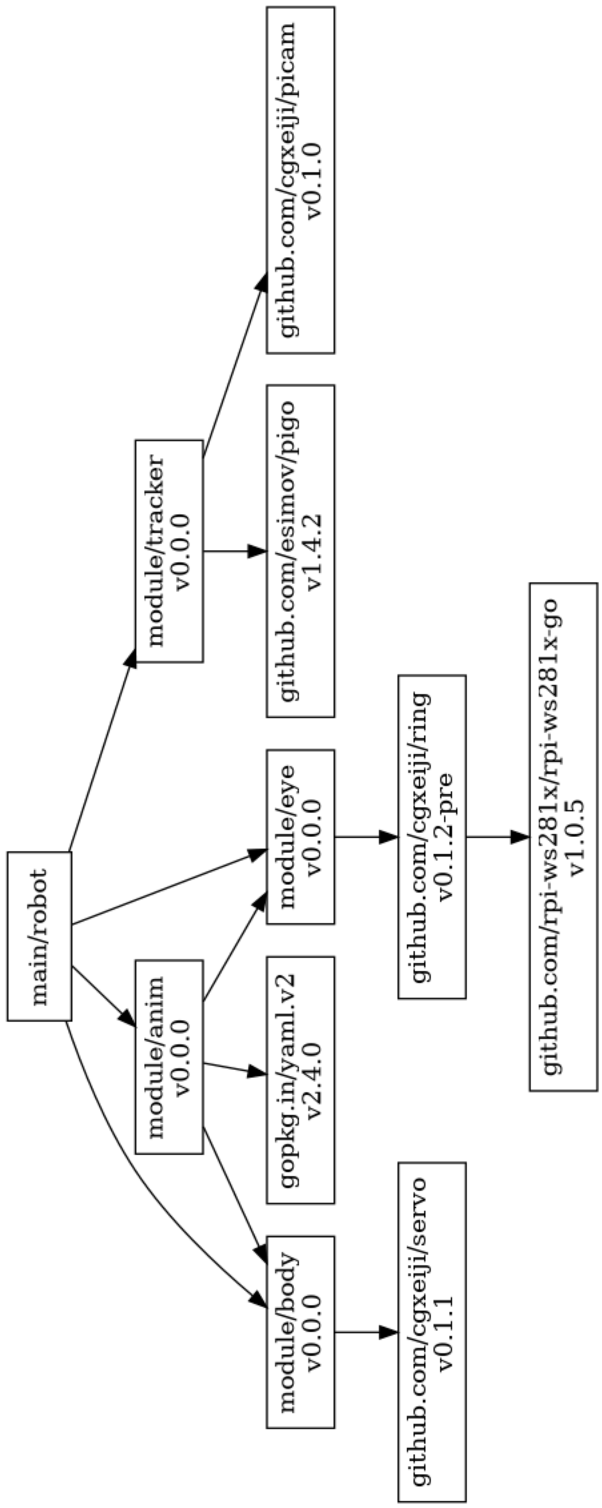


FIGURE 4.1: Modules used and developed in Go for the robot.

Source: own.

```yaml
eye.data:
  think:
    - eye: sync 1.0, 0.0, 500, 1.0
    - eye: sync 1.0, 0.0, 500, 1.0
      # shake
    - eye: sync 0.7, 0.2, 200, 1.0
    - eye: 0.8, 0.1, 200, 1.0
    - eye: 0.7, 0.2, 200, 1.0
    - eye: sync 0.8, 0.1, 200, 1.0
      # return
    - eye: sync 1.0, 0.0, 500, 1.0

body.data:
  think:
    - body: sync 0, 0.5, 0, 1.0
    - body: sync 0, 0.5, 0, 1.0
      # shake
    - body: sync 0, 0.6, 0, 1.0
    - body: 0, 0.5, 0, 1.0
    - body: 0, 0.6, 0, 1.0
    - body: sync 0, 0.5, 0, 1.0
      # return
    - body: sync 0, 0, 0, 1.0
```

LISTING 3: Example of how to program an animation into the robot using a YAML configuration file.

to delete unused ones. This functionality permitted rapid testing of complex animations. Listing 3 presents an example of an animation in the robot.

4.2.2 `servo`

This module uses `pi-blaster` (Hirst et al., 2013) to control servo motors on a Raspberry Pi. Under the hood, it opens a pipeline to `/dev/pi-blaster` and sends commands in the format `GPIO=PWM`. The module calculates the appropriate PWM[7] based on the speed and position of the servo motor and groups the writes to `/dev/pi-blaster` at a rate of 40 ms, if multiple servos are connected. A detailed documentation of this module is presented at the reference site (`https://pkg.go.dev/github.com/cgxeiji/servo`). An example of how to use his module is presented in Listing 4.

Each connected servo motor is managed independently from one another

[7] **PWM:** Pulse-Width Modulation

```go
package main

import (
	"fmt"
	"log"
	"github.com/cgxeiji/servo"
)

func main() {
	// Use servo.Close() to close the connection of all servos and pi-blaster.
	defer servo.Close()
	// Create a new servo connected to gpio 14.
	myServo := servo.New(14)
	myServo.MinPulse = 0.05 // Set the minimum pwm pulse width (default: 0.05).
	myServo.MaxPulse = 0.25 // Set the maximum pwm pulse width (default: 0.25).
	myServo.SetPosition(90) // Set the initial position to 90 degrees.
	myServo.SetSpeed(0.2)   // Set the speed to 20% (default: 1.0).
	myServo.Name = "My Servo"
	// Print the information of the servo.
	fmt.Println(myServo)
	// Connect the servo to the daemon.
	err := myServo.Connect()
	if err != nil { log.Fatal(err) }
	defer myServo.Close()

	myServo.SetSpeed(0.5) // Set the speed to half. This is concurrent-safe.
	myServo.MoveTo(180)   // This is a non-blocking call.

	/* do some work */

	myServo.Wait() // Call Wait() to sync with the servo.

	// MoveTo() returns a Waiter interface that can be used to move and wait on
	// the same line.
	myServo.MoveTo(0).Wait() // This is a blocking call.
}
```

LISTING 4: Example of the servo module to control a servo motor.

and is designed to be concurrent-safe[8]. If the package `servo` detects that `pi-blaster` is not running on the system when executed, it will throw a warning and redirect all writes to `/dev/null`. This way, it is possible to build and test code on machines other than a Raspberry Pi or do a cold run before committing.

[8] It is safe to use in parallel or pseudo-parallel processes without the risk of a race condition.

4.2.3 `ring`

`ring` is a wrapper of `rpi-ws281x-go` (Supcik et al., 2020) specialized in controlling ring-shaped LEDs. This module adds the ability to use ***layers*** to do complex animations. Each ***layer*** supports color transparency and blending is handled automatically. A detailed documentation of this module is presented at the reference site (`https://pkg.go.dev/github.com/cgxeiji/ring`). An example of how to use the module with a simple fading animation is presented in Listing 5. Because `rpi-ws281x` needs to access `/dev/mem` to create correct PWM timings, it is necessary to run the compiled binary with root[9] permissions.

Compilation Compiling directly on a Raspberry Pi might take too long. The recommended way to compile this module is to cross-compile using a Docker container[10].

4.2.4 PiCam

PiCam is a Go wrapper to `raspiyuv`[11] to get `[]uint8` and `image.Image` data of the latests frame captured by the Raspberry Pi camera. Under the hood, it executes "`$ raspiyuv -timeout 0 -timelapse 0`" to get raw frames. This module was created to avoid the dependency on `GoCV`[12] to access the camera on a Raspberry Pi to do real-time face detection (Listing 6). A detailed documentation of this module is presented at the reference site (`https://pkg.go.dev/github.com/cgxeiji/picam`).

[9]Superuser account in Unix and Linux systems used for administrative purposes.

[10]An isolated container inside a computer that can execute a virtual machine with an specific configuration.

[11]Utility program from Raspberry Pi to acquire raw images of a camera.

[12]Go implementation of the OpenCV image processing library.

```go
package main

import (
	...
	"github.com/cgxeiji/ring"
)

func main() {
	// Initialize the ring.
	r, err := ring.New(&ring.Options{
		LedCount:      12,  // adjust this to the number of LEDs you have
		MaxBrightness: 180, // value from 0 to 255
	})
	r.Offset(-math.Pi / 3) // you can set a rotation offset for the ring
	if err != nil { log.Fatal(err) }
	defer r.Close() // Make sure to properly close the ring.
	// Create a new layer.  This will be a static white background.
	bg, err := ring.NewLayer(&ring.LayerOptions{ Resolution:  1, ContentMode: ring.ContentScale })
	if err != nil { log.Fatal(err) }
	bg.SetAll(color.White) // Set all pixels of the layer to white.
	r.AddLayer(bg) // Add the layer to the ring.
	// Create a mask layer.  This will fade the background.
	bgMask, err := ring.NewLayer(&ring.LayerOptions{ Resolution: 1 })
	if err != nil { log.Fatal(err) }
	r.AddLayer(bgMask)
	if err := r.Render(); err != nil { log.Fatal(err) } // Render the ring.
	/* ANIMATION SETUP */
	done := make(chan struct{})   // this will cancel all animations
	render := make(chan struct{}) // this will request a concurrent-safe render
	var ws sync.WaitGroup         // this makes sure we close all goroutines
	/* render goroutine */
	ws.Add(1)
	go func() {
		defer ws.Done()
		for {
			select {
			case <-done:
				return
			case <-render:
				if err := r.Render(); err != nil { log.Fatal(err) }
			}
		}
	}()
	/* fading goroutine */
	ws.Add(1)
	go func() {
		defer ws.Done()
		c := color.NRGBA{0, 0, 0, 0}
		step := uint8(5)
		for {
			for a := uint8(0); a < 255; a += step {
				c.A = a
				bgMask.SetAll(c)
				select {
				case <-done:
					return
				case render <- struct{}{}:
				}
				time.Sleep(20 * time.Millisecond)
			}
			for a := uint8(255); a > 0; a -= step {
				c.A = a
				bgMask.SetAll(c)
				select {
				case <-done:
					return
				case render <- struct{}{}:
				}
				time.Sleep(20 * time.Millisecond)
			}
		}
	}()
	fmt.Println("Press [ENTER] to exit")
	stdin := bufio.NewReader(os.Stdin)
	stdin.ReadString('\n')
	close(done) // Stop all animations
	ws.Wait() // Wait for goroutines to exit
}
```

LISTING 5: Example of the `ring` module with several layers (abbreviated due to space).

Currently, three image formats are available:

- picam.YUV
- picam.RGB
- picam.Gray

The time between frames, measured on a ***Raspberry Pi Zero W***, is between 180 ms to 210 ms for a 640 × 480 pixels image. It is possible to test the acquisition speed by running:

```
$ cd $(go env GOPATH)/src/github.com/cgxeiji/picam
$ go test -bench .  -benchtime=10x
```

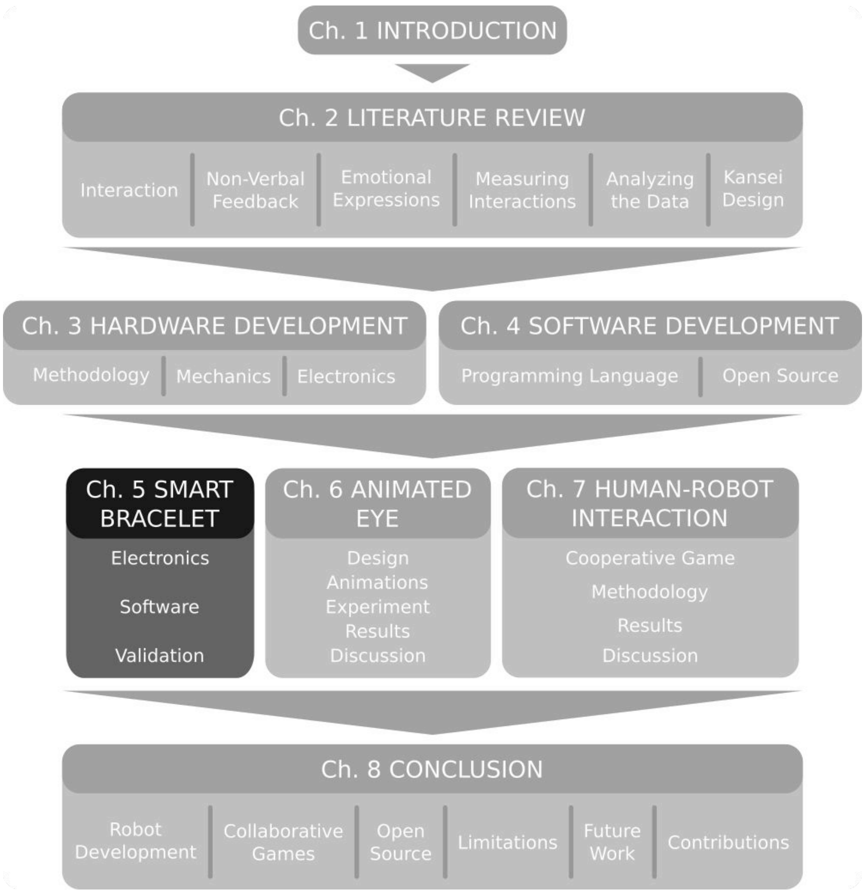
codes of line on a Linux based machine. This will take 10 frames and output the average time between each frame. Changing `-benchtime=10x` to `100x` or `Nx` will change the number of frames to test.

```go
package main

import (
	"fmt"
	"io/ioutil"
	"log"
	"github.com/cgxeiji/picam"
	pigo "github.com/esimov/pigo/core"
)

func main() {
	cam, err := picam.New(640, 480, picam.Gray)
	if err != nil { log.Fatal(err) }
	defer cam.Close()
	cParams := pigo.CascadeParams{
		MinSize:     90,
		MaxSize:     200,
		ShiftFactor: 0.1,
		ScaleFactor: 1.1,
		ImageParams: pigo.ImageParams{
			Rows: cam.Height,
			Cols: cam.Width,
			Dim:  cam.Width,
		},
	}
	classifierFile, err := ioutil.ReadFile("./facefinder")
	if err != nil { log.Fatal(err) }
	p := pigo.NewPigo()
	classifier, err := p.Unpack(classifierFile)
	if err != nil { log.Fatal(err) }
	fmt.Println("Starting face detection")
	fmt.Println("Press Ctrl+C to stop")
	for {
		cParams.Pixels = cam.ReadUint8()
		faces := classifier.RunCascade(cParams, 0.0) // 0.0 is the angle
		faces = classifier.ClusterDetections(faces, 0.1)
		// Get the face with the highest confidence level
		var maxQ float32
		index := 0
		for i, face := range faces {
			if face.Q > maxQ {
				maxQ = face.Q
				index = i
			}
		}
		face := pigo.Detection{}
		if index < len(faces) {
			face = faces[index]
		}
		if face.Scale == 0 {
			// no face detected
			fmt.Printf("\rno face detected")
			continue
		}
		x := face.Col - cam.Width/2
		y := -face.Row + cam.Height/2 // y is flipped
		fmt.Printf("\rface is (%d, %d) pixels from the center", x, y)
	}
}
```

LISTING 6: Example of the PiCam module to do real-time face detection on a Raspberry Pi.

Chapter 5

Smart Bracelet



This section includes the ***electronics*** and ***development*** process of the ***smart bracelet*** used to measure physiological information for this study. It also presents the ***open source modules*** created to control its sensors, and ends with a ***validation*** of its measurements.

FIGURE 5.1: Custom made smart bracelet to record physiological data during the experiment.

Source: own.

5.1 Electronic Development

To measure physiological data for this research, the open-source portable measuring device, shown in Figure 5.1, was developed that could record data autonomously.

The schematic of the main part of the bracelet is presented in Figure 5.2. This part of the bracelet housed a ***SSD1306 OLED display***, a ***Raspberry Pi Zero W*** (Raspberry Pi Foundation, 2018), a 3.7 V@400 mA · h ***Lithium Ion battery*** with an ***USB-C Micro-Lipo Charger*** (Adafruit Industries, 2021a) to charge it and a ***PowerBoost 500 Basic*** (Adafruit Industries, 2021d) to transform the voltage from 3.7 V to 5 V, and ***USB-C female*** connectors.

To design a future-proof flexible device with swappable sensors, each sensor was connected using a ***USB-C cable*** and communicates with I^2C serial interface with the main controller. Three types of sensor were required for this study: a ***heart-rate sensor*** to measure the heart-rate of participants, a ***galvanic skin response sensor*** to measure electrodermal activity (EDA), and an ***inertial sensor*** to measure hand movement.

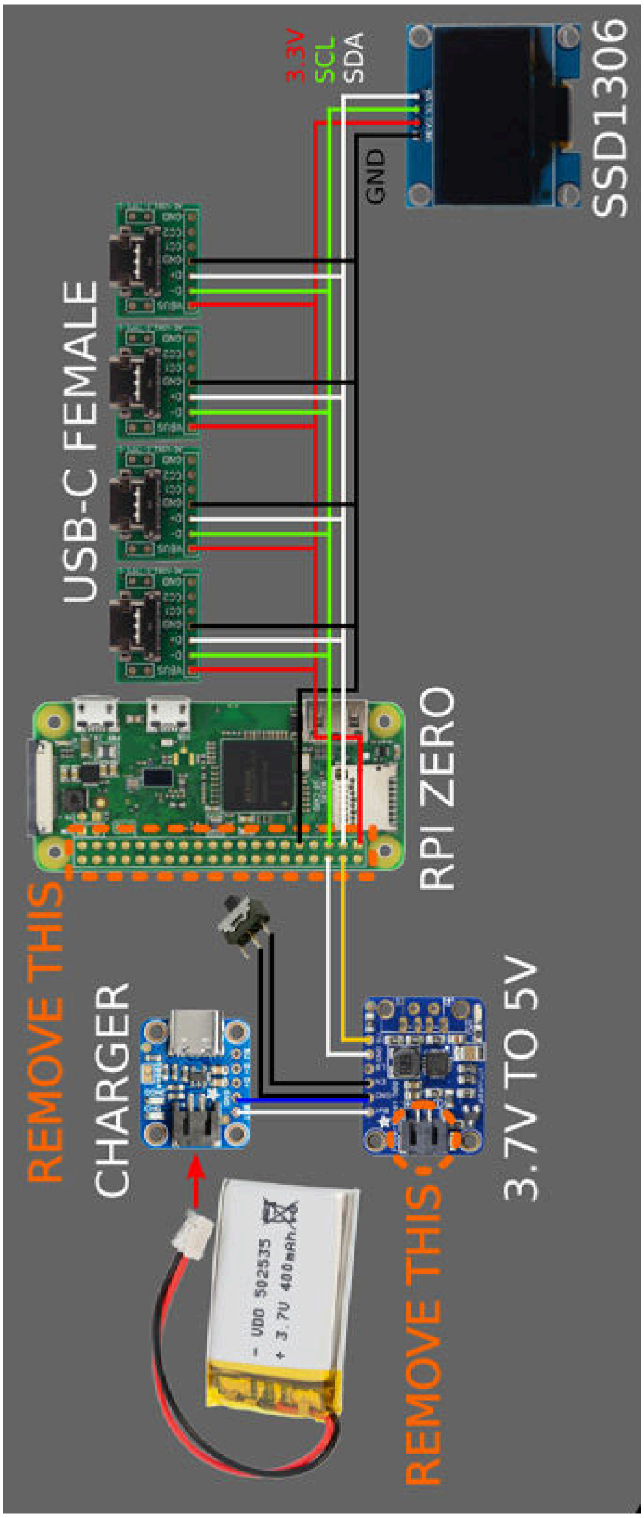


FIGURE 5.2: Schematic of the smart bracelet.

Source: own.

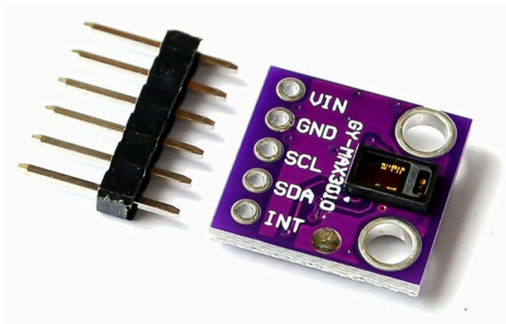


FIGURE 5.3: MAX30102[2]: pulse oximeter and heart-rate sensor.
Source: HiLetgo (2019).

5.1.1 Heart-Rate: MAX30102

Heart-rate is measured with a ***MAX30102*** pulse oximeter and heart-rate sensor from Maxim Integrated (2020). This sensor includes a red and infrared LED photodetectors with a sampling resolution of 16-bits. It operates at 3.3 V and can communicate with an I^2C[1] serial interface. The module presented in Figure 5.3 was used during development.

5.1.2 Electrodermal Activity Sensor: Grove GSR

EDA[3] is measured with the ***GSR Sensor***[4] module from Seeed Technology Inc. (2014). This sensor, shown in Figure 5.4, measures the electrical conductance of the skin using two contact points made out of nickel and calculates the resistance between those probes. It has an operating voltage of 3.3 V and 5 V and outputs an analog signal relative to the resistance of the probes.

The analog signal is connected to an ***ADS1015*** ADC[5] module from Adafruit Industries (2021b), which transforms the signal into digital information at a

[1] **I^2C**: Inter-Integrated Circuit, a two wire serial communication interface.
[2] While the text on the circuit board says “MAX3010”, the actual sensor placed on the circuit board is the “MAX30102”.
[3] **EDA:** Electrodermal Activity
[4] **GSR:** Galvanic Skin Response.
[5] **ADC:** Analog-Digital Converter.

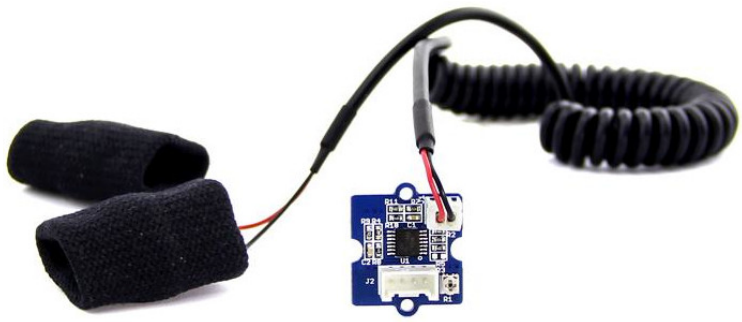


FIGURE 5.4: Galvanic skin response sensor.
Source: Seeed Technology Inc. (2014).

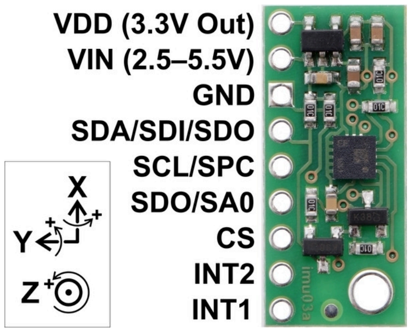


FIGURE 5.5: LSM6DS33 6-degree IMU Module.
Source: Pololu (2021).

resolution of 12-bits. This module has an operating range of 2 V to 5.5 V and uses I^2C to transfer data.

5.1.3 Motion Sensor: LSM6DS33

To synchronize the timing of events when recording the information, the ***LSM6DS33*** 6-degree IMU[6] module from Pololu (2021) was used. This module, presented in Figure 6.4, has a 3-axis accelerometer and 3-axis gyroscope integrated, which provide movement information. It has an operating range of 2.5 V to 5.5 V and uses both I^2C and SPI[7] interfaces to transfer data.

[6]**IMU:** Inertial Measurement Unit.
[7]**SPI:** Serial Peripheral Interface.

5.2 Software Development

Several open source modules were developed to use the sensors on the Raspberry Pi platform using the programming language Go[8]. Figure 5.6 shows a flowchart of how each module is interconnected in the smart bracelet. The main module of the bracelet is located at the top (Appendix D.7). This modules uses the `ads1x15` module (Appendix D.8) to control the skin conductance board, the `lsm6` module (Appendix D.9) to control the IMU sensor, the `max3010x` module (Appendix D.10) to control the heart-rate sensor, and the public modules `image` (Go, 2021) and `periph` (The Periph Authors, 2021) as a low-level interface for the Raspberry Pi. Each developed module uses the `serial` module (Appendix D.11) to communicate using I^2C with the peripherals.

5.2.1 ads1x15

`ads1x15` is a wrapper of `periph` (The Periph Authors, 2021) specialized in controlling ADS1015 and ADS1115 analog to digital converter devices. This module streamlines the acquisition of analog measurements with helper functions that can be called asynchronously. A detailed documentation of this module is presented at the reference site (`https://pkg.go.dev/github.com/cgxeiji/ads1x15`).

5.2.2 lsm6

`lsm6` is a wrapper of `periph` (The Periph Authors, 2021) specialized in controlling LSM6DS3 inertial measurement units. This module streamlines the acquisition of accelerometer and gyroscope readings with helper functions

[8]See Section 4.1 for more information about why this programming language was used.

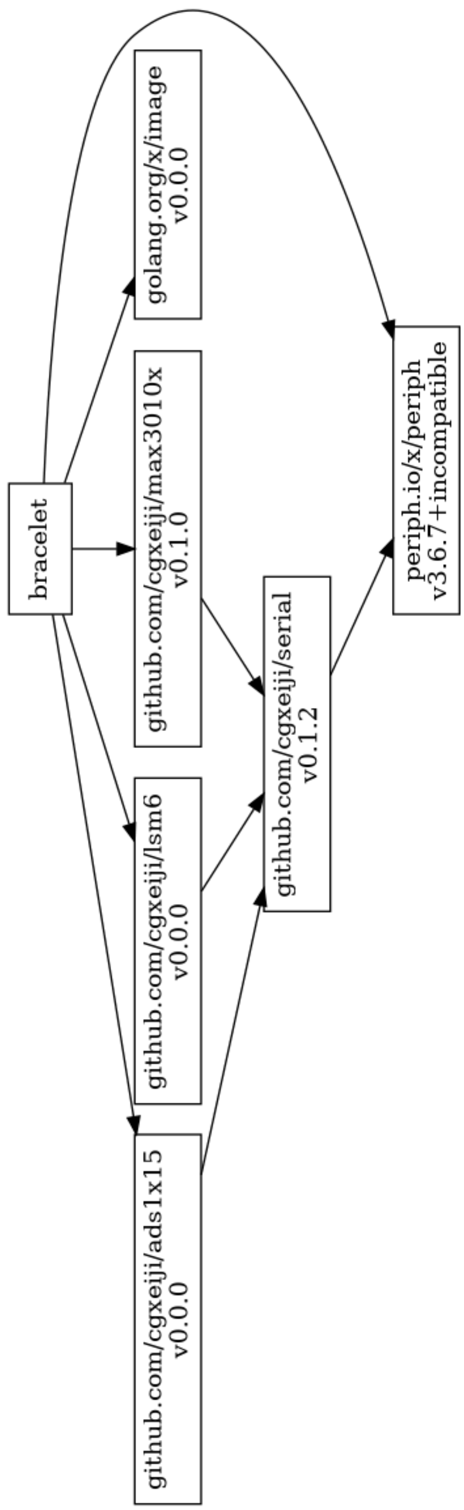


FIGURE 5.6: Modules used and developed in Go for the bracelet.

Source: own.

```go
func main() {
    sensor, err := max3010x.New()
    if err != nil {
        log.Fatal(err)
    }
    defer sensor.Close()

    // Detect the heart rate
    hr, err := sensor.HeartRate()
    if errors.Is(err, max3010x.ErrNotDetected) {
        hr = 0
    } else if err != nil {
        log.Fatal(err)
    }
    fmt.Println("Heart rate:", hr)

    // Detect the SpO2 level
    spO2, err := sensor.SpO2()
    if errors.Is(err, max3010x.ErrNotDetected) {
        spO2 = 0
    } else if err != nil {
        log.Fatal(err)
    }
    fmt.Println("SpO2:", spO2)
}
```

LISTING 7: Example of the max3010x module to read heart-rate and SpO_2 information.

that can be called asynchronously. A detailed documentation of this module is presented at the reference site (`https://pkg.go.dev/github.com/cgxeiji/lsm6`).

5.2.3 `max3010x`

`lsm6` is a wrapper of `periph` (The Periph Authors, 2021) specialized in reading heart-rate and SpO_2 information from the MAX3010x sensor family. A detailed documentation of this module is presented at the reference site (`https://pkg.go.dev/github.com/cgxeiji/max3010x`). An example of how to use his module is presented in Listing 7. An `max3010x.ErrNotDetected` error code will be returned when trying to read the heart-rate or SpO_2 values when the sensor is not in contact with a person. Moreover, an option to access low-level features of the sensor is presented in Listing 8.

```
func main() {
    sensor, err := max3010x.New()
    if err != nil {
        log.Fatal(err)
    }

    defer sensor.Close()
    device, err := sensor.ToMax30102()
    if errors.Is(err, max3010x.ErrWrongDevice) {
        fmt.Println("device is not MAX30102")
        return
    } else if err != nil {
        log.Fatal(err)
    }

    // Get the values for the IR and red LEDs.
    ir, red, err := device.IRRed()
    if err != nil {
        log.Fatal(err)
    }
}
```

LISTING 8: Example of the max3010x module to read raw information from the red and IR LEDs.

5.2.4 Validation

Two simple tests were conducted to check if the device could record movements and physiological information correctly. The first one was a motion test, where the motion sensor was placed over the thumb and 5 simples gestures were done in consecutive order. Each gesture was repeated three times, with a delay of 1 s between each movement. This information was then plotted and visually compared with video footage, as shown in Figure 5.7.

The second test was a mock experiment where both the device and the tablet were synchronized. The participants were asked to keep their hand with the smart bracelet relaxed on top of the table to avoid inducing too much noisy into the electrodermal and heart-rate sensors. A sample recording of the mock experiment is presented in Figure 5.8. The first two graphs from the top show the accelerometer and gyroscope information respectively. The middle graph show the data from the red and infrared LEDs. The next graph is the information from the electrodermal sensor. Finally, the last graph shows the times when a suggestion and an action was taken on the maze.

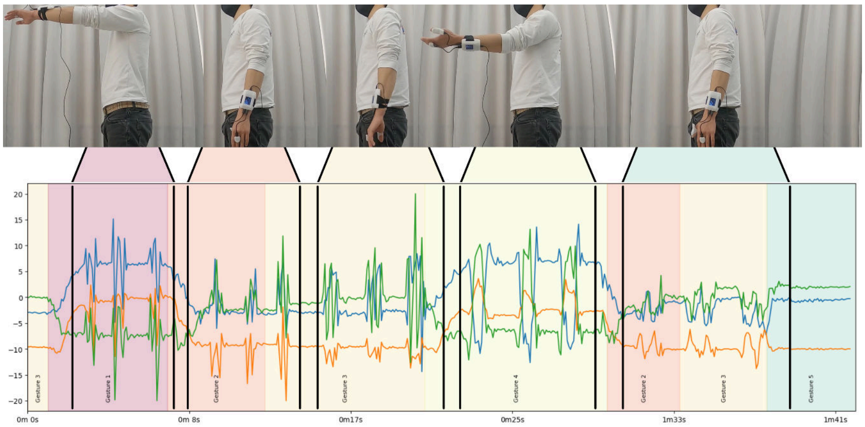


FIGURE 5.7: Sample measurement to verify that each sensor is correctly recoding the data of the participant.

Source: own.

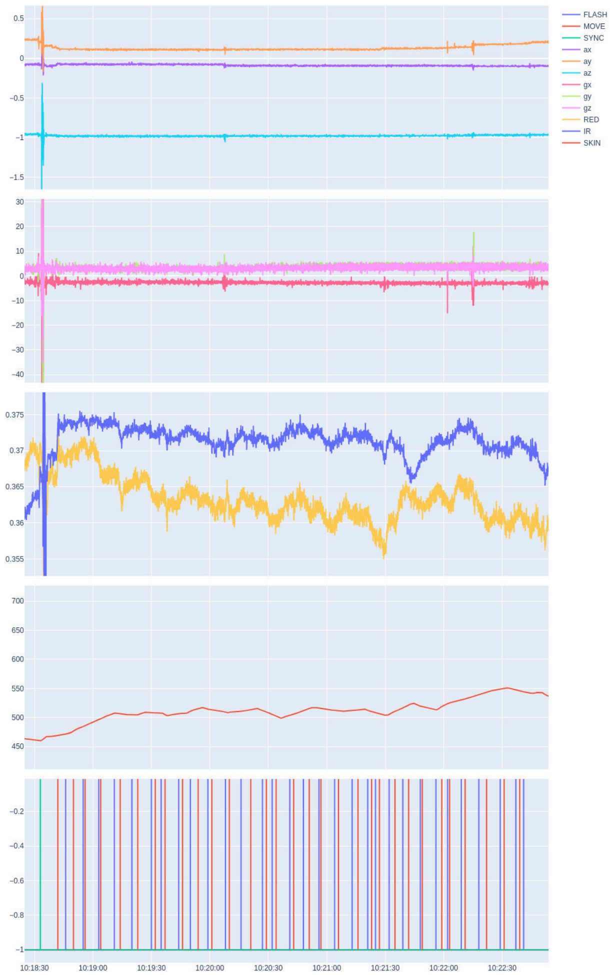
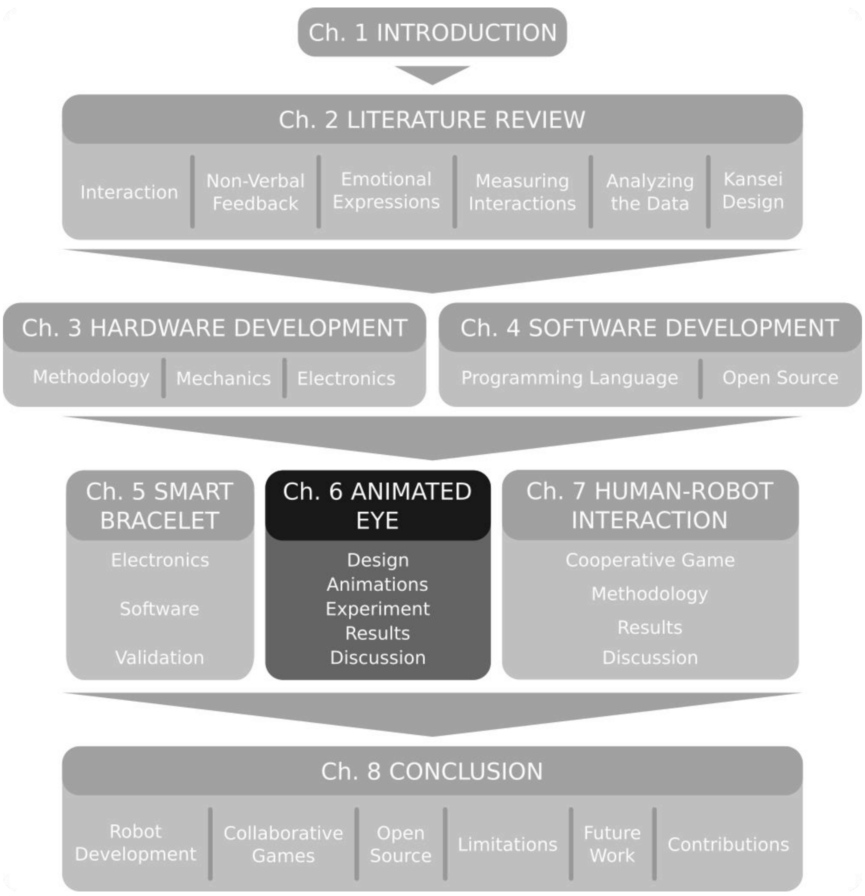


FIGURE 5.8: Sample measurement to verify that each sensor is correctly recoding the data of the participant.

Source: own.

Chapter 6

Animated Eye



This chapters includes the ***redesign of the eye*** of the robot, and how each ***animation*** was created. It also presents the ***validation study*** done to test the effect of LED animations on the emotional feedback of the spherical robot. It ends with a ***discussion*** of the results obtained.

6.1 Research Motivation

Among the different mediums of non-verbal communication that can be used to express information in robots, movement, light, and sound can be used to convey information (Löffler, Schmidt, and Tscharn, 2018). To find the minimal means to create engagement, instead of adding all possible elements to a robot, this study aims to explore each modality, one by one, adding extra features only when needed. The previous research by Onchi and Lee (2019) focused on the study of motion to express emotion-like information. Adding upon that research and based on the findings of Onchi, Saakes, and Lee (2020), the goal of this research is to validate if having simple light animations can enhance the information being express by the spherical robot.

Our working hypothesis is that, by adding lights that resemble the iris of the eye, it will be possible to simulate the emotional information expressed through the eyelids (Ekman, Friesen, and Ellsworth, 1972), without the need to add a physical mechanism which may add upon the complexity of the robot.

FIGURE 6.1: Robot facing to the front with LED fully bright.
Source: own.

6.2 Design

Light animation was added to the robot as part of the ***iris*** on the main eye. A 12 ***LED ring*** ⑤ was placed behind the 3D printed ***iris*** ④. A semi-transparent black plastic film was glued in front of the iris to ***cover*** ③ the LED ring and smooth the output of the LEDs.

6.2.1 LED Ring: NeoPixel 12 × 5050 RGB

The animations from the eye were done with a ***NeoPixel Ring*** 12 × 5050 RGB LED (Adafruit Industries, 2021c). This ring uses a series of WS2812B RGB LEDs which have an integrated driver that can control the intensity of the LED with only one data cable. These drives require a specific communication timing in order to correctly read the information, thus the custom made `ring` module was developed (see Section 4.2.3 for a detailed explanation of this module).

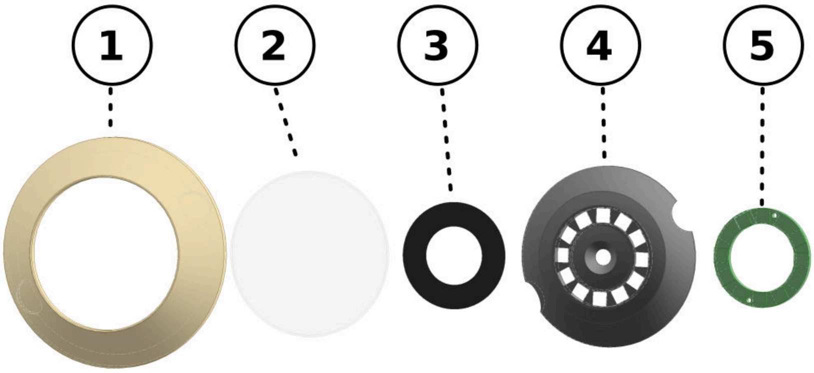


FIGURE 6.2: Explosion view of the robot's Eye. ① Eye Rim | ② Cornea | ③ Iris Cover | ④ Iris | ⑤ LED Ring |

Source: own.

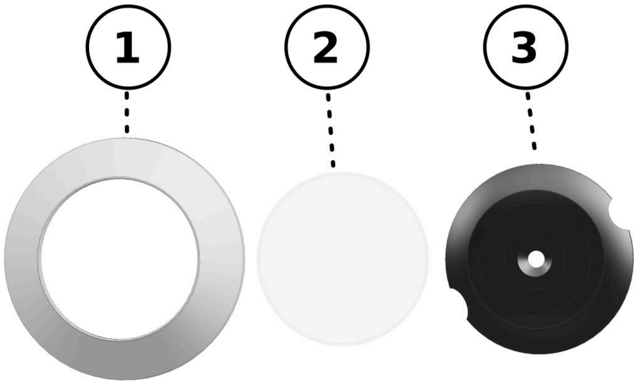


FIGURE 6.3: Original version of the robot's Eye. ① Eye Rim | ② Cornea | ③ Iris |

Source: own.

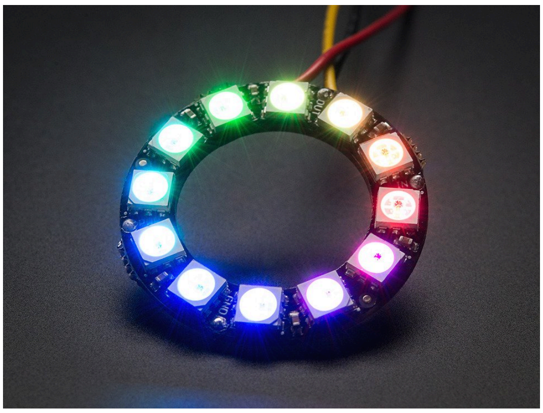


FIGURE 6.4: NeoPixel ring 12 × 5050 RGB LED with integrated drivers.

Source: Adafruit Industries (2021c).

FIGURE 6.5: Motion and LED variations. Front facing full eye opened (left), downward facing upper eyelid closed (middle), and upward facing bottom eyelid closed (right).
Source: own.

6.3 Animations

The animation of the body and blinking patterns were based on how actors and digital animators show emotions (Thomas and Johnston, 1981). There was no brow present, and the directional focus was coupled with a whole-body movement of the robot. To express more complex shapes without a brow, different arc lengths were displayed with the LED rings as shown in Figure 6.5. Moreover, subtle eye saccades were replaced with bigger movements and body jerks. The color of the LED can affect the emotion being conveyed (Hyeon, Pan, and Yoo, 2019; Kim, Kim, and Jo, 2020). In this regard, a correlation between color and emotion can be made based on Plutchik's Wheel of Emotions (Terada, Yamauchi, and Ito, 2012). Thus, to control for the effect of the color of the LED, the color of the iris was set to a constant hue of 203 (#009CFF, Deep Sky Blue) to give a neutral impression and the brightness to the maximum value. Only the shape of the LED ring was modulated and the brightness was gradually adjusted between transitions (e.g. blinking) to make the animations smoother. From there, building upon the results of Onchi and Lee (2019), the following animations were designed:

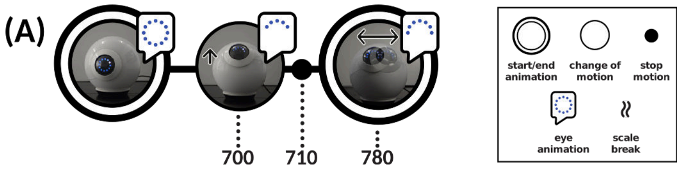


FIGURE 6.6: Animation timeline of happy state. Times are shown in milliseconds.
Source: Onchi, Cornet, and Lee (2021).

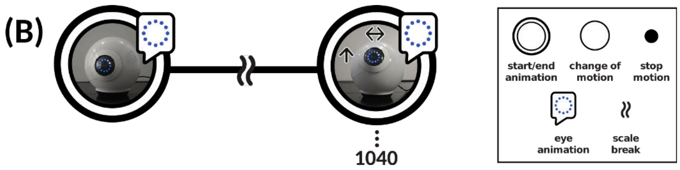


FIGURE 6.7: Animation timeline of surprise state. Times are shown in milliseconds.
Source: Onchi, Cornet, and Lee (2021).

Happy (A) The expression of happiness is in the positive end of the emotional valence scale. Depending on the level of arousal, this expression can range from relaxation to excitement. To convey the emotion, the robot faces upwards from an eye level position in 700 ms and then quickly moves left and right, akin to a happy dog wagging its tail. The bottom half of the LED ring is turned off to simulate the effect of raised cheeks. The timeline of the animation is presented in Figure 6.6.

Fear/Surprise (B) Fear is a highly aroused state with a negative valence, while surprise is a highly aroused state with a neutral valence (Posner, Russell, and Peterson, 2005). To visually represent this state in the robot, the LED ring is completely shown and several fast blinks are added to indicate disbelieve. The attentional focus of the robot goes from the person to an upwards position in around 1 s, and quickly moves left and right while blinking. The timeline of the animation is presented in Figure 6.7.

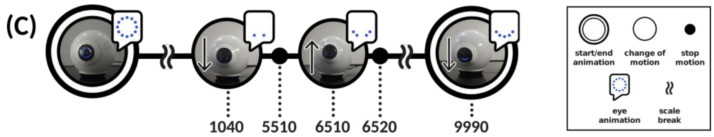


FIGURE 6.8: Animation timeline of sadness state. Times are shown in milliseconds.
Source: Onchi, Cornet, and Lee (2021).

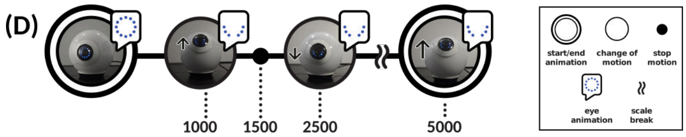


FIGURE 6.9: Animation timeline of anger state. Times are shown in milliseconds.
Source: Onchi, Cornet, and Lee (2021).

Sadness (C) Sadness is described as an unpleasant, low aroused emotion (Posner, Russell, and Peterson, 2005). A slow-paced downward motion can be used to show this state (Onchi and Lee, 2019). To heighten the effect of looking down and give the illusion of sobbing, the top half of the LED ring is turned off and the bottom two LEDs change intensity during the animation. This change is synchronized with the robot slowly moving in small ovals. The timeline of the animation is presented in Figure 6.8.

Disgust/Anger (D) Both disgust and anger are associated with low valence and high arousal (Posner, Russell, and Peterson, 2005). Usually, an intense glare with tighten brows accompany these emotions (Ekman and Friesen, 2003). To simulate the squeeze of the brows, only the bottom middle part of the LED ring is lighted. At the same time, the robot does a fast-paced upwards movement, quickly moves down, and slowly returns up. The timeline of the animation is presented in Figure 6.9.

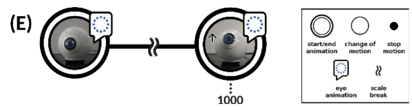


FIGURE 6.10: Animation timeline of confusion state. Times are shown in milliseconds.
Source: Onchi, Cornet, and Lee (2021).

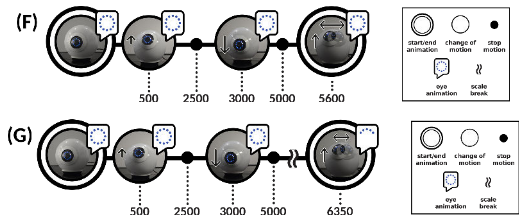


FIGURE 6.11: Animation timeline of assertion and negation states. Times are shown in milliseconds.
Source: Onchi, Cornet, and Lee (2021).

Confusion (E) The expression of confusion is not necessarily attached to an emotional valence or arousal, but still conveys useful information during communication. To mimic a perplexed gaze, the robot moves slightly upwards with all lights in the LED ring fully brightened. Then, a single blink is shown. After 1 s, the robot blinks twice fast. The timeline of the animation is presented in Figure 6.10.

Assertion (F) and Negation (G) During communication between two people, upwards and downwards gaze aversion can signal a cognitive effort (Andrist et al., 2014). Based on that, these animations were designed with an initial downwards motion to convey a thinking process. To express disagreement, the robot looks up and slowly moves left and right with all LED in the ring turned on, whereas, for agreement, the robot looks up and swiftly moves left and right with the bottom half of the LED ring turned off. The timeline of the assertion animation and the one for negation animation are shown in Figure 6.11.

6.4 Experiment

6.4.1 Methodology

A within-subjects experiment with 8 design students (4F / median age: 20–32) from the University of Tsukuba was conducted between the robot showing only motion feedback (control) and motion with LED feedback (LED).

Participants were asked to sit in front of a 14 inch laptop and watch 28 video clips of a researcher interacting with the robot, while the robot replies with a specific feedback. To avoid order bias (Furnham, 1986; Krosnick and Alwin, 1987), the presentation of each video clip was randomized following the Fisher-Yates shuffling algorithm, which is a randomization scheme that produces unbiased permutations (Knuth, 1998, p. 145). Each video sample was muted to avoid any possible context bias caused by stimuli other than the movement and the LED feedback (Lee, Yang, and Lee, 2019).

After watching one video sample, the participants evaluated the emotional impression of the interaction using the Self-Assessment Manikin (SAM) (see Subsection 2.5.2). Then, they were asked to write an imaginary dialog or context, in English, that describes the interaction between the instructor and the robot. In particular, they were asked:

- *"What did the person possibly say to the robot? (You can imagine any situation)"*
- *"How did the robot possibly react? (You can imagine any reply)"*

The participants could take as much time as necessary to fill the questionnaires and take breaks in between if required. This survey was programmed using the Godot game engine. The length of the whole experiment was around 40 minutes.

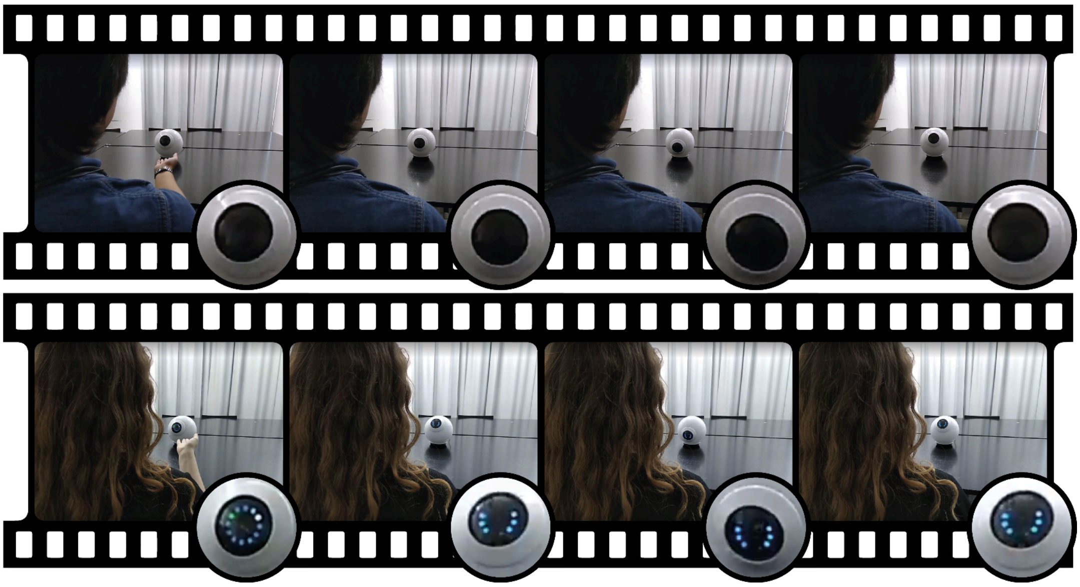


FIGURE 6.12: Video samples showing the robot with and without LED animations.

Source: own.

6.4.1.1 Video Samples

Each video sample was recorded from behind a male and female instructor doing the same hand gesture and their facial expressions are not shown to keep a neutral context. Then, the robot replied with an animation with the LED ring on (target group) or off (control group). Video samples of the animations were used instead of direct interaction to ensure the consistency of the animations among the participants.

Given the seven animations presented in Section 6.3, a total of 28 video samples were recorded ($2_{male/female} \times 2_{control/LED} \times 7_{animations} = 28$).

6.5 Results

The results presented are two fold: the emotional scoring of each video sample, and text analysis of the animations.

TABLE 6.1: Shapiro-Wilk's Normality Test of SAM Scores

Type	Animation	Valence	Arousal	Dominance
Control	**Angry**	0.173	***0.021**	0.119
Control	**Assertion**	***0.037**	***0.017**	0.246
Control	**Confused**	***0.017**	***0.035**	0.347
Control	**Happy**	0.188	0.153	***0.073**
Control	**Negation**	***0.025**	0.255	***0.025**
Control	**Sad**	***0.019**	***0.048**	***0.044**
Control	**Surprised**	0.426	***0.035**	***0.049**
LED	**Angry**	0.326	***0.039**	0.162
LED	**Assertion**	***0.039**	***0.023**	0.066
LED	**Confused**	***0.039**	***0.023**	0.446
LED	**Happy**	0.314	0.066	0.122
LED	**Negation**	***0.046**	0.095	0.051
LED	**Sad**	***0.027**	0.516	***0.012**
LED	**Surprised**	***0.008**	***0.012**	0.195

*significance level at 0.05

6.5.1 SAM

A Shapiro-Wilk's normality test, presented in Table 6.1, was performed to check the normality of the SAM scoring. This highlighted that several sample scores did not pass the normality test. Therefore, it was decided that the analysis of the data will be done using a Wilcoxon signed-rank test, rather than paired t-test, as this test does not assume that the samples are normally distributed, is capable to work with small sample sizes, and does not require continuous data.

A Wilcoxon signed rank test (Table 6.2) was conducted to compare the ***valence***, ***arousal***, and ***dominance*** scores of SAM with the control and LED conditions. The results suggests that the effect of LED ($M = 5.0$, $SD = 1.6$) over the control ($M = 4.1$, $SD = 1.5$) condition on the emotional ***valence*** is positive and statistically significant; $V = 2658.5$, $p < 0.001$. Also, the effect of LED ($M = 5.7$, $SD = 1.7$) over the control ($M = 4.4$, $SD = 1.7$) condition on the emotional ***arousal*** is positive and statistically significant; $V = 3392.5$,

TABLE 6.2: Wilcoxon Signed Rank Test of SAM by LED

SAM	$\bar{X}_{control}$ (SD)	$\bar{X}_{LED}$ (SD)	V	p
valence	4.1 (1.5)	5.0 (1.6)	2658.5	***0.000**
arousal	4.4 (1.7)	5.7 (1.7)	3392.5	***0.000**
dominance	4.7 (1.7)	5.1 (1.6)	2422.5	***0.049**

*significance level at 0.05

TABLE 6.3: Wilcoxon Signed Rank Test of Valence by LED

Valence	$\bar{X}_{control}$ (SD)	$\bar{X}_{LED}$ (SD)	V	p
angry	4.4 (1.4)	5.0 (1.5)	45	0.301
assertion	4.4 (1.2)	5.1 (1.7)	48	0.191
confused	3.8 (1.6)	5.4 (1.1)	55	***0.005**
happy	4.1 (1.3)	5.4 (1.9)	64	***0.048**
negation	3.8 (1.1)	4.5 (1.2)	69	0.100
sadness	3.3 (1.6)	3.4 (1.5)	68	0.645
surprised	5.2 (1.6)	5.9 (1.5)	52	0.305

*significance level at 0.05

$p < 0.001$. Finally, the effect of LED ($M = 5.1$, $SD = 1.6$) over the control ($M = 4.7$, $SD = 1.7$) condition on the emotional ***dominance*** is positive and statistically significant; $V = 2422.5$, $p = 0.049$.

A detailed analysis into the difference of each animation (Figure 6.13) showed that some animations changed more than others. A Wilcoxon signed rank test on the ***valence*** scores of each animation (Table 6.3) evidenced positive, statistically significant differences for $\boldsymbol{happy_{control}}$ ($M = 4.1$, $SD = 1.3$) and $\boldsymbol{happy_{LED}}$ ($M = 5.4$, $SD = 1.9$) conditions ($V = 64$, $p = 0.048$); and $\boldsymbol{confused_{control}}$ ($M = 3.8$, $SD = 1.6$) and $\boldsymbol{confused_{LED}}$ ($M = 5.4$, $SD = 1.1$) conditions ($V = 55$, $p = 0.005$).

In the same manner, a Wilcoxon signed rank test on the ***arousal*** scores of each animation (Table 6.4) evidenced positive, statistically significant differences for $\boldsymbol{angry_{control}}$ ($M = 4.2$, $SD = 1.3$) and $\boldsymbol{angry_{LED}}$ ($M = 5.8$, $SD = 1.7$) conditions ($V = 86$, $p = 0.034$); $\boldsymbol{assertion_{control}}$ ($M = 4.6$, $SD = 1.7$) and

TABLE 6.4: Wilcoxon Signed Rank Test of Arousal by LED

Arousal	$\bar{X}_{control}$ (SD)	$\bar{X}_{LED}$ (SD)	V	p
angry	4.2 (1.3)	5.8 (1.7)	86	***0.034**
assertion	4.6 (1.7)	6.5 (1.4)	98	***0.004**
confused	2.9 (1.5)	5.3 (1.5)	91	***0.002**
happy	5.7 (1.5)	5.9 (1.7)	45	0.661
negation	4.4 (1.5)	5.3 (1.3)	86	***0.037**
sadness	4.1 (1.5)	4.3 (1.4)	30	0.797
surprised	4.6 (2.1)	6.4 (0.9)	84	***0.007**

*significance level at 0.05

TABLE 6.5: Wilcoxon Signed Rank Test of Dominance by LED

Dominance	$\bar{X}_{control}$ (SD)	$\bar{X}_{LED}$ (SD)	V	p
angry	4.5 (1.9)	5.1 (1.6)	64	0.466
assertion	4.4 (1.8)	6.1 (1.3)	100	***0.002**
confused	4.1 (1.5)	4.9 (1.4)	51	0.116
happy	6.2 (1.4)	5.5 (1.5)	32	0.183
negation	5.2 (1.6)	4.8 (1.2)	25	0.282
sadness	4.1 (1.4)	3.9 (2.0)	36	0.811
surprised	4.3 (1.4)	5.5 (1.5)	60	***0.017**

*significance level at 0.05

assertion$_{LED}$ ($M = 6.5$, $SD = 1.4$) conditions ($V = 98$, $p = 0.004$); ***confused***$_{control}$ ($M = 2.9$, $SD = 1.5$) and ***confused***$_{LED}$ ($M = 5.3$, $SD = 1.5$) conditions ($V = 91$, $p = 0.002$); ***negation***$_{control}$ ($M = 4.4$, $SD = 1.5$) and ***negation***$_{LED}$ ($M = 5.3$, $SD = 1.3$) conditions ($V = 86$, $p = 0.037$); and ***surprised***$_{control}$ ($M = 4.6$, $SD = 2.1$) and ***surprised***$_{LED}$ ($M = 6.4$, $SD = 0.9$) conditions ($V = 84$, $p = 0.007$).

Finally, a Wilcoxon signed rank test on the ***dominance*** scores of each animation (Table 6.5) showed positive, statistically significant differences for ***assertion***$_{control}$ ($M = 4.4$, $SD = 1.8$) and ***assertion***$_{LED}$ ($M = 6.1$, $SD = 1.3$) conditions ($V = 100$, $p = 0.002$); and ***surprised***$_{control}$ ($M = 4.3$, $SD = 1.4$) and ***surprised***$_{LED}$ ($M = 5.5$, $SD = 1.5$) conditions ($V = 60$, $p = 0.017$).

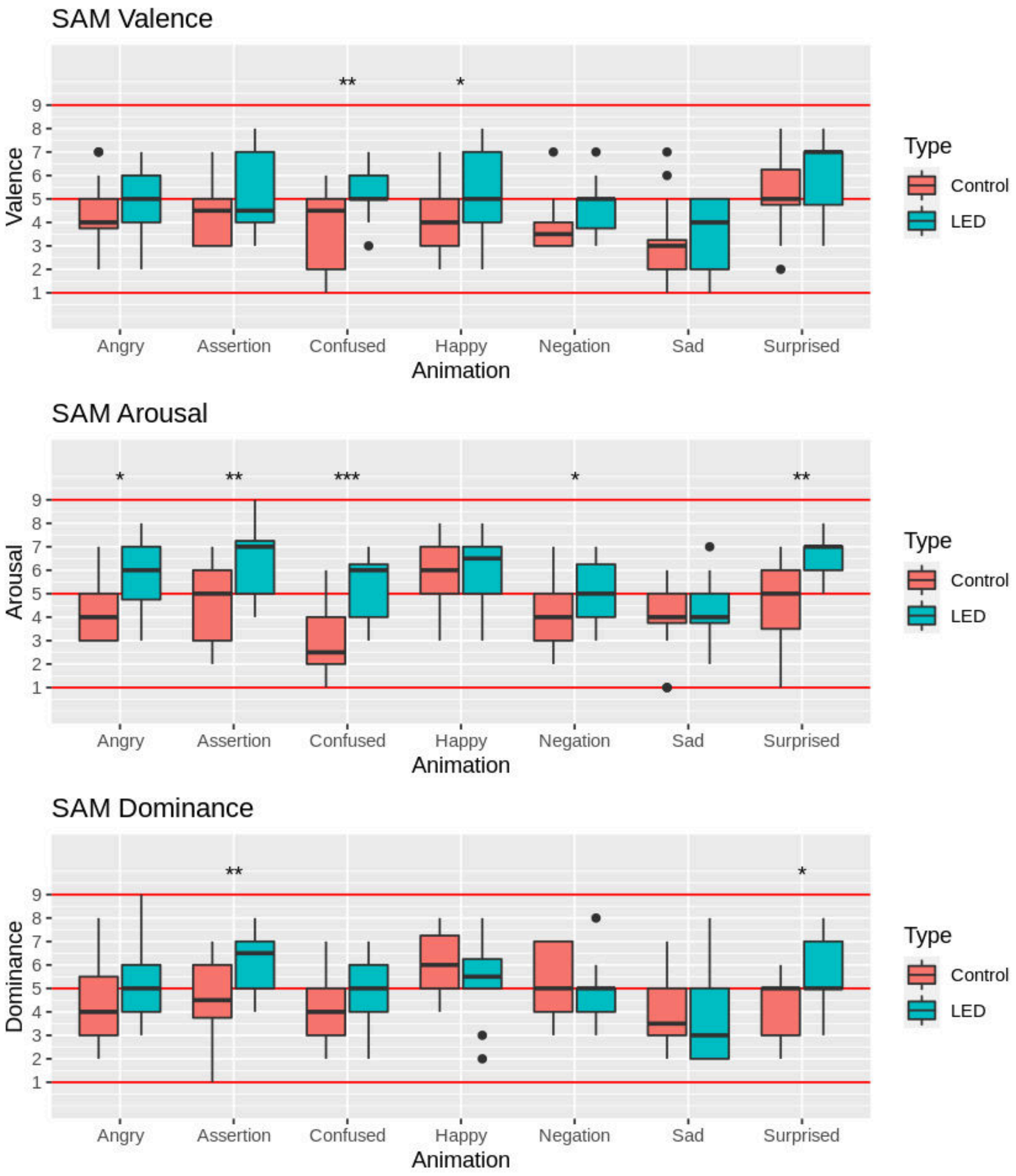


FIGURE 6.13: Box-plot of the scores of SAM on the emotional impression of each animation.

Source: own.

TABLE 6.6: Most Common Unique Words per Animation

Animation	Control	LED
angry	can, look, shopping, think, no	find, think, wired, you, maybe
assertion	like, question, joke, thinking, wait	let, think, cat, do, like
confused	ok, answer, ah, understand, yes	idea, let, like, maybe, the
happy	question, thing, answer, do, like	question, something, think, happy, look
negation	question, negative, no, show, sorry	look, question, no, surprised, thinking
sadness	yes, answer, difficult, terrible, sad	sad, sorry, embarrassed, promise, seemed
surprised	activated, surprised, can, like, reacted	like, task, want, excited, wow

6.5.2 Text Analysis

Each participant was asked to write a short imaginary dialog between the person and the robot after watching each sample video. Different adjectives and nouns were used to describe each interaction. These dialogues were evaluated using the ***NRC Valence, Arousal, and Dominance (NRC-VAD)*** lexicon (Mohammad, 2018; Mohammad, 2020), available from the ***textdata*** package (Hvitfeldt and Silge, 2020) of R Studio. This lexicon contains a list of words scored from 0.0 (lowest VAD) to 1.0 (highest VAD). The participants were free to write any type of dialog. The following text is an example of the dialog created for the LED Surprised animation:

- Person: "*See! There is a monkey running on the road.*"
- Robot: "*Where? Where?*"

The dialogues were tokenized, and English stop words cleaned, using the ***tm*** (Feinerer, Hornik, and Artifex Software, Inc., 2020) and ***stopwords*** (Benoit, Muhr, and Watanabe, 2021) packages from R Studio. The most common words across all video samples were *"I"*, *"asked"*, *"robot"*, *"person"*, and *"know"*. The five most common unique words per animation are presented in Table 6.6. Missing scores of sentences that did not contain any words from the NRC-VAD lexicon were assumed to be neutral in valence, arousal, and dominance. These sentences were assigned a score of 0.5 for each dimension. The general scoring of each animation is presented in Figure 6.14. A paired samples t-test was conducted to compare the score of NRC-VAD in control and LED conditions. There were no statistically significant differences in the emotional evaluation of the texts, except for the valence value of assertion control ($M = 0.54$, $SD = 0.15$) and assertion LED ($M = 0.63$, $SD = 0.13$) conditions; $t_{(15)} = 3.44$, $p = 0.004$. A discussion of these results is presented in Section 6.6.

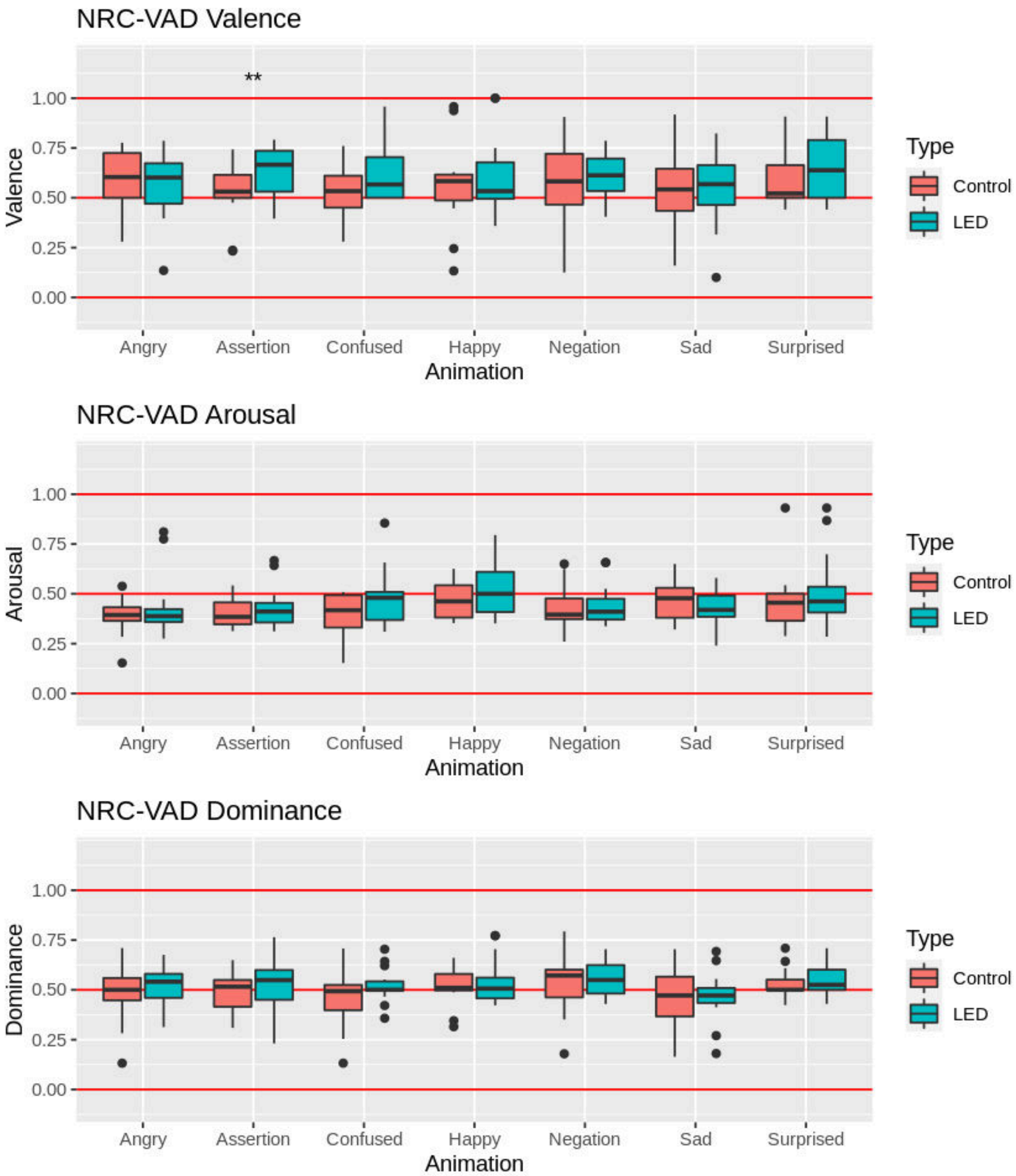


FIGURE 6.14: Box-plot of the scores of NRC-VAD on the context of each animation.

Source: own.

6.6 Discussion

The results showed that, in general, adding light feedback raises the emotional valence and arousal of the emotion being expressed. Also, some of the animations' dominance increased if light feedback is added.

When looking at each animation in detail, adding the LED ring raised the valence of happy and confused animations, from negative to slightly positive scores. On the other hand, the arousal level of all animations increased by 0.9 points or more, except for happy and sad animations. Finally, the emotional dominance for surprised and assertion increased.

Because of the nature of the Circumplex Model of Affection[1], an exact categorization of an emotion to a numerical value on the valence-arousal scale cannot be made. However, it is possible to show tendencies toward a type of emotion. According to Russell (1980), anger tends toward a slightly negative valence and slightly high arousal, happiness tends toward positive valence and slightly high arousal, sadness tends toward a negative valence and neutral arousal, and surprise tends toward a slightly positive valence and high arousal[2].

When comparing the average SAM scores of the emotions expressed by the robot (Tables 6.3 and 6.4) and the placement on the Circumplex Model of Affection suggested by Russell (1980), it is possible to see that the ***angry*** animation is closer the expected valence-arousal state when the LED is used, while the ***sad*** animation is closer to the expected valence-arousal state for the control robot. An interesting effect of the LED can be seen for the ***surprised*** animation, which participants considered to be closer to excited, while the ***happy*** animation seemed to be closer to astonished.

[1]See Section 2.5.1 for more details.
[2]See Section 2.5.1 for a visual representation of this scale.

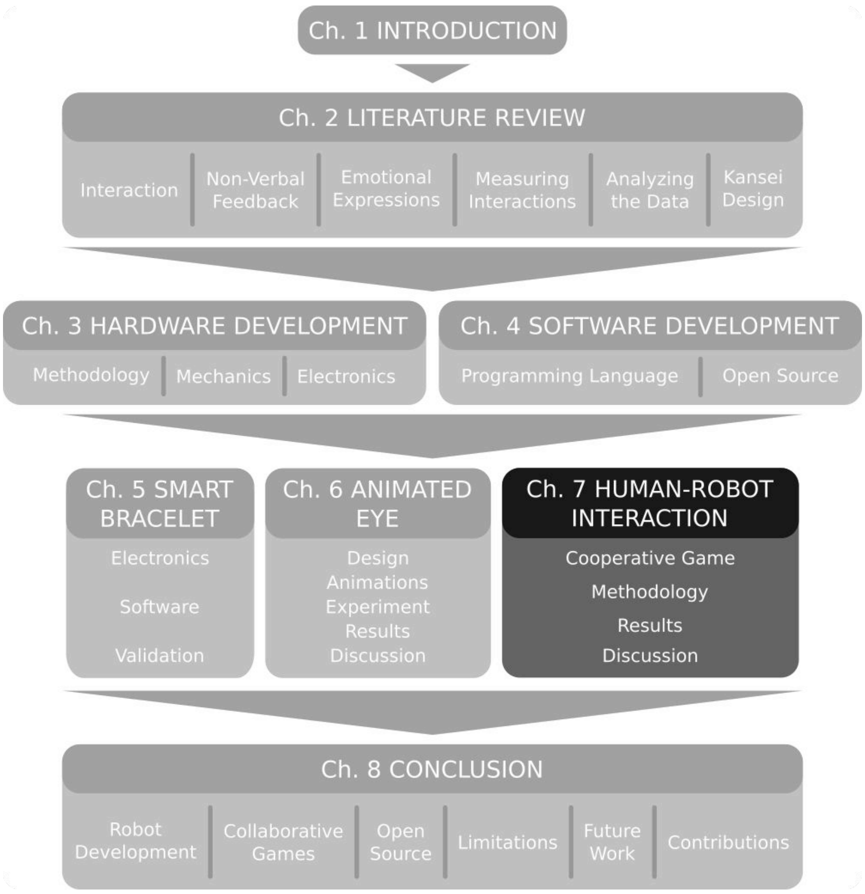
These results go in line with studies about light in human-robot communication (Baraka, Paiva, and Veloso, 2015; Funakoshi et al., 2008; Song and Yamada, 2017) in which lights and colors are used to increase non-verbal expressions. It is worth noting that there was no significant change in the emotional impression of sadness, which could imply that slow paced negative motions are not affected by adding LED feedback. What is more, light feedback made informational movements (confusion, assertion, and negation) express more neutral valences, making it less likely to misunderstand those motions as negative emotions. In other words, it is possible to include light animations to adjust the intensity of an emotion and to create mixed emotional states within robots. Having a variable output other than just movement may increase the animacy perceived, thus creating a more engaging experience.

In summary, this study evaluated the effects of adding light animation feedback to the movements of a single-eyed spherical robot. The results indicated that adding an LED changes the emotional impression of the robot. It is possible to use this stimulus to modulate the valence and arousal of the emotion being expressed and create a tool to dynamically change the emotion of a non-humanoid robot using movement, light, and animations.

Some of the limitations of this study is the restricted number of participants and the limited interaction they had with the robot to evaluate the animations. It is possible that forming a long-term bond with the robot might change or reinforce the information that the robot is trying to express. More research is needed in this regard.

Chapter 7

Human-Robot Interaction



This chapters starts with the design of the ***cooperative game*** used on the study on ***human-robot interaction***. It has a section on the development and validation of the ***smart bracelet*** used to measure physiological data. A ***discussion*** of the results is presented at the end of the chapter.

7.1 Research Motivation

After identifying the minimal means to express acceptable non-verbal information, the next goal is to verify if those elements can be used to create engagement in human-robot interactions. As presented in Section 2.6.3, a kind of interaction that promotes collaboration between people and other agents is a cooperative game. These activities have been used in Human-Robot Interaction research (Lee and Hwang, 2008; Wainer et al., 2013; Jeri et al., 2017; Paetzel, Perugia, and Castellano, 2020) and it was deemed fitting to study how the interaction experience with the robot compares to interacting with a computer and a person.

Our working hypothesis is that the interaction experience with the robot will be in between interacting with a person and a computer. This research also aims to verify the interaction on a subjective and biological level by using validated questionnaires[1] and biometric sensors[2].

7.2 Cooperative Game Design

In this research, a cooperative maze solving game[3], developed in the Godot Game Engine (Godot, 2021), was used in which a person (***Player A***) and a game partner (***Player B***) cooperate with each other to escape a randomly generated maze. In particular, ***Player B*** is randomly selected between the spherical robot, a computer, or a person (research instructor) to study how the impression of ***Player A*** changes when playing the game with different partners.

[1] SAM (Section 2.5.2) and WAI (Section 2.6.1)
[2] Using the smart bracelet (Chapter 5).
[3] Section 2.6.3 has a detailed explanation about ***cooperative games***.

This collaborative maze solving game consisted of three sessions with five stages. Each stage had one ideal path leading to the exit with several dead ends along the way (Figure 7.2). Two players (***Player A*** and ***Player B***) collaborate to solve the maze in the least amount of steps possible. The challenge consisted in that ***Player A*** could only decide which way to go, but did not have any information about the maze (Figure 7.5), while ***Player B*** was only aware of the general direction of the exit but was not aware of which path was the correct one (Figure 7.3). During the experiment, it was up to ***Player A*** to decide whether to follow ***Player B***'s guidance or not.

To control for unwanted effects caused by the decision making process of the robot, computer, and human, the same underlying algorithm was used regardless of who was ***Player B***. During the game, ***Player B*** suggested the correct path 4 out of 5 times. The suggested path was randomized using the Fisher-Yates shuffling algorithm (Knuth, 1998, p. 145). The decision time of ***Player B*** was kept between 1 s and 3 s.

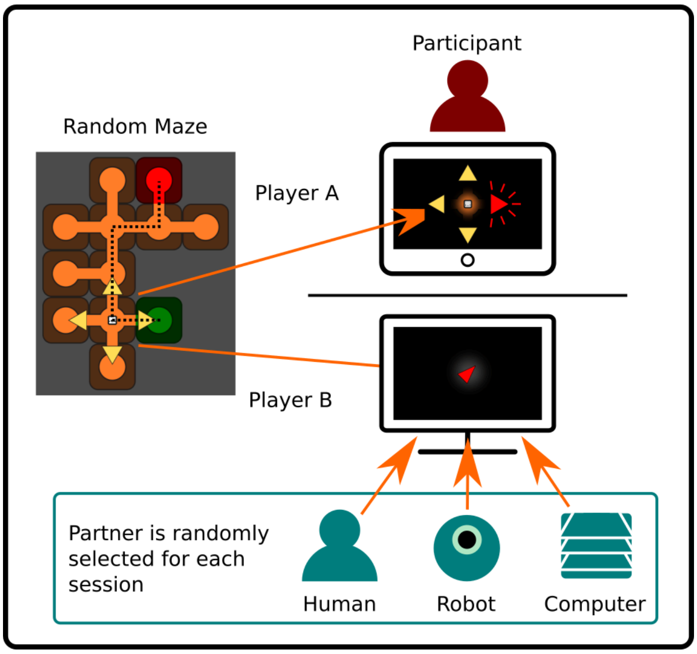


FIGURE 7.1: Layout of the collaborative game.

Source: own

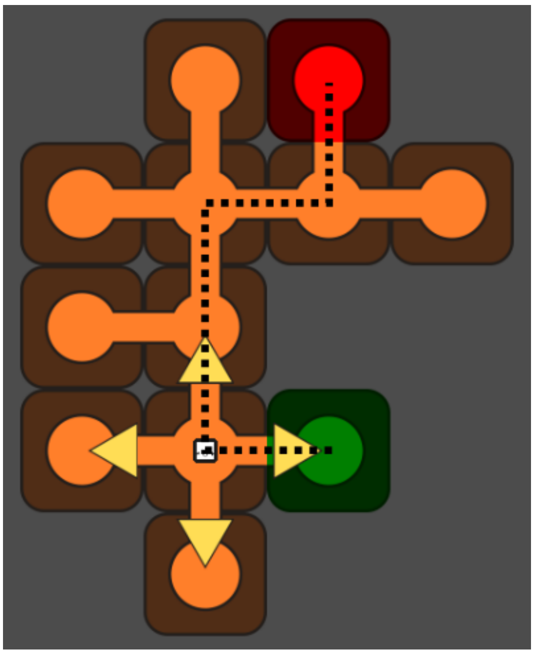


FIGURE 7.2: Randomly generated maze showing the correct path from the entrance (green) leading to the exit (red), and several dead ends attached along the way.

Source: own

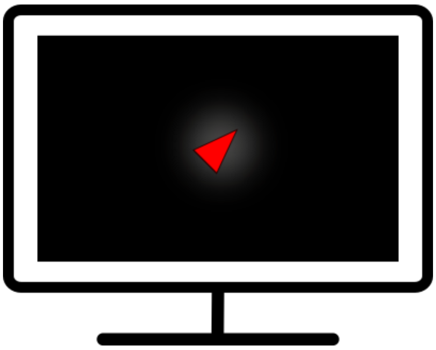


FIGURE 7.3: Screen seen by Player B, showing the general direction of the exit of the maze.

Source: own

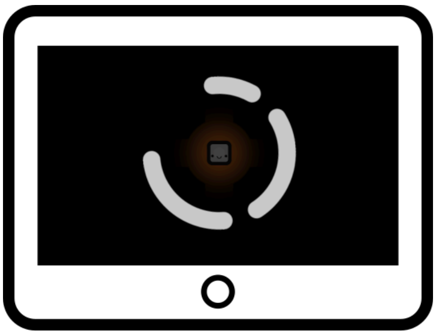


FIGURE 7.4: Thinking animation shown to Player A while waiting for Player B to maze a suggestion

Source: own

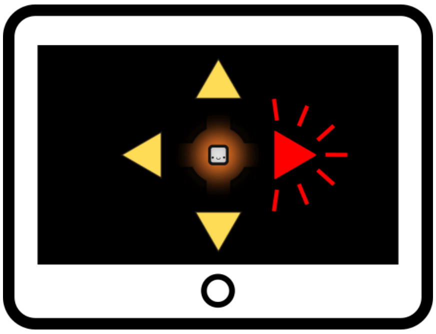


FIGURE 7.5: Selection screen seen by Player A, showing the direction suggested by Player B.

Source: own

7.3 Methodology

To research the difference when people interact with a sociable robot, compared to a computer or human, a within-subjects experiment with 24 participants (12F / $\text{median}_{\text{age}}$ = 25–29) from the University of Tsukuba was conducted. The purpose of this study was to use a collaborative game-based interaction to measure the trust and emotional changes of a person toward human and non-human partners. In particular, this study focused on the robot's motion and light feedback to see if the person's trust in the robot can be improved.

7.3.1 Experimental Procedure

The procedure of the experiment was be divided in four major parts, shown in the flowchart of Figure 7.6. From the top, during the ***orientation*** phase, an explanation of the research and its procedure was read aloud to the participant either in Japanese (Appendix B.2) or English (Appendix B.3). Then, the participant could choose to take part or not in the experiment. After signing the ***Agreement Form*** (in Japanese Appendix B.4 or English Appendix B.5), the participant was asked to complete the RoSAS survey (see more at Section 2.6.2).

During the ***familiarization*** phase, a detailed explanation on the use of the tablet was given. The instructor explained how to solve the maze with a partner, and placed the smart bracelet (see more at Section 5) on the non-dominant hand of the participant. After that, the participant had a practice period to get familiar with the system. During this practice session, the person solved the maze with the instructor until they felt comfortable.

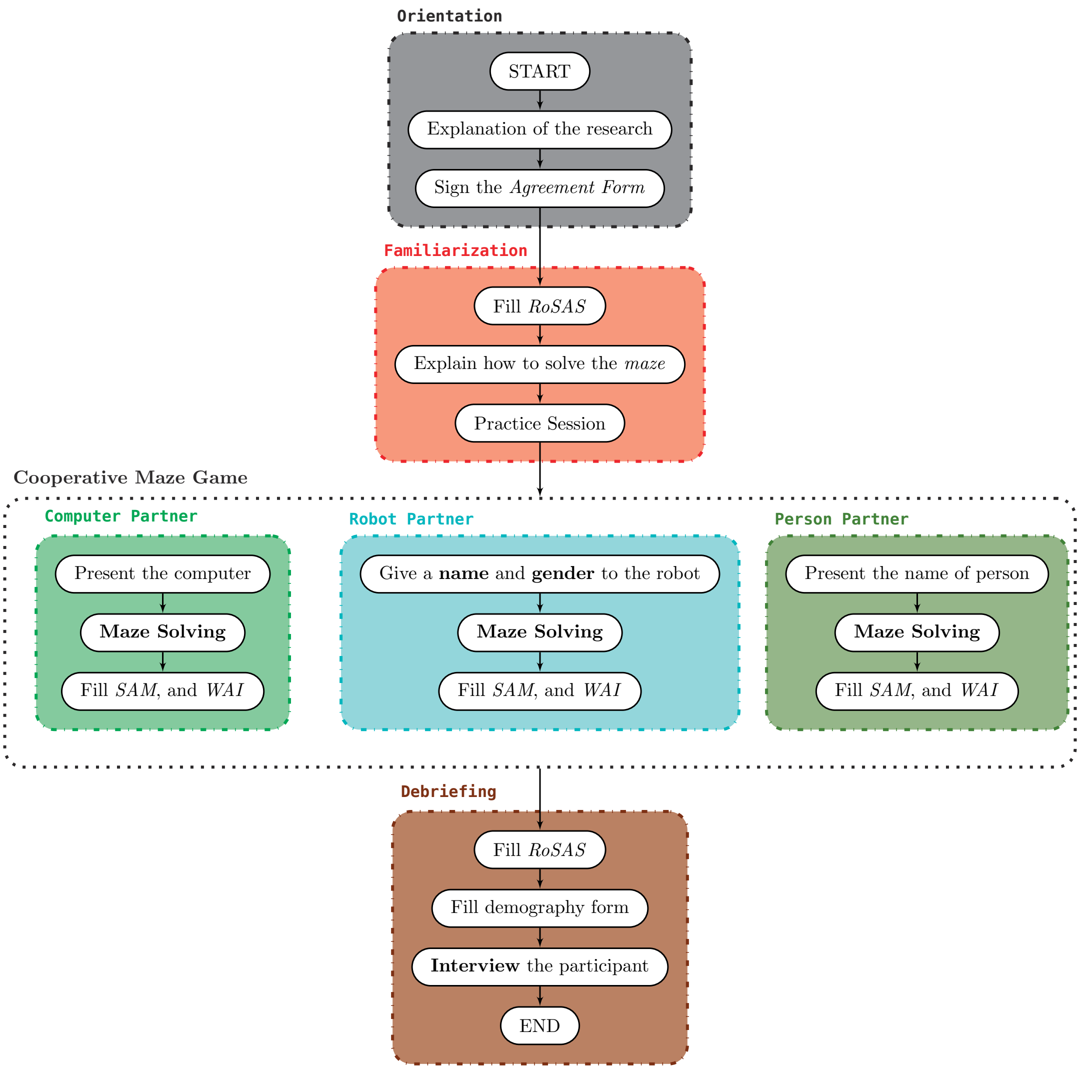


FIGURE 7.6: Flow of the interaction experiment.

Source: own

Once the participant got used to the interface, the ***collaborative maze*** phase began. A random partner (i.e. person, robot, or computer) was assigned to solve the maze together with the participant. The participant was asked to solve this maze ***five times*** with each partner. In general, the participants took less than 5 min to solve the maze five times with each partner. After each session was completed, the participant was asked to fill SAM and WAI keeping in mind what they experienced during the session (see Sections 2.5.2 and 2.6.1 to know more about SAM and WAI respectively). The participant was asked if they required to take a small break to rest before proceeding to the next session.

After solving the maze with all partners, the ***debriefing*** phase started. The participant was asked to fill the RoSAS questionnaire once again. Finally, they filled a demography form, followed by an interview. The demographic survey was asked at the end of the experiment to avoid any stereotype bias caused by gender or ethnicity (Shih, Pittinsky, and Ambady, 1999; Gibson, Losee, and Vitiello, 2014). Age was retrieved as a range to decrease the level of personal information collected.

7.3.2 Robot Interaction

The spherical robot was programmed with the seven animations studied in Chapter 6. These animations were shown during the thinking time of the robot, before it made a suggestion. To ensure that no hint regarding the maze was given by the robot, the animation was randomly chosen. These animations were used because they provided a combination of emotion-like expressions and informational expressions. In this sense, the participants were free to attach any meaning to the reaction presented by the robot.

On the other hand, previous researches have shown that people tend to unconsciously anthropomorphize computers and apply social rules (Nass and Moon, 2000), while names and genders closer to the user's inner group change the perception toward the robot (Eyssel and Kuchenbrandt, 2011; Eyssel et al., 2012). In this regard, participants were asked to give a name and gender of their liking to the ***robot*** before interacting with it to study their first impressions toward it. Moreover, this approach was also used by Onchi and Lee (2019) to study motion in the spherical robot. Similarly, when interacting with the other partners, the participants addressed the ***person*** by their first name, while referring to the ***computer*** as 'computer'.

7.3.3 Physiological Data Measurement

The participant was asked to wear a custom-developed smart bracelet (see Section 5) to measure the ***heart-rate***, ***electrodermal activity (EDA)***, and the movement of the hand. This information was used to validate the subjective surveys with unconscious reactions from the participant. In particular, this data was used to analyze the changes in arousal[4] over time during the activity.

To keep the information collected by the smart bracelet as clean as possible, the experimental room had curtains that blocked the view to outside distractions. Furthermore, the participants were asked to wear the device on their non-dominant hand and place it on top of the table in a relaxed manner. The participants wore the device during the practice session to get accustomed to it and the data collected during this period was used to calibrate the device. Finally, the time between interactions was filled with the survey, which allowed the biometric data to settle to baseline levels. Nevertheless,

[4]See Section 2.5.3 for more details on the relationship between arousal and physiological measurements.

FIGURE 7.7: Participant interacting with the robot while solving the collaborative maze game.

Source: own

external factors outside the control of the researchers made some of the data too noisy to be analyzed. This is addressed in Section 8.4: Limitations.

7.3.4 Health Considerations

Because the experiment was conducted during the pandemic of COVID-19, several measures to prevent the spread of pandemic diseases were conducted, according to the guidelines of the University of Tsukuba (2021).

First, the instructor measured the temperature of the participant to ensure that it was within the accepted body temperature range. Also, the experiment was conducted in a well ventilated room, and a distance of 2 m was kept between the instructor and the participant at all times. All the equipment and furniture used during the experiment was properly disinfected with alcohol before and after the experiments. Both the participant and the instructor wore facial masks throughout the whole experiment.

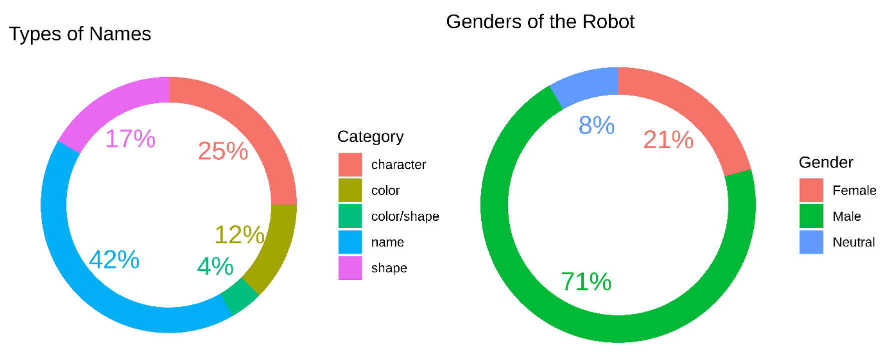


FIGURE 7.8: Types of names and genders given to the robot.
Source: own.

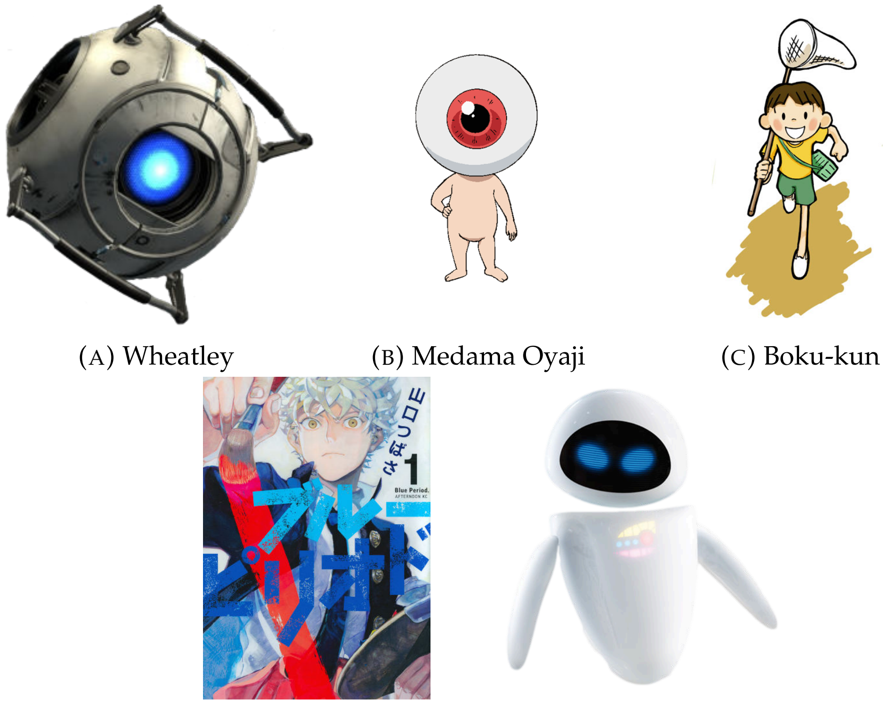
7.4 Results

As introduced in Section 7.3, each participant was asked to play a cooperative game with a person, a computer, and the robot to study how the interaction with a minimalistic robot compares to those other agents. The person and the computer were used as extreme baselines for a complete living agent and an inanimate agent respectively. In particular, the aim was to see how engaging the interaction with the robot was.

7.4.1 Subjective Impressions

In the experiment, participants were asked to name the robot and give it a gender before doing the cooperative game. This was done without any prior knowledge of the robot to test the first impression cause by the design of the robot.

Among the names given to the robot 12% were related to the color of the robot. 17% referred to the geometry or shape of the robot as in being round or spherical. Another 25% took elements familiar with popular characters from animated series or video games and used those names. Finally, 42%



(A) Wheatley (B) Medama Oyaji (C) Boku-kun

(D) Yatora (E) EVE

FIGURE 7.9: Example of the characters used when naming the robot. *(A)* character from the video game *"Portal 2"*. *(B)* character from the franchise *"GeGeGe no Kitaro"*. *(C)* character from the video game *"My Summer Vacation"*. *(D)* character from the franchise *"Blue Period"*. *(E)* character from the movie *"Wall-E"*.
Source: (A) Combine Overwiki (2011), (B) GeGeGe no Kitaro Wiki (2021), (C) Denshinbashira (2016), (D) KODANSHA (2021), (E) Pixar Wiki (2021)

used a nickname either based on the perceived gender of the robot or an acquaintance of the participant.

Regarding the perceived gender of the robot, 21% thought of the robot as ***female***, citing reasons like *"she looks curious and looks around all the time"* or *"the shape and material (plastic) are really soft, which makes me think of a girl"*. On the other hand, 71% of the participants thought of the robot as ***male***, stating that *"his eye is blue, which is the color of boys"* and *"he is so energetic, moving around like a little boy"*. Finally, a 8% of the surveyed thought of the robot as ***gender neutral*** because *"robots don't have gender"* or *"I don't think we can assign a gender, everyone is gender neutral these days"*.

TABLE 7.1: Name and Gender of the Robot

Gender	Name	Category	Notes
female	シロちゃん *(Shiro-chan)*	color	*shiro* is Japanese for *white* and *chan* is a Japanese suffix denoting endearment
female	アオちゃん *(Ao-chan)*	color	*ao* is Japanese for *blue* and *chan* is a Japanese suffix denoting endearment
female	あい *(Ai)*	name	*ai* is Japanese for *love*
female	小熊 *(Xiaoxiong)*	name	*Xiaoxiong* is Chinese for *little bear*
female	ロロちゃん *(Roro-chan)*	name	*Roro* is a Japanese nickname and *chan* is a Japanese suffix denoting endearment
male	Mr. Ball	geometry	based on the shape of the robot
male	まるくん *(Maru-kun)*	geometry	*maru* is Japanese for *round* and *kun* is a Japanese suffix denoting endearment
male	タマちゃん *(Tama-chan)*	geometry	*tama* is Japanese for *round* and *chan* is a Japanese suffix denoting endearment
male	White Ball	color/geometry	based on the shape and color of the robot
male	Wheatley	character	based on a video game character (see Figure 7.9a)
male	目玉おやじ *(Medama Oyaji)*	character	based on an animated character (see Figure 7.9b)
male	ボクくん *(Boku-kun)*	character	based on video game character (see Figure 7.9c)
male	八虎 *(Yatora)*	character	based on an animated character (see Figure 7.9d)
male	EVE	character	based on a video game character (see Figure 7.9e)
male	Park	name	*Park* is an Korean surname
male	Andrea	name	*Andrea* is an Italian name
male	Bob	name	*Bob* is an American name
male	Li	name	*Li* is a common Chinese name for males
male	太郎 *(Tarou)*	name	*Tarou* is a common Japanese name for males
male	アキオ *(Akio)*	name	*Akio* is a Japanese name
neutral	しろ *(Shiro)*	color	*shiro* is Japanese for *white*
neutral	Bubbles	geometry	based on the shape

TABLE 7.2: Shapiro-Wilk's Normality Test of RoSAS Scores

Score	W	p
$competence_{before}$	0.98	0.846
$competence_{after}$	0.96	0.440
$warmth_{before}$	0.97	0.674
$warmth_{after}$	0.97	0.629
$discomfort_{before}$	0.96	0.453
$discomfort_{after}$	0.92	0.067

*significance level at 0.05

7.4.2 Robotic Social Attributes Scale

The ***Robotic Social Attributes Scale (RoSAS)***[5] measures the social attributes of robots in three dimensions: ***competence***, ***warmth***, and ***discomfort***.

A standardized Cronbach's α was calculated per dimension to verify the reliability of the survey. According to Carpinella et al. (2017), the reliability of ***RoSAS*** per factor is 0.84 for ***competence***, 0.91 for ***warmth***, and 0.82 for ***discomfort***. The reliability calculated in this experiment showed a Cronbach's α of 0.85 for ***competence***, 0.74 for ***warmth***, and 0.75 for ***discomfort***. Given that these values were within the acceptable range of 0.70 to 0.95 (Tavakol and Dennick, 2011), this survey was considered reliable for this thesis.

A Shapiro-Wilk's normality test, presented in Table 7.2, was performed to check the normality of the ***RoSAS*** scoring. The results did not reject the null hypothesis, therefore the scores were considered to be normally distributed.

A paired t-test (Table 7.3) was conducted to compare the ***competence***, ***warmth***, and ***discomfort*** scores of RoSAS before and after interacting with the robot. There was a positive, medium, and statistically significant difference in the perceived ***warmth*** toward the robot ***before*** ($\bar{X} = 4.0$, $SD = 0.9$) and ***after*** ($\bar{X} = 4.0$, $SD = 0.9$) interacting with it ($\Delta_{means} = 0.42$, 95% CI [0.1, 0.8], $t_{23} = 2.43$, $p = 0.023$ Cohen's $d = 0.51$, 95% CI [0.07, 0.94]). No

[5]Check Section 2.6.2 for more details on this survey.

TABLE 7.3: Paired T-Test of RoSAS when Interacting with the Robot

RoSAS	$\bar{X}_{before}$ (SD)	$\bar{X}_{after}$ (SD)	$t_{(23)}$	p
competence	4.5 (1.0)	4.8 (1.1)	1.71	0.100
warmth	4.0 (0.9)	4.4 (0.9)	2.43	***0.023**
discomfort	2.5 (1.0)	2.3 (1.0)	-1.51	0.144

*significance level at 0.05

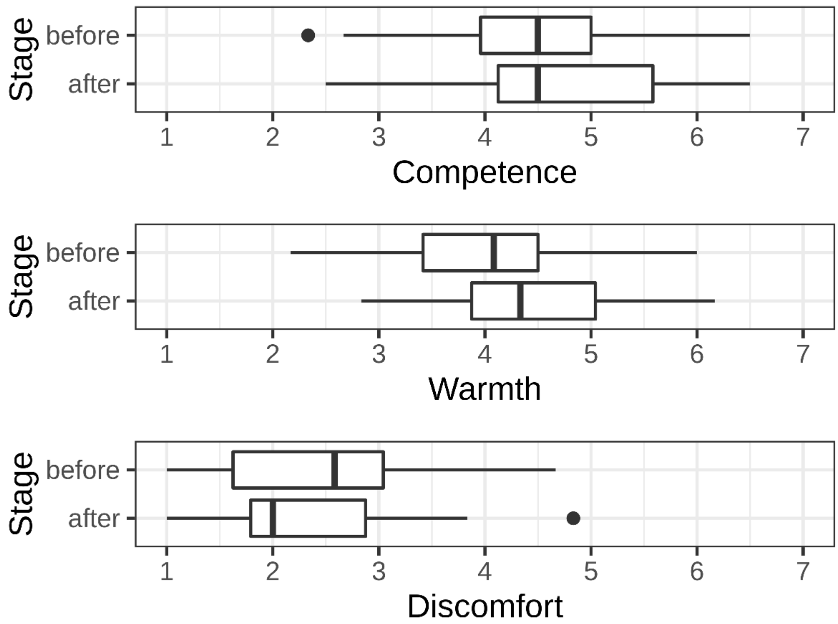


FIGURE 7.10: Box-plot of the scores of RoSAS on the social impression of the robot before and after the experiment.

Source: own.

significant differences were found for the other dimension. A visual comparison of the results is presented in Figure 7.10 and the effect size was labeled following Cohen (1988) recommendations.

7.4.3 Robot Gender and RoSAS

To check if the initial impression of the participant had and effect on the perceived gender of the robot, a t-test (Table 7.4) was conducted to compare the ***competence***, ***warmth***, and ***discomfort*** scores of RoSAS for the female and

TABLE 7.4: T-Test of RoSAS by Gender Before Interacting with the Robot

RoSAS	$\bar{X}_{female}$ (SD)	$\bar{X}_{male}$ (SD)	$t_{(8)}$	p
competence	4.2 (0.9)	4.6 (1.0)	-0.85	0.419
warmth	3.3 (0.8)	4.1 (0.9)	-1.88	0.098
discomfort	2.1 (0.7)	2.6 (1.1)	-1.23	0.245

*significance level at 0.05

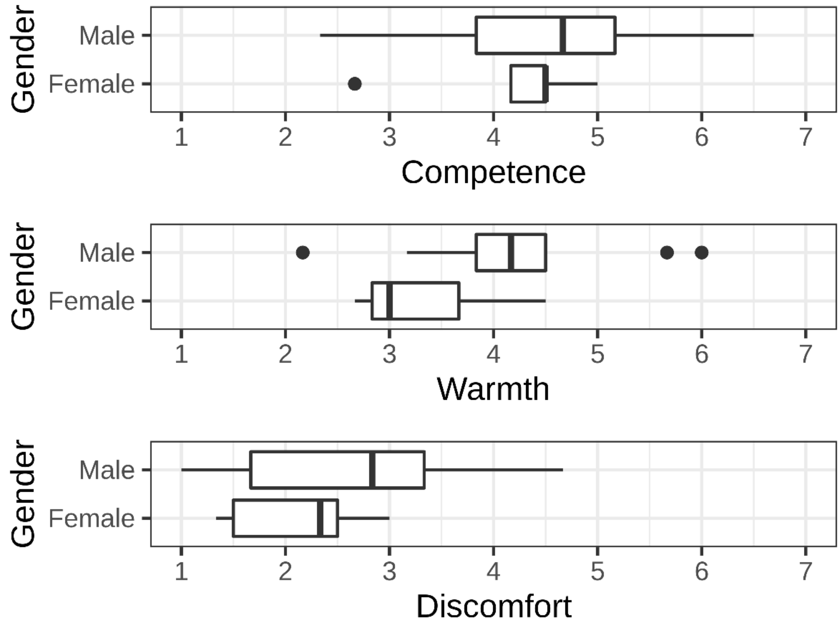


FIGURE 7.11: Box-plot of the scores of RoSAS on the social impression of the robot by gender before the experiment.
Source: own.

male robot gender before interacting with it. Neutral gender was omitted because of the low number of participants that thought of the robot as gender neutral. No significant differences were found for the scores of RoSAS. A visual comparison of the results is presented in Figure 7.11.

A similar result can be seen after the participants interacted with the robot. A t-test (Table 7.5) was conducted to compare the ***competence***, ***warmth***, and ***discomfort*** scores of RoSAS for the female and male robot gender after interacting with it. Neutral gender was omitted because of the low number

TABLE 7.5: T-Test of RoSAS by Gender After Interacting with the Robot

RoSAS	$\bar{X}_{female}$ *(SD)*	$\bar{X}_{male}$ *(SD)*	$t_{(8)}$	p
competence	4.9 (1.0)	4.6 (1.2)	0.62	0.552
warmth	4.3 (0.7)	4.4 (1.0)	-0.28	0.782
discomfort	1.9 (0.6)	2.4 (1.1)	-1.34	0.204

*significance level at 0.05

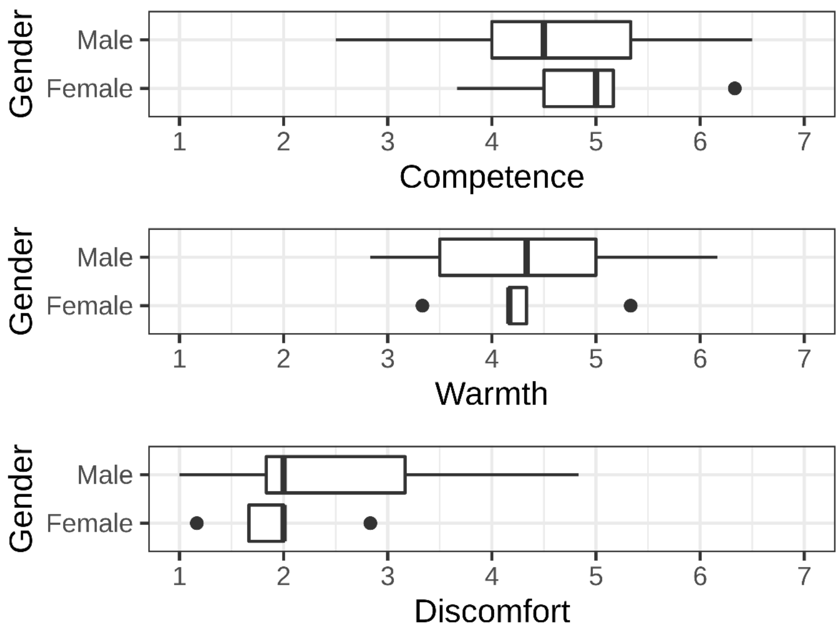


FIGURE 7.12: Box-plot of the scores of RoSAS on the social impression of the robot by gender after the experiment.
Source: own.

of participants that thought of the robot as gender neutral. No significant differences were found for the scores of RoSAS. A visual comparison of the results is presented in Figure 7.12.

Finally, a statistical analysis of the scores of RoSAS by robot name category was not conclusive because of the diverse number of categories and low number of participants in each category.

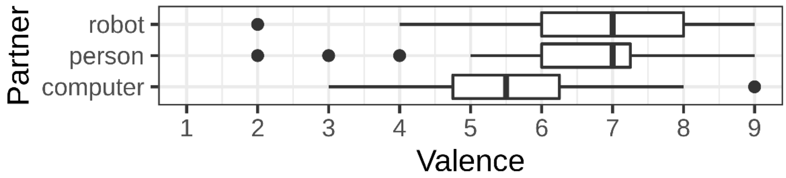


FIGURE 7.13: Box-plot of the valence during the interaction with different partners.

Source: own.

7.4.4 Self-Assessment Manikin

7.4.4.1 Valence

A Linear Mixed Model[6] (estimated using REML[7] and `nloptwrap`[8] optimizer) using the R package `lmer4` (Bates et al., 2015) was fitted to predict the effects of the ***partner*** on the ***valence*** of the interaction. The specification of the model was: `Valence ~ Partner + (1|Participant)`. Standardized parameters were obtained by fitting the model on a standardized version of the dataset, while confidence intervals and p-values were computed using the Wald approximation.

The model's total explanatory power is substantial ($R^2_{conditional} = 0.330$) and the part related to the fixed effects alone is $R^2_{marginal} = 0.123$. The model's intercept, corresponding to when the ***partner*** was the ***computer*** is $estimate = 5.46$ (95% CI $[4.78, 6.14]$, $t = 15.96$, $p < 0.001$). There were positive and statistically significant main effects when the ***partner*** was a ***person*** ($\beta = 1.13$, 95% CI $[0.28, 1.97]$, $t = 2.66$, $p = 0.010$) and when the ***partner*** was the ***robot*** ($\beta = 1.46$, 95% CI $[0.61, 2.30]$, $t = 3.45$, $p = 0.001$). A summary of the analysis is presented in Table 7.6 and its corresponding box-plot in Figure 7.13.

[6]See Section 2.7.1 for more details about this analysis method.
[7]**REML:** method for estimating variance in models with random effects.
[8]Alternative nonlinear optimizers from the `nloptr` package (Ypma et al., 2020).

TABLE 7.6: LMM of Valence by Partner

Predictors	***Estimate***	***CI***	***p***
computer (intercept)	5.46	4.78–6.14	<0.001
person	1.13	0.28–1.97	***0.010**
robot	1.46	0.61–2.30	***0.001**
Random Effects			
σ^2	2.14		
τ_{00}	0.66		
ICC	0.24		
$R^2_{marginal}$ / $R^2_{conditional}$	0.123 / 0.330		

*significance level at 0.05

TABLE 7.7: Pairwise Comparison of Valence by Partner

Contrasts	Δ_{mean}	***SE***	***CI***	p^a
person - computer	1.1	0.4	0.1–2.2	***0.032**
robot - computer	1.5	0.4	0.4–2.5	***0.004**
robot - person	0.3	0.4	-0.7–1.4	1.000

*significance level at 0.05
[a] Bonferroni adjustment

A post hoc analysis to test pairwise comparisons with Bonferroni adjustment using the R package `lsmeans` (Lenth, 2016) evidenced that there were statistically significant differences between interacting with the ***robot*** ($LSM = 6.9$, $SE = 0.3$, CI [6.2, 7.6]) compared to the ***computer*** ($LSM = 5.5$, $SE = 0.3$, CI [4.8, 6.1]) on the ***valence*** ($p = 0.004$); and the ***person*** ($LSM = 6.6$, $SE = 0.3$, CI [5.9, 7.3]) compared to the ***computer*** ($LSM = 5.5$, $SE = 0.3$, CI [4.8, 6.1]) on the ***valence*** ($p = 0.032$). No significant differences were found for the other combinations. Table 7.7 presents a summary of the analysis.

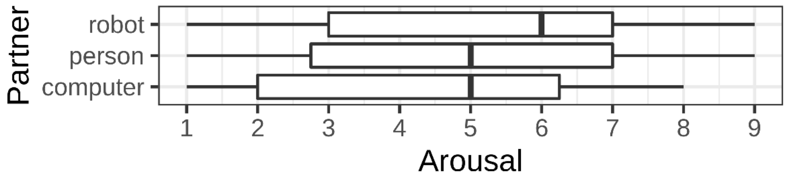


FIGURE 7.14: Box-plot of the arousal during the interaction with different partners.

Source: own.

7.4.4.2 Arousal

A Linear Mixed Model[9] (estimated using REML[10] and `nloptwrap`[11] optimizer) using the R package `lmer4` (Bates et al., 2015) was fitted to predict the effects of the ***partner*** on the ***arousal*** of the interaction. The specification of the model was: `Arousal ~ Partner + (1|Participant)`. Standardized parameters were obtained by fitting the model on a standardized version of the dataset, while confidence intervals and p-values were computed using the Wald approximation.

The model's total explanatory power is substantial ($R^2_{conditional} = 0.596$) and the part related to the fixed effects alone is $R^2_{marginal} = 0.016$. The model's intercept, corresponding to when the ***partner*** was the ***computer*** is $estimate = 4.42$ (95% CI $[3.42, 5.42]$, $t = 8.81$, $p < 0.001$). There were positive and statistically non-significant main effects when the ***partner*** was a ***person*** ($\beta = 0.42$, 95% CI $[-0.49, 1.32]$, $t = 0.92$, $p = 0.362$) and when the ***partner*** was the ***robot*** ($\beta = 0.75$, 95% CI $[-0.16, 1.66]$, $t = 1.65$, $p = 0.103$). A summary of the analysis is presented in Table 7.8 and its corresponding box-plot in Figure 7.14.

[9]See Section 2.7.1 for more details about this analysis method.

[10]**REML:** method for estimating variance in models with random effects.

[11]Alternative nonlinear optimizers from the `nloptr` package (Ypma et al., 2020).

TABLE 7.8: LMM of Arousal by Partner

Predictors	***Estimate***	***CI***	***p***
computer (intercept)	4.42	3.42–5.42	<0.001
person	0.42	-0.49–1.32	0.362
robot	0.75	-0.16–1.66	0.103
Random Effects			
σ^2	2.48		
τ_{00}	3.56		
ICC	0.59		
$R^2_{marginal} / R^2_{conditional}$	0.016 / 0.596		

*significance level at 0.05

7.4.4.3 Dominance

A Linear Mixed Model[12] (estimated using REML[13] and `nloptwrap`[14] optimizer) using the R package `lmer4` (Bates et al., 2015) was fitted to predict the effects of the ***partner*** on the ***dominance*** of the interaction. The specification of the model was: `Dominance ~ Partner + (1|Participant)`. Standardized parameters were obtained by fitting the model on a standardized version of the dataset, while confidence intervals and p-values were computed using the Wald approximation.

The model's total explanatory power is substantial ($R^2_{conditional} = 0.325$) and the part related to the fixed effects alone is $R^2_{marginal} = 0.023$. The model's intercept, corresponding to when the ***partner*** was the ***computer*** is $estimate = 5.08$ (95% CI $[4.30, 5.86]$, $t = 13.01$, $p < 0.001$). There were negative and statistically non-significant main effects when the ***partner*** was a ***person*** ($\beta = -0.71$, 95% CI $[-1.63, 0.21]$, $t = -1.54$, $p = 0.128$) and when the ***partner*** was the ***robot*** ($\beta = -0.25$, 95% CI $[-1.17, 0.67]$, $t = -0.54$, $p = 0.588$). A

[12]See Section 2.7.1 for more details about this analysis method.
[13]**REML:** method for estimating variance in models with random effects.
[14]Alternative nonlinear optimizers from the `nloptr` package (Ypma et al., 2020).

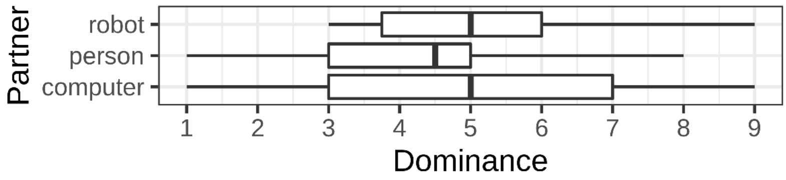


FIGURE 7.15: Box-plot of the dominance during the interaction with different partners.

Source: own.

TABLE 7.9: LMM of Dominance by Partner

Predictors	***Estimate***	***CI***	***p***
computer (intercept)	5.08	4.30–5.86	<0.001
person	-0.71	-1.63–0.21	0.128
robot	-0.25	-1.17–0.67	0.588
Random Effects			
σ^2	2.53		
τ_{00}	1.13		
ICC	0.31		
$R^2_{marginal} / R^2_{conditional}$	0.023 / 0.325		

*significance level at 0.05

summary of the analysis is presented in Table 7.9 and its corresponding box-plot in Figure 7.15.

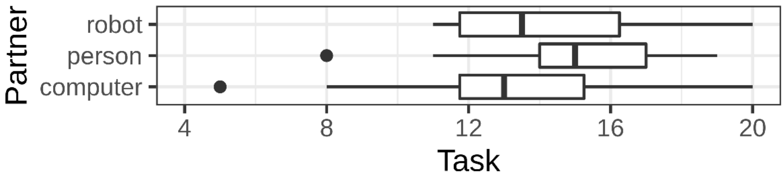


FIGURE 7.16: Box-plot of the task during the interaction with different partners.

Source: own.

7.4.5 Working Alliance Inventory

7.4.5.1 Task

A Linear Mixed Model[15] (estimated using REML[16] and `nloptwrap`[17] optimizer) using the R package `lmer4` (Bates et al., 2015) was fitted to predict the effects of the ***partner*** on the ***task*** of the interaction. The specification of the model was: `Task ~ Partner + (1|Participant)`. Standardized parameters were obtained by fitting the model on a standardized version of the dataset, while confidence intervals and p-values were computed using the Wald approximation.

The model's total explanatory power is substantial ($R^2_{conditional} = 0.547$) and the part related to the fixed effects alone is $R^2_{marginal} = 0.060$. The model's intercept, corresponding to when the ***partner*** was the ***computer*** is $estimate = 13.25$ (95% CI $[11.98, 14.52]$, $t = 20.84$, $p < 0.001$). There were positive and statistically significant main effects when the ***partner*** was a ***person*** ($\beta = 1.92$, 95% CI $[0.67, 3.16]$, $t = 3.07$, $p = 0.003$) and positive and statistically non-significant main effects when the ***partner*** was the ***robot*** ($\beta = 1.08$, 95% CI $[-0.16, 2.33]$, $t = 1.74$, $p = 0.087$). A summary of the analysis is presented in Table 7.10 and its corresponding box-plot in Figure 7.16.

[15]See Section 2.7.1 for more details about this analysis method.
[16]**REML:** method for estimating variance in models with random effects.
[17]Alternative nonlinear optimizers from the `nloptr` package (Ypma et al., 2020).

TABLE 7.10: LMM of Task by Partner

Predictors	***Estimate***	***CI***	***p***
computer (intercept)	13.25	11.98–14.52	<0.001
person	1.92	0.67–3.16	***0.003**
robot	1.08	-0.16–2.33	0.087
Random Effects			
σ^2	4.67		
τ_{00}	5.02		
ICC	0.52		
$R^2_{marginal} / R^2_{conditional}$	0.060 / 0.547		

*significance level at 0.05

TABLE 7.11: Pairwise Comparison of Task by Partner

Contrasts	Δ_{mean}	***SE***	***CI***	p^a
person - computer	1.9	0.6	0.4–3.5	***0.011**
robot - computer	1.1	0.6	-0.5–2.6	0.268
robot - person	-0.8	0.6	-2.4–0.7	0.565

*significance level at 0.05
[a] Bonferroni adjustment

A post hoc analysis to test pairwise comparisons with Bonferroni adjustment using the R package `lsmeans` (Lenth, 2016) evidenced that there were statistically significant differences between interacting with the ***person*** (LSM = 15.2, SE = 0.6, CI [13.9, 16.4]) compared to the ***computer*** (LSM = 13.3, SE = 0.6, CI [12.0, 14.5]) on the ***task*** (p = 0.011). No significant differences were found for the other combinations. Table 7.11 presents a summary of the analysis.

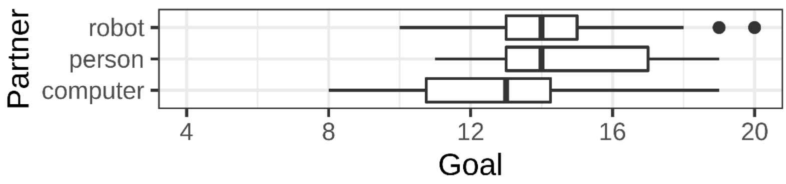


FIGURE 7.17: Box-plot of the goal during the interaction with different partners.

Source: own.

7.4.5.2 Goal

A Linear Mixed Model[18] (estimated using REML[19] and `nloptwrap`[20] optimizer) using the R package `lmer4` (Bates et al., 2015) was fitted to predict the effects of the ***partner*** on the ***goal*** of the interaction. The specification of the model was: `Goal ~ Partner + (1|Participant)`. Standardized parameters were obtained by fitting the model on a standardized version of the dataset, while confidence intervals and p-values were computed using the Wald approximation.

The model's total explanatory power is substantial ($R^2_{conditional} = 0.527$) and the part related to the fixed effects alone is $R^2_{marginal} = 0.083$. The model's intercept, corresponding to when the ***partner*** was the ***computer*** is $estimate = 12.96$ (95% CI $[11.92, 14.00]$, $t = 24.84$, $p < 0.001$). There were positive and statistically significant main effects when the ***partner*** was a ***person*** ($\beta = 1.79$, 95% CI $[0.73, 2.85]$, $t = 3.38$, $p = 0.001$) and when the ***partner*** was the ***robot*** ($\beta = 1.37$, 95% CI $[0.32, 2.43]$, $t = 2.59$, $p = 0.012$). A summary of the analysis is presented in Table 7.12 and its corresponding box-plot in Figure 7.17.

A post hoc analysis to test pairwise comparisons with Bonferroni adjustment using the R package `lsmeans` (Lenth, 2016) evidenced that there were statistically significant differences between interacting with the ***robot***

[18]See Section 2.7.1 for more details about this analysis method.
[19]**REML:** method for estimating variance in models with random effects.
[20]Alternative nonlinear optimizers from the `nloptr` package (Ypma et al., 2020).

TABLE 7.12: LMM of Goal by Partner

Predictors	***Estimate***	***CI***	***p***
computer (intercept)	12.96	11.92–14.00	<0.001
person	1.79	0.73–2.85	***0.001**
robot	1.37	0.32–2.43	***0.012**
Random Effects			
σ^2	3.37		
τ_{00}	3.16		
ICC	0.48		
$R^2_{marginal} / R^2_{conditional}$	0.083 / 0.527		

*significance level at 0.05

TABLE 7.13: Pairwise Comparison of Goal by Partner

Contrasts	Δ_{mean}	***SE***	***CI***	p^a
person - computer	1.8	0.5	0.5–3.1	***0.004**
robot - computer	1.4	0.5	0.1–2.7	***0.038**
robot - person	-0.4	0.5	-1.7–0.9	1.000

*significance level at 0.05
[a] Bonferroni adjustment

($LSM = 14.3$, $SE = 0.5$, CI [13.3, 15.4]) compared to the ***computer*** ($LSM = 13.0$, $SE = 0.5$, CI [11.9, 14.0]) on the ***goal*** ($p = 0.038$); and the ***person*** ($LSM = 14.8$, $SE = 0.5$, CI [13.7, 15.8]) compared to the ***computer*** ($LSM = 13.0$, $SE = 0.5$, CI [11.9, 14.0]) on the ***goal*** ($p = 0.004$). No significant differences were found for the other combinations. Table 7.13 presents a summary of the analysis.

7.4.5.3 Bond

A Linear Mixed Model[21] (estimated using REML[22] and `nloptwrap`[23] optimizer) using the R package `lmer4` (Bates et al., 2015) was fitted to predict the effects of the ***partner*** on the ***bond*** of the interaction. The specification of the

[21] See Section 2.7.1 for more details about this analysis method.
[22] **REML:** method for estimating variance in models with random effects.
[23] Alternative nonlinear optimizers from the `nloptr` package (Ypma et al., 2020).

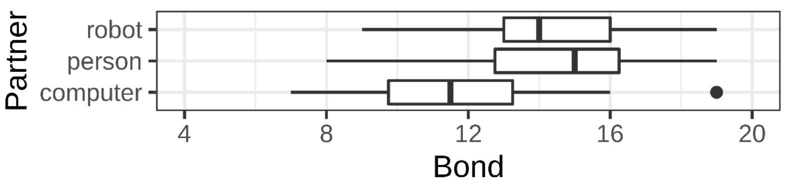


FIGURE 7.18: Box-plot of the bond during the interaction with different partners.

Source: own.

TABLE 7.14: LMM of Bond by Partner

Predictors	***Estimate***	***CI***	***p***
computer (intercept)	11.54	10.42–12.66	<0.001
person	2.96	1.74–4.18	***0.000**
robot	2.71	1.49–3.93	***0.000**
Random Effects			
σ^2	4.48		
τ_{00}	3.09		
ICC	0.41		
$R^2_{marginal}/R^2_{conditional}$	0.194 / 0.523		

*significance level at 0.05

model was: `Bond ~ Partner + (1|Participant)`. Standardized parameters were obtained by fitting the model on a standardized version of the dataset, while confidence intervals and p-values were computed using the Wald approximation.

The model's total explanatory power is substantial ($R^2_{conditional} = 0.523$) and the part related to the fixed effects alone is $R^2_{marginal} = 0.194$. The model's intercept, corresponding to when the ***partner*** was the ***computer*** is $estimate = 11.54$ (95% CI $[10.42, 12.66]$, $t = 20.55$, $p < 0.001$). There were positive and statistically significant main effects when the ***partner*** was a ***person*** ($\beta = 2.96$, 95% CI $[1.74, 4.18]$, $t = 4.84$, $p = 0.000$) and when the ***partner*** was the ***robot*** ($\beta = 2.71, 95\%$ CI $[1.49, 3.93]$, $t = 4.43$, $p = 0.000$). A summary of the analysis is presented in Table 7.14 and its corresponding box-plot in Figure 7.18.

TABLE 7.15: Pairwise Comparison of Bond by Partner

Contrasts	Δ_{mean}	*SE*	*CI*	p^a
person - computer	3.0	0.6	1.4–4.5	***0.000**
robot - computer	2.7	0.6	1.2–4.2	***0.000**
robot - person	-0.2	0.6	-1.8–1.3	1.000

*significance level at 0.05
[a] Bonferroni adjustment

A post hoc analysis to test pairwise comparisons with Bonferroni adjustment using the R package `lsmeans` (Lenth, 2016) evidenced that there were statistically significant differences between interacting with the ***robot*** ($LSM = 14.2$, $SE = 0.6$, CI [13.1, 15.4]) compared to the ***computer*** ($LSM = 11.5$, $SE = 0.6$, CI [10.4, 12.7]) on the ***bond*** ($p = 0.000$); and the ***person*** ($LSM = 14.5$, $SE = 0.6$, CI [13.4, 15.6]) compared to the ***computer*** ($LSM = 11.5$, $SE = 0.6$, CI [10.4, 12.7]) on the ***bond*** ($p = 0.000$). No significant differences were found for the other combinations. Table 7.15 presents a summary of the analysis.

7.4.6 Decision Making

7.4.6.1 Decision Time

A Linear Mixed Model[24] (estimated using REML[25] and `nloptwrap`[26] optimizer) using the R package `lmer4` (Bates et al., 2015) was fitted to predict the effects of the ***partner*** on the ***decision time*** of the interaction. The specification of the model was: `Decision Time ~ Partner + (1|Participant)`. Standardized parameters were obtained by fitting the model on a standardized version of the dataset, while confidence intervals and p-values were computed using the Wald approximation.

[24]See Section 2.7.1 for more details about this analysis method.
[25]**REML:** method for estimating variance in models with random effects.
[26]Alternative nonlinear optimizers from the `nloptr` package (Ypma et al., 2020).

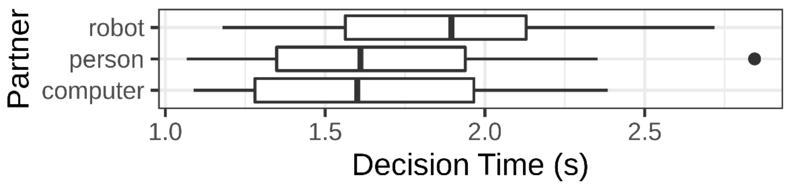


FIGURE 7.19: Box-plot of the decision time during the interaction with different partners.

Source: own.

TABLE 7.16: LMM of Decision Time by Partner

Predictors	***Estimate***	***CI***	***p***
computer (intercept)	1.64	1.47–1.82	<0.001
person	0.02	-0.14–0.18	0.811
robot	0.22	0.06–0.38	***0.007**
Random Effects			
σ^2	0.08		
τ_{00}	0.11		
ICC	0.59		
$R^2_{marginal} / R^2_{conditional}$	0.052 / 0.608		

*significance level at 0.05

The model's total explanatory power is substantial ($R^2_{conditional} = 0.608$) and the part related to the fixed effects alone is $R^2_{marginal} = 0.052$. The model's intercept, corresponding to when the ***partner*** was the ***computer*** is $estimate = 1.64$ (95% CI $[1.47, 1.82]$, $t = 18.60$, $p < 0.001$). There were positive and statistically non-significant main effects when the ***partner*** was a ***person*** ($\beta = 0.02$, 95% CI $[-0.14, 0.18]$, $t = 0.24$, $p = 0.811$) and positive and statistically significant main effects when the ***partner*** was the ***robot*** ($\beta = 0.22$, 95% CI $[0.06, 0.38]$, $t = 2.77$, $p = 0.007$). A summary of the analysis is presented in Table 7.16 and its corresponding box-plot in Figure 7.19.

A post hoc analysis to test pairwise comparisons with Bonferroni adjustment using the R package `lsmeans` (Lenth, 2016) evidenced that there were statistically significant differences between interacting with the ***robot*** ($LSM = 1.9$, $SE = 0.1$, CI [1.7, 2.0]) compared to the ***computer*** ($LSM = 1.6$,

TABLE 7.17: Pairwise Comparison of Decision Time by Partner

Contrasts	Δ_{mean}	***SE***	***CI***	p^a
person - computer	0.0	0.1	-0.2–0.2	1.000
robot - computer	0.2	0.1	0.0–0.4	***0.024**
robot - person	0.2	0.1	0.0–0.4	***0.045**

*significance level at 0.05
[a] Bonferroni adjustment

$SE = 0.1$, CI [1.5, 1.8]) on the ***decision time*** ($p = 0.024$); and the ***robot*** ($LSM = 1.9$, $SE = 0.1$, CI [1.7, 2.0]) compared to the ***person*** ($LSM = 1.7$, $SE = 0.1$, CI [1.5, 1.8]) on the ***decision time*** ($p = 0.045$). No significant differences were found for the other combinations. Table 7.17 presents a summary of the analysis.

7.4.6.2 Follow Rate

A Linear Mixed Model[27] (estimated using REML[28] and `nloptwrap`[29] optimizer) using the R package `lmer4` (Bates et al., 2015) was fitted to predict the effects of the ***partner*** on the ***follow rate*** of the interaction. The specification of the model was: `Follow Rate ~ Partner + (1|Participant)`. Standardized parameters were obtained by fitting the model on a standardized version of the dataset, while confidence intervals and p-values were computed using the Wald approximation.

The model's total explanatory power is moderate ($R^2_{conditional} = 0.210$) and the part related to the fixed effects alone is $R^2_{marginal} = 0.032$. The model's intercept, corresponding to when the ***partner*** was the ***computer*** is $estimate = 86.80$ (95% CI $[84.77, 88.83]$, $t = 85.25$, $p < 0.001$). There were negative and statistically non-significant main effects when the ***partner*** was a ***person*** ($\beta = -0.48$, 95% CI $[-3.07, 2.12]$, $t = -0.37$, $p = 0.715$) and when the ***partner***

[27]See Section 2.7.1 for more details about this analysis method.
[28]**REML:** method for estimating variance in models with random effects.
[29]Alternative nonlinear optimizers from the `nloptr` package (Ypma et al., 2020).

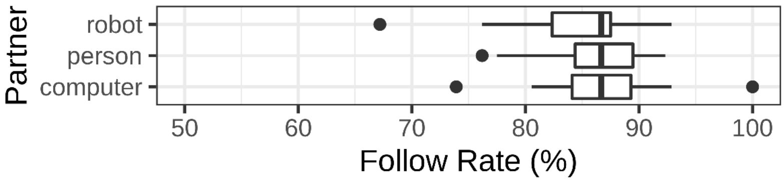


FIGURE 7.20: Box-plot of the follow rate during the interaction with different partners.

Source: own.

TABLE 7.18: LMM of Follow Rate by Partner

Predictors	***Estimate***	***CI***	***p***
computer (intercept)	86.80	84.77–88.83	<0.001
person	-0.48	-3.07–2.12	0.715
robot	-2.11	-4.70–0.49	0.110
Random Effects			
σ^2	20.32		
τ_{00}	4.56		
ICC	0.18		
$R^2_{marginal} / R^2_{conditional}$	0.032 / 0.210		

*significance level at 0.05

was the ***robot*** ($\beta = -2.11$, 95% CI $[-4.70, 0.49]$, $t = -1.62$, $p = 0.110$). A summary of the analysis is presented in Table 7.18 and its corresponding box-plot in Figure 7.20.

7.4.7 Physiological Data

7.4.7.1 Heart-Rate

A Linear Mixed Model[30] (estimated using REML[31] and `nloptwrap`[32] optimizer) using the R package `lmer4` (Bates et al., 2015) was fitted to predict the effects of the ***partner*** on the ***heart-rate*** of the interaction. The specification of the model was: `Heart-Rate ~ Partner + (1|Participant)`. Standardized

[30] See Section 2.7.1 for more details about this analysis method.
[31] **REML:** method for estimating variance in models with random effects.
[32] Alternative nonlinear optimizers from the `nloptr` package (Ypma et al., 2020).

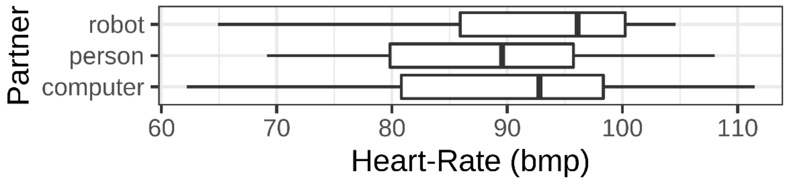


FIGURE 7.21: Box-plot of the heart-rate during the interaction with different partners.

Source: own.

TABLE 7.19: LMM of Heart-Rate by Partner

Predictors	***Estimate***	***CI***	***p***
computer (intercept)	89.80	85.25–94.35	<0.001
person	-1.13	-3.48–1.22	0.340
robot	1.88	-0.47–4.23	0.116
Random Effects			
σ^2	16.65		
τ_{00}	108.23		
ICC	0.87		
$R^2_{marginal} / R^2_{conditional}$	0.012 / 0.868		

*significance level at 0.05

parameters were obtained by fitting the model on a standardized version of the dataset, while confidence intervals and p-values were computed using the Wald approximation.

The model's total explanatory power is substantial ($R^2_{conditional} = 0.868$) and the part related to the fixed effects alone is $R^2_{marginal} = 0.012$. The model's intercept, corresponding to when the ***partner*** was the ***computer*** is $estimate = 89.80$ (95% CI $[85.25, 94.35]$, $t = 39.37$, $p < 0.001$). There were negative and statistically non-significant main effects when the ***partner*** was a ***person*** ($\beta = -1.13$, 95% CI $[-3.48, 1.22]$, $t = -0.96$, $p = 0.340$) and positive and statistically non-significant main effects when the ***partner*** was the ***robot*** ($\beta = 1.88$, 95% CI $[-0.47, 4.23]$, $t = 1.59$, $p = 0.116$). A summary of the analysis is presented in Table 7.19 and its corresponding box-plot in Figure 7.21.

TABLE 7.20: Pairwise Comparison of Heart-Rate by Partner

Contrasts	Δ_{mean}	***SE***	***CI***	p^a
person - computer	-1.1	1.2	-4.1–1.8	1.000
robot - computer	1.9	1.2	-1.0–4.8	0.353
robot - person	3.0	1.2	0.1–5.9	***0.042**

*significance level at 0.05
[a] Bonferroni adjustment

A post hoc analysis to test pairwise comparisons with Bonferroni adjustment using the R package `lsmeans` (Lenth, 2016) evidenced that there were statistically significant differences between interacting with the ***robot*** ($LSM = 91.7$, $SE = 2.3$, CI [87.0, 96.4]) compared to the ***person*** ($LSM = 88.7$, $SE = 2.3$, CI [84.0, 93.3]) on the ***heart-rate*** ($p = 0.042$). No significant differences were found for the other combinations. Table 7.20 presents a summary of the analysis.

7.4.7.2 Electrodermal Activity

In this research, there were two main questions regarding changes in EDA[33]: *"how does the conductance change during the interaction?"* and *"does following instructions from different partners affect the arousal of people?"*.

To answer the first question, *"how does the conductance change during the interaction?"*, the tonic component of EDA was extracted using the ***NeuroKit2*** toolbox for Python (Makowski et al., 2021). The interaction with each ***partner*** was isolated and each time series was interpolated to a length of 250 s to analyze the change in SCL[34] over the time of the interaction. Then, the mean of all the time series was calculated.

Figure 7.22 shows the average change in SCL over the whole interaction with each partner. Each value was time normalized to the shortest interaction

[33]**EDA:** Electrodermal activity.
[34]**SCL:** Skin Conductance Level.

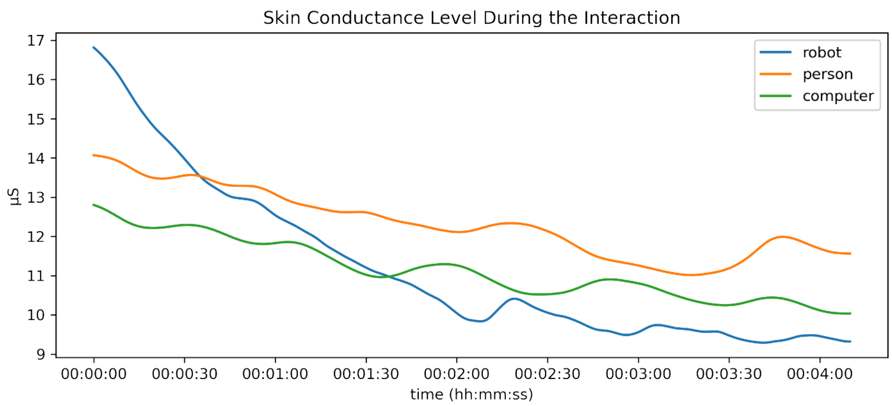


FIGURE 7.22: Change in skin conductance level when interacting with different partners.

Source: own.

time (around 4 minutes) to have a comparable range between participants and interactions. In other words, these signals represent the average of all 5 attempts to solve the maze with each partner. As seen in Figure 7.22, when participants interacted with the ***computer***, their SCL did not show much variation and steadily went from 12.80 ţS to 10.04 ţS. When participants interacted with the ***person***, their SCL slowly went from 14.07 ţS to 11.57 ţS, which could suggest a decrease in ***arousal*** over time. Finally, when participants interacted with the ***robot***, their SCL went from 16.81 ţS to 9.32 ţS at a faster pace than when interacting with the ***person*** or ***computer***. This might indicate that the majority of participants had an initial interest in the ***robot***, thus creating high arousal, that faded over time once the novelty subdued.

To answer the second question, *"does following instructions from different partners affect the arousal of people?"*, the skin conductance response (SCR) of the participants was analyzed. A window of 1 s before the epoch and 2 s after the epoch was extracted from the data. The epoch was defined as the instant when the suggested path from the partner was presented on the screen to the participant. The phasic component of EDA was extracted using the ***NeuroKit2*** toolbox for Python (Makowski et al., 2021).

A closer look to the SCR of participants when they received the suggested direction evidenced that there were no major reactions whether the interaction was with the robot, the person, or the computer (Figures 7.23, 7.24, and 7.25 respectively). The same can be said for the occasions when the participants chose a different path than what was suggested by their partners (Figures 7.26, 7.27, and 7.28). While each person had some specific reactions to the interaction (see Appendix C for the EDA analysis of each individual participant), noise recorded during the experiment and individual factors made it difficult to correctly assess a particular reaction to those events.

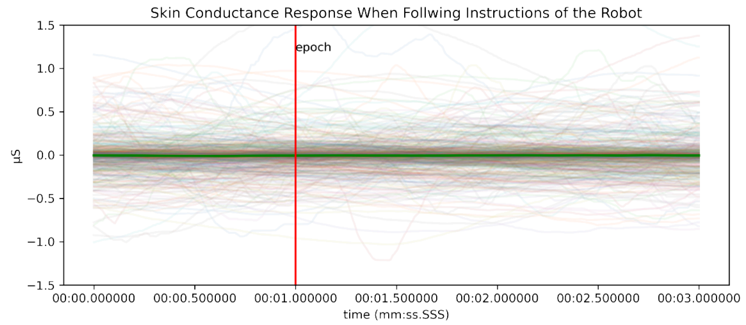


FIGURE 7.23: Change in skin conductance response when following the instructions of the robot. The epoch is the moment when the robot shows the suggestion on the screen.

Source: own.

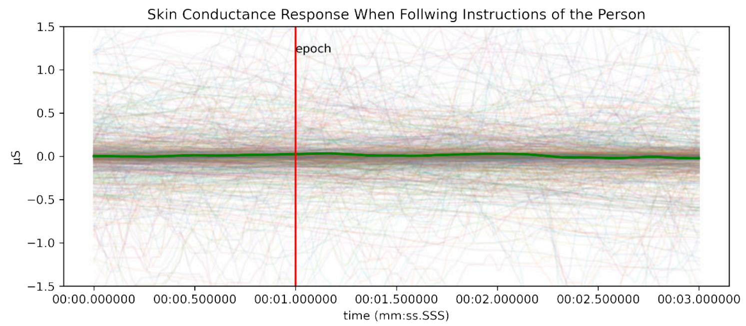


FIGURE 7.24: Change in skin conductance response when following the instructions of the person. The epoch is the moment when the person shows the suggestion on the screen.

Source: own.

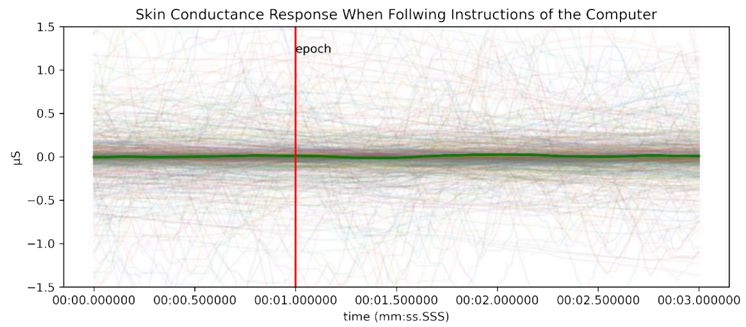


FIGURE 7.25: Change in skin conductance response when following the instructions of the computer. The epoch is the moment when the computer shows the suggestion on the screen.

Source: own.

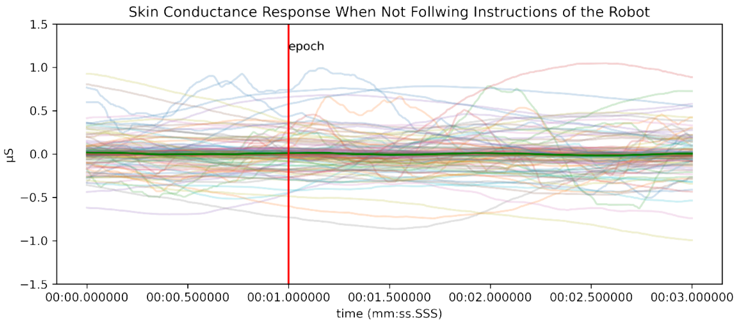


FIGURE 7.26: Change in skin conductance response when not following the instructions of the robot. The epoch is the moment when the robot shows the suggestion on the screen.

Source: own.

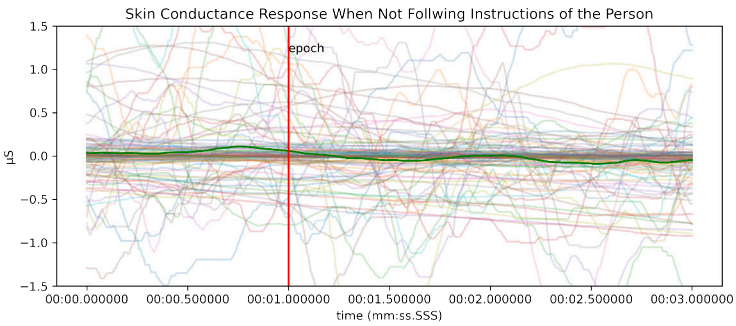


FIGURE 7.27: Change in skin conductance response when not following the instructions of the person. The epoch is the moment when the person shows the suggestion on the screen.

Source: own.

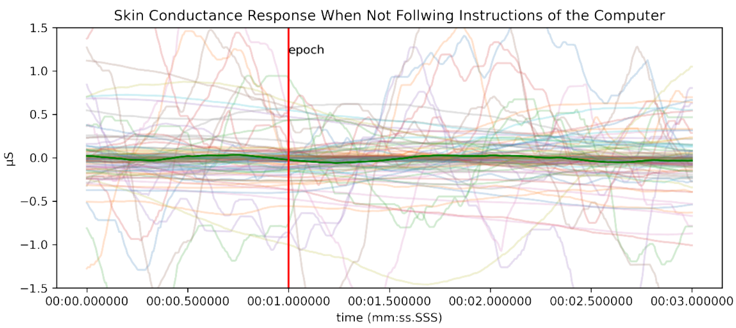


FIGURE 7.28: Change in skin conductance response when not following the instructions of the computer. The epoch is the moment when the computer shows the suggestion on the screen.

Source: own.

7.5 Discussion

This section divides into subsections several discussion points made from the results obtained in this experiment.

7.5.1 Social Robot

Before the participants interacted with the robot, they were asked to provide a name and a gender to the robot. While most of the participants suggested a name based on the robot's physical appearance (e.g. color, shape), some named it after a robotic character. These participants stated that they remembered watching or playing games that included robots, thus they provided that name. Indeed, it seems that one's own experience plays an important role on how people will interact with the robot.

On the other hand, some participants thought of the robot as male due to the color of its eye (Deep Sky Blue), while some others thought of the robot as female for the same reason. Likewise, some participants thought that the robot was male because of its *"quick and energetic movements"*, while some others gave the same reason to say it was female. Finally, some participants considered the robot to be gender neutral because *"robots do not have gender"*, or *"everyone is gender neutral these days"*. Therefore, a robot that does not show any significant detail regarding its gender is subject to the person's own experience regarding of what constitutes a behavior expected to that gender.

The results from the ***Robotic Social Attributes Scale (RoSAS)*** showed that the first impression toward the robot was that it was ***somewhat competent*** ($\bar{X} = 4.5$, $SD = 1.0$), with a ***neutral warmth*** ($\bar{X} = 4.0$, $SD = 0.9$), and caused ***low discomfort*** ($\bar{X} = 2.5$, $SD = 1.0$). This impression did not change much after interacting with the robot, although slight statistically significant increase in ***warmth*** ($\Delta_{means} = 0.42$), and a slight statistically non-significant increase in ***competence*** ($\Delta_{means} = 0.28$) and a slight statistically non-significant decrease in ***discomfort*** ($\Delta_{means} = -0.21$) did appear, from before the interaction to after the interaction. It might be possible that the more time people spend with the robot, the more used to and comfortable they will feel. These changes follow the same trend observed by Paetzel, Perugia, and Castellano (2020), where repeated interactions with a social robot in a collaborative game setting lowered the perceived discomfort and thread of the robot.

However, the results from ***RoSAS*** should be treated with care. It is possible that this evaluation fails to categorize robots that are non-humanoid in shape. As evidenced by the Cronbach's α (0.85 for competence, 0.74 for warmth, and 0.75 for discomfort), the reliability of the survey was lower that the original study by Carpinella et al. (2017) (see Section 7.4.2). Furthermore, there were items inside each category that were complete opposite to the other items of the same factor. For example, ***discomfort*** included ***strange*** as one of the items. Most of the participants regarded the robot as "*strange*" due to its unique shape, but did not feel discomfort toward it. Although Stroessner and Benitez (2018) showed high reliability of this test[35] when evaluating "*machine-like*" robots, the results from this thesis highlighted the opposite.

7.5.2 Emotional Impressions

If we imagine playing collaborative games with a person, a computer, and a robot, it seems clear that different partners will give us different experiences. However, is this difference significant when all partners use the same

[35] competence: $\alpha = .82 - .93$, warmth: $\alpha = .89 - .93$, and discomfort: $\alpha = .82 - .90$

collaborative strategy and the only variable is who we think we are playing against?

The ***Self-Assessment Manikin (SAM)*** survey showed that the ***emotional valence*** of the participants changed when interacting with different partners (see Section 7.4.4.1). In particular, participants evaluated the interaction with the ***computer*** as ***neutral*** ($LSM = 5.5$, $SE = 0.3$, CI [4.8, 6.1]). In contrast, when the participants interacted with the ***robot*** ($LSM = 6.9$, $SE = 0.3$, CI [6.2, 7.6]) and the ***person*** ($LSM = 6.6$, $SE = 0.3$, CI [5.9, 7.3]), they felt a slight positive experience. This ***positive valence*** of the participants when interacting with the ***robot*** and the ***person*** were statistically the same, with the robot getting a higher average score, meaning that an interactive social robot can provide a similar positive experience as a person. During the interview, some participants commented that they could not believe how much of a difference they felt when playing against the computer and the robot. Even though *"both are programmed machines"*, the robot created a more pleasant experience and they felt engaged during the activity.

In contrast, the perceived ***arousal*** of the interaction was the same for all partners, scoring around 5 points (neutrally aroused) on the SAM scale (see Section 7.4.4.2). While the subjective impression did not show any difference, a contrasting picture can be seen on the EDA during the activity (see Section 7.4.7.2). Following on the research that link skin conductance to arousal (Pakarinen, Pietila, and Nieminen, 2019), it is possible to see that the participants felt more aroused when interacting with the robot (SCL = 16.81 ţS) at the beginning of the game, compared to the person (SCL = 14.07 ţS) and the computer (SCL = 12.80 ţS). When looking at the change in skin conductance level over the course of the activity, this arousal decreased by the end of the activity (robot: 9.32 ţS, person: 11.57 ţS, computer: 10.04 ţS), which could mean that the participants felt initially excited to interact with the robot, as this was a unique experience for many of them, and later got used to interacting with it. Thus, the initial spike could be due to the novelty of the experience. On the other hand, the SAM was measured after the end of each game, therefore it was a snapshot at the end of the activity, where participants felt calmed and used to collaborating with each partner. This could explain why there were no apparent differences in the arousal scores of the Self-Assessment Manikin.

Likewise, the results from the heart-rate of the participants during the interaction show that, in average, participants had a higher beats per minute when interacting with the ***robot*** ($LSM = 91.7$, $SE = 2.3$, CI [87.0, 96.4]) compared to the ***person*** ($LSM = 88.7$, $SE = 2.3$, CI [84.0, 93.3]) and ***computer*** ($LSM = 89.8$, $SE = 2.3$, CI [85.1, 94.5]). Indeed, this increase in heart-rate could be related to the changes in EDA during the activity, further corroborating that interacting with the robot created a novel experience for the participants that caused arousal. Whether this increased arousal can be maintained or not in a long-term relationship with the robot will require a more extensive research on long-term interactions with social robots, like the ones from Kidd and Breazeal (2008).

Finally, the perceived ***dominance*** of the participant toward the partners did not present any significant differences. Their scores were around a neutral (5 points) dominance level.

7.5.3 Social Interaction

For the results related to the collaborative game, the WAI evidenced that participants felt that the person was more aware of the ***task*** than the computer (see Section 7.4.5.1). However, it was interesting to see that the scores for the robot ($LSM = 14.3$, $SE = 0.6$, CI [13.1, 15.6]) covered the range of both the person ($LSM = 15.2$, $SE = 0.6$, CI [13.9, 16.4]) and the computer ($LSM = 13.3$, $SE = 0.6$, CI [12.0, 14.5]), which could signify that some participants considered the robot to be as aware of the task as the person, while some others felt that it was no different than the computer. Whether the robot is considered the same as the computer or person depends on each participant.

An interesting result was found for the ***goal*** dimension between partners. According to the survey, the participants felt that the robot ($LSM = 14.3$, $SE = 0.5$, CI [13.3, 15.4]) and the person ($LSM = 14.8$, $SE = 0.5$, CI [13.7, 15.8]) had better awareness of the goal that the computer ($LSM = 13.0$, $SE = 0.5$, CI [11.9, 14.0]). This was unexpected as the solving method[36] for the maze was the same regardless of which partner the participants played with. The interesting part came from the interviews, where several participants indicated that, even though all partners made mistakes during the came, they expected that the computer would be correct all the time, thus they felt strongly against its mistakes. In contrast, when the robot made a mistake it was forgivable. Some participants felt that the mistake was even on purpose because of *"the playful nature of the robot"*. The case where the partner was the person, mistakes were understandable as both parties did not have a complete awareness of the maze.

On the other hand, the bonding between the participant and each partner showed the most change. While the participants felt neutral bonding with the computer ($LSM = 11.5$, $SE = 0.6$, CI [10.4, 12.7]), both the robot ($LSM = 14.2$, $SE = 0.6$, CI [13.1, 15.4]) and the person ($LSM = 14.5$, $SE = 0.6$, CI [13.4, 15.6]) partners did create some bonding during the experience. A participant even expressed that they *"could not believe how much of a difference it would*

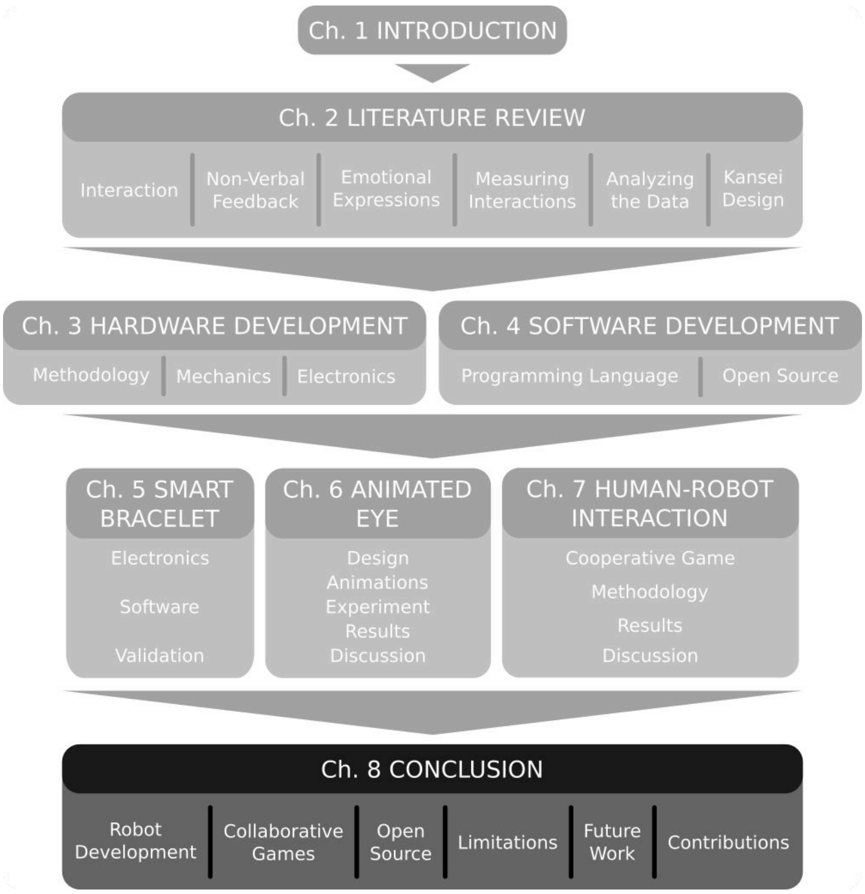
[36]See Section 7.2 for a detailed explanation on the underlying algorithm used to solve the maze.

make playing the game with the robot and the computer". It is worth mentioning that the level of bonding with the robot and the person were almost the same. This could imply that having an embodied agent giving non-verbal feedback can create a similar connection as interacting with a person. Of course, the interaction with both the person and robot should be considered as interacting with someone met for the first time.

Finally, the participants took, on average, 0.2 s more time to make a decision when interacting with the robot compared to the person, and 0.2 s more time compared to the computer. This delay could be due to the participants trying to understand the meaning behind the actions of the robot after a suggestion was made. As some participants stated, they *"wanted to know if there was any connection between how the robot behaved and the suggested path"*. Regardless of the time required to make a decision, the participants followed the direction 86.3% of the time on average. This was expected as the mistake rate when solving the maze was fixed at 20% regardless which partner was giving the instructions.

Chapter 8

Conclusions



This chapter presents the final ***conclusions*** of this thesis. It divides the conclusions into the ***development of the robot***, ***cooperative games with the robot***, and ***open source*** development. It finishes with the ***limitations*** of the study, ***future work***, and ***contributions*** to the scientific community.

8.1 Robot Development

The goal of this thesis was to research what are the minimal means to design a social robot that people feel engaged with. To that end, building upon the robot developed by Onchi and Lee (2019), several improvements were made and an interaction study was conducted.

8.1.1 Light Animations

The single-eyed spherical robot created by Onchi and Lee (2019) showed that emotion-like feedback using only movement is capable of expressing a wide range of emotions and this motion feedback on non-humanoid robots can improve the training session compared to training a static robot.

From there, this research studied how adding light animations affected the emotion understood. As mentioned in Section 6.6, adding light animations raised the emotional valence of the behavior of the robot. Moreover, it also raised the perceived arousal of the robot. This means we can use lights to regulate the arousal and valence of the emotion being expressed more granularly.

Therefore, rather than using a display to show detailed animations, it is possible to create interactive robots and change their expressions using simpler and economic methods like an array of colored LED.

8.2 Human-Robot Interaction

The second part of the study focused on how the robot compared to other interacting agents, like humans or computers. As discussed in Section 7.5, the overall results evidenced that people prefer to interact with physical beings that can express some type of feedback during the interaction. The performance of the robot was comparatively similar to when people interacted with another person, while the interaction with the computer was categorized as emotionally neutral and no social bonding happened.

This highlighted the importance of creating physical agents as the social gateway to technology. Instead of using a technology as an inert tool, it is possible to use a minimalistic proxy, like a non-humanoid robot, that can react according to the inner state of the device. This interaction can be extended to virtual assistants that, up till now, are designed as disembodied voices. This agent does not require to have complex mechanics to still be considered as a social agent. Indeed, simple movements and light animations can elevate the social impression of a computer to that of a person. However, the animacy of the robot developed in this thesis was increased by its ability to maintain eye contact, through facial tracking, with the person. Although the tracking algorithm did not have high reliability, this side-effect played in favor in the social level of the robot. People did not felt like they were being watched as the robot only kept eye contact for a short time before shifting its focus to other fake positives. In addition, these unexpected movements helped increase the curiosity toward the robot, as it made it seem like an energetic pet that was aware of its surroundings.

Moreover, by designing a non-humanoid robot, people tended to treat the robot as a pet, rather than a scary device. This avoided the aversion from the Uncanny Valley, and encourage people to be more compassionate toward the robot. Likewise, adding a name to the robot may have created a higher level of bonding. Some people even used familiar names to consciously create a better connection with the robot. This connection and metal image of the robot as a pet led to people forgiving mistakes made by the robot, or considering those mistakes as joking behavior from the robot.

8.3 Open Source

The robot and measuring tools developed for this research used technology relatively young in the robotic community. While this technology has been modernized to take advantage of the current computing capabilities, it was necessary to create the necessary software to make it work correctly. This led to several open source modules being developed.

In this regard, six Go modules optimized for the Raspberry Pi platform were created. Half of them focused on controlling electronic sensors, while the other half made controlling peripherals easier. To the best of the authors' knowledge, no previous modules were available for Raspberry Pi that could use the Go programming environment to control those devices. At the time of publication of this thesis, some of the modules have been used by other developers for their own projects.

8.4 Limitations

Recording physiological data is useful to acquire unbiased information about the emotional state of a person. However, its analysis assume that the data collected is noise free. The ***smart bracelet*** developed to measure these data worked well in most of the cases, but it was prone to capture noisy signals due to the unexpected movement of the participants, which made some of the collected data unreliable to be analyzed. Better sensors might be required for a more in depth study of the physiological reaction of people when interacting with social robots, but, for this thesis, the data collected provided reasonable insights on the trends in the change of emotional state.

Furthermore, the information collected through the surveys assumed that the participants were honest in their answers and that they understood the questions being asked. It is possible that some participants found the task repetitive or were tired after doing the tasks, which may have affected the final results of the survey. To minimize this, each participant had a randomized order of partners. Likewise, there was an implicit trust on the language ability of the participants and they were free to choose between a Japanese or English version of the questionnaires. Moreover, the participants of this research belonged to a Japanese university context, which means that they had a higher degree of education and were exposed to advanced technologies. Therefore, the results of this research may be representative for a target group similar to university students or alumni.

On the other hand, this research was conducted during the onset of a global pandemic, which made face-to-face interactions challenging. This meant that some experiments had to be revised for online evaluation, while others had to be postponed until the state of emergency was lifted. Nevertheless, especial care was taken when conducting the experiments following the guidelines of the University of Tsukuba for the prevention of infectious diseases (University of Tsukuba, 2021).

8.5 Contributions

This research generated not only insights in the area of Human-Robot Interaction, but also useful tools available to the scientific and programming community. In particular, the following open source libraries were developed:

- `ads1x15`: which controls analog-to-digital converters (see Section 5.2.1).
- `lsm6`: which controls inertial motion unit sensors (see Section 5.2.2).
- `max3010x`: which controls heart-rate and SpO_2 sensors (see Section 5.2.3).
- `PiCam`: which enables continues sampling of a camera from a Raspberry Pi (see Section 4.2.4).

- `ring`: which controls RGB LEDs in a ring shape and adds layered animation functionality (see Section 4.2.3).
- `servo`: which controls several servo motors asynchronously on a Raspberry Pi (see Section 4.2.2).

The source code of the robot, as well as the developed modules, are released under the MIT License[1] to contribute to the literature in Social Robotics and for further use in the development of similar technologies. These source codes are shared under GitHub[2] and a printed version is attached to the Appendix (D.2–D.11) of this thesis.

8.6 Future Work

The following research studied how using non-verbal light animations in a spherical robot affected the emotion-like information being conveyed. The results go in line with previous studies on non-verbal expressions in robots (Terada, Yamauchi, and Ito, 2012; Terada, Takeuchi, and Ito, 2013; Wilms and Oberfeld, 2018). In particular, the idea that using light animations to control the level of arousal (Wilms and Oberfeld, 2018) and valence (Kaya and Epps, 2004) was supported. On the other hand, compared to previous studies, this research mainly focused on the visual shape of the light animation and did not test color changes. Because the relationship between emotional meaning of color is a broad one, it was out of the scope for this research. Therefore, a future research on color variation may provide more insights on how to control emotion-like expressions.

Likewise, this thesis studied how interacting with the spherical robot compared to interacting with a person and a computer. From the results,

[1] A permissive free license with very limited restrictions.
[2] https://github.com/cgxeiji

it was possible to see that people preferred interacting with the robot and the persons instead of the computer, even though all partners use the same method when giving instructions. This supports the idea that having a physical body can influence how people perceive the interacting agent (Scholl and Tremoulet, 2000; Bartneck et al., 2009a). Furthermore, it was possible to see that the perception of the robot changed after interacting with it, which could support the idea that familiarity with a robotic agent will change how people regard that particular agent (Paetzel, Perugia, and Castellano, 2020). While the comparison of the spherical robot with a human agent and a computer agent gave insights in the performance of the robot, future research is need in contrasting how this design compares to other social robots designed with a minimalistic approach. Nevertheless, there is still room for improvement in the design of the physical appearance, the inner mechanisms, and the interacting behavior of the spherical robot.

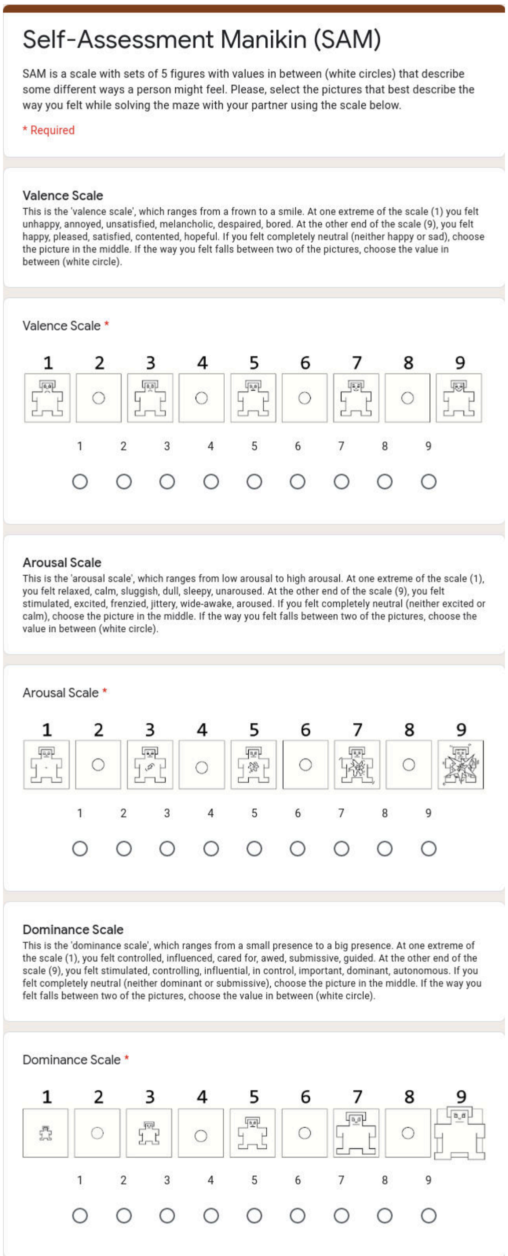
The author of this thesis hopes to design a version of the robot that could be used inside households and could make the upcoming technology more intuitive to use. Therefore, other manufacturing techniques, as well as mediums to express emotions, need to be researched. It is important to find the intersection between complexity and the information provided to create robots that are easy to use, without colliding with the Uncanny Valley or increasing its development cost.

Appendix A

Surveys

A.1 Self-Assessment Manikin: English Version

【資料 5.1】SAM（英語版）



Self-Assessment Manikin (SAM)

SAM is a scale with sets of 5 figures with values in between (white circles) that describe some different ways a person might feel. Please, select the pictures that best describe the way you felt while solving the maze with your partner using the scale below.

* Required

Valence Scale

This is the 'valence scale', which ranges from a frown to a smile. At one extreme of the scale (1) you felt unhappy, annoyed, unsatisfied, melancholic, despaired, bored. At the other end of the scale (9), you felt happy, pleased, satisfied, contented, hopeful. If you felt completely neutral (neither happy or sad), choose the picture in the middle. If the way you felt falls between two of the pictures, choose the value in between (white circle).

Valence Scale *

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9

Arousal Scale

This is the 'arousal scale', which ranges from low arousal to high arousal. At one extreme of the scale (1), you felt relaxed, calm, sluggish, dull, sleepy, unaroused. At the other end of the scale (9), you felt stimulated, excited, frenzied, jittery, wide-awake, aroused. If you felt completely neutral (neither excited or calm), choose the picture in the middle. If the way you felt falls between two of the pictures, choose the value in between (white circle).

Arousal Scale *

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9

Dominance Scale

This is the 'dominance scale', which ranges from a small presence to a big presence. At one extreme of the scale (1), you felt controlled, influenced, cared for, awed, submissive, guided. At the other end of the scale (9), you felt stimulated, controlling, influential, in control, important, dominant, autonomous. If you felt completely neutral (neither dominant or submissive), choose the picture in the middle. If the way you felt falls between two of the pictures, choose the value in between (white circle).

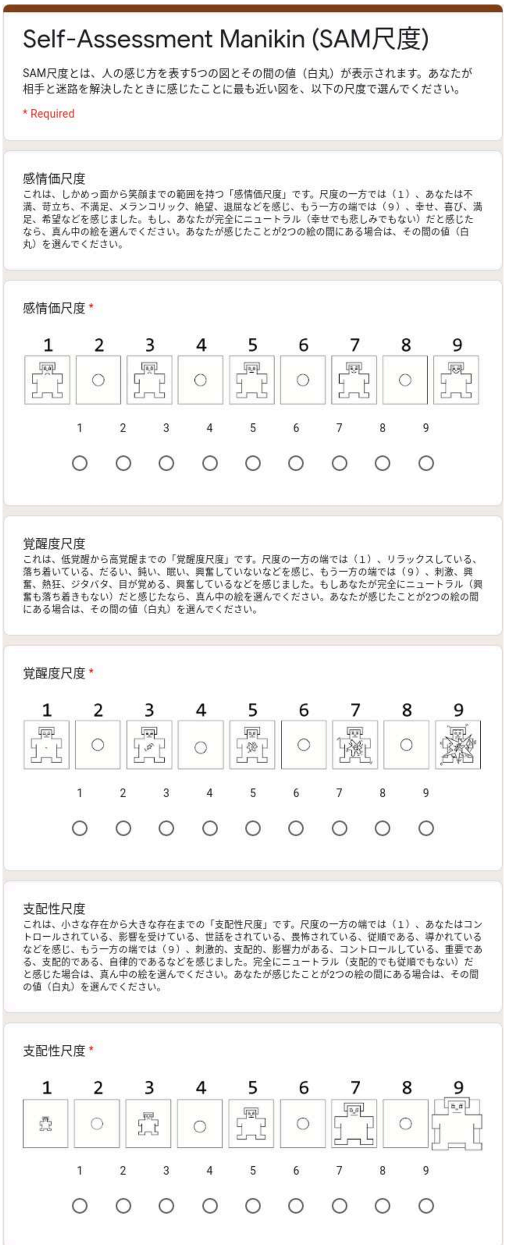
Dominance Scale *

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9

A.2 Self-Assessment Manikin: Japanese Version

【資料 5.2】SAM（日本語版）



Self-Assessment Manikin (SAM尺度)

SAM尺度とは、人の感じ方を表す5つの図とその間の値（白丸）が表示されます。あなたが相手と迷路を解決したときに感じたことに最も近い図を、以下の尺度で選んでください。

* Required

感情価尺度

これは、しかめっ面から笑顔までの範囲を持つ「感情価尺度」です。尺度の一方では（1）、あなたは不満、苛立ち、不満足、メランコリック、絶望、退屈などを感じ、もう一方の端では（9）、幸せ、喜び、満足、希望などを感じました。もし、あなたが完全にニュートラル（幸せでも悲しみでもない）だと感じたなら、真ん中の絵を選んでください。あなたが感じたことが2つの絵の間にある場合は、その間の値（白丸）を選んでください。

感情価尺度 *

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9

覚醒度尺度

これは、低覚醒から高覚醒までの「覚醒度尺度」です。尺度の一方の端では（1）、リラックスしている、落ち着いている、だるい、鈍い、眠い、興奮していないなどを感じ、もう一方の端では（9）、刺激、興奮、熱狂、ジタバタ、目が覚める、興奮しているなどを感じました。もしあなたが完全にニュートラル（興奮も落ち着きもない）だと感じたなら、真ん中の絵を選んでください。あなたが感じたことが2つの絵の間にある場合は、その間の値（白丸）を選んでください。

覚醒度尺度 *

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9

支配性尺度

これは、小さな存在から大きな存在までの「支配性尺度」です。尺度の一方の端では（1）、あなたはコントロールされている、影響を受けている、世話をされている、畏怖されている、従順である、導かれているなどを感じ、もう一方の端では（9）、刺激的、支配的、影響力がある、コントロールしている、重要である、支配的である、自律的であるなどを感じました。完全にニュートラル（支配的でも従順でもない）だと感じた場合は、真ん中の絵を選んでください。あなたが感じたことが2つの絵の間にある場合は、その間の値（白丸）を選んでください。

支配性尺度 *

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9

A.3 Robotic Social Attributes Scale: English Version

【資料 4.1】RoSAS（英語版）

RoSAS

The Robotic Social Attributes Scale (RoSAS) is a survey to measure how people think of robots. You will be presented with 18 adjectives that describe the robot in front of you. Please, rate your impression of the robot based on the scale below.

1 = definitely not associated
7 = definitely associated

* Required

You will be presented with 18 adjectives that describe the robot in front of you. On a scale of 1 to 7, how closely are the adjectives associated with the robot? *

	1 (definitely not associated)	2	3	4	5	6	7 (definitely associated)
Interactive	○	○	○	○	○	○	○
Reliable	○	○	○	○	○	○	○
Awkward	○	○	○	○	○	○	○
Competent	○	○	○	○	○	○	○
Happy	○	○	○	○	○	○	○
Dangerous	○	○	○	○	○	○	○
Capable	○	○	○	○	○	○	○
Compassionate	○	○	○	○	○	○	○
Social	○	○	○	○	○	○	○
Feeling	○	○	○	○	○	○	○
Responsive	○	○	○	○	○	○	○
Scary	○	○	○	○	○	○	○
Agressive	○	○	○	○	○	○	○
Awful	○	○	○	○	○	○	○
Strange	○	○	○	○	○	○	○
Knowledgable	○	○	○	○	○	○	○
Emotional	○	○	○	○	○	○	○
Organic	○	○	○	○	○	○	○

A.4 Robotic Social Attributes Scale: Japanese Version

【資料 4.2】RoSAS（日本語版）

RoSAS

Robotic Social Attributes Scale（RoSAS）は、人々がロボットをどのように考えているかを測定するための調査です。目の前のロボットを説明する18の形容詞が提示されます。以下の尺度に基づいてロボットの印象を評価してください。

1＝全く感じなかった
7＝かなり感じた

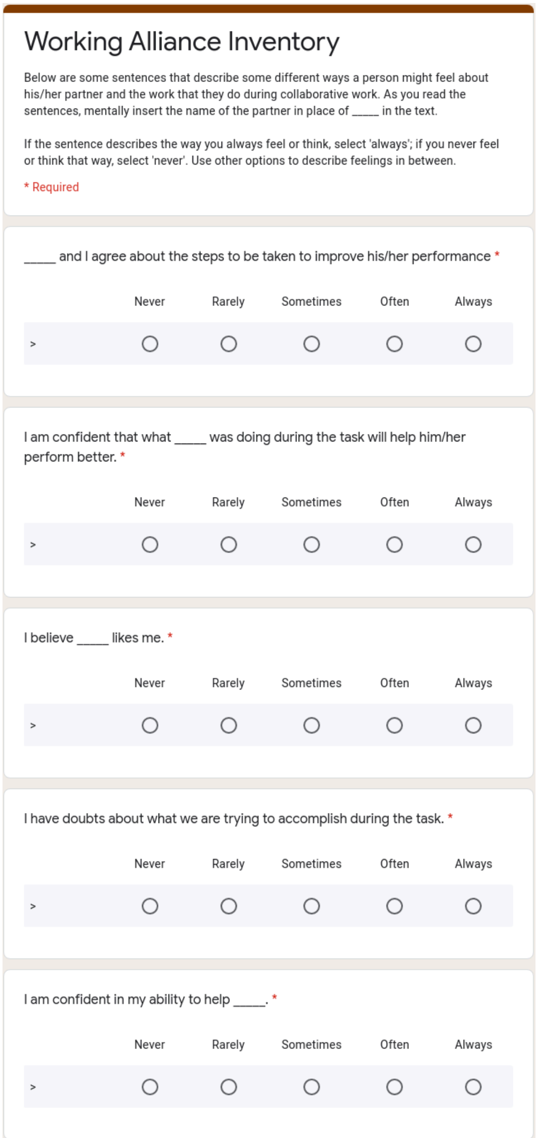
* Required

目の前のロボットを説明する18の形容詞が提示されます。以下の尺度に基づいてロボットの印象を評価してください。 *

	1（全く感じなかった）	2	3	4	5	6	7（かなり感じた）
攻撃的な	○	○	○	○	○	○	○
双方向的な	○	○	○	○	○	○	○
ぎこちない	○	○	○	○	○	○	○
適格な	○	○	○	○	○	○	○
敏感な	○	○	○	○	○	○	○
ひどい	○	○	○	○	○	○	○
多感な	○	○	○	○	○	○	○
危険な	○	○	○	○	○	○	○
思いやりのある	○	○	○	○	○	○	○
奇妙な	○	○	○	○	○	○	○
有能な	○	○	○	○	○	○	○
有機的な	○	○	○	○	○	○	○
情緒的な	○	○	○	○	○	○	○
信頼できる	○	○	○	○	○	○	○
おそろしい	○	○	○	○	○	○	○
幸せな	○	○	○	○	○	○	○
物知りな	○	○	○	○	○	○	○
社会的な	○	○	○	○	○	○	○

A.5 Working Alliance Inventory: English Version

【資料 6.1】WAI（英語版）



Working Alliance Inventory

Below are some sentences that describe some different ways a person might feel about his/her partner and the work that they do during collaborative work. As you read the sentences, mentally insert the name of the partner in place of ____ in the text.

If the sentence describes the way you always feel or think, select 'always'; if you never feel or think that way, select 'never'. Use other options to describe feelings in between.

* Required

____ and I agree about the steps to be taken to improve his/her performance *

	Never	Rarely	Sometimes	Often	Always
>	○	○	○	○	○

I am confident that what ____ was doing during the task will help him/her perform better. *

	Never	Rarely	Sometimes	Often	Always
>	○	○	○	○	○

I believe ____ likes me. *

	Never	Rarely	Sometimes	Often	Always
>	○	○	○	○	○

I have doubts about what we are trying to accomplish during the task. *

	Never	Rarely	Sometimes	Often	Always
>	○	○	○	○	○

I am confident in my ability to help ____. *

	Never	Rarely	Sometimes	Often	Always
>	○	○	○	○	○

【資料 6.1】WAI（英語版）

We are working toward mutually agreed upon goals. *

	Never	Rarely	Sometimes	Often	Always
>					

I enjoy working with ____. *

	Never	Rarely	Sometimes	Often	Always
>					

____ and I have a common perception of him/her goals. *

	Never	Rarely	Sometimes	Often	Always
>					

____ and I have built a mutual trust. *

	Never	Rarely	Sometimes	Often	Always
>					

____ and I have different ideas on what his/her real difficulties are. *

	Never	Rarely	Sometimes	Often	Always
>					

We agree about the kind of changes that would be good for ____. *

	Never	Rarely	Sometimes	Often	Always
>					

I think that ____ believes that the way we are working is useful. *

	Never	Rarely	Sometimes	Often	Always
>					

A.6 Working Alliance Inventory: Japanese Version

【資料 6.2】WAI（日本語版）

Working Alliance Inventory

以下は、共同作業中の相手とその作業について、人が感じるであろういくつかの異なる方法を記述する文章です。文章を読みながら、____ の代わりに相手の名前を心の中で入れてください。

文章が、あなたがいつも感じたり考えたりすることを表している場合は「いつも」を、決してそのように感じたり考えたりしない場合は「全くない」を選択します。その中間の気持ちを表す場合は、他の選択肢を使ってください。

* Required

____と私は、____の状況を改善するためにはどのようなステップを踏めばいいのか、意見が一致している。 *

	全くない	たまに	ときどき	ほとんど	いつも
>	○	○	○	○	○

課題の有用性について、____と私の意見が一致している。 *

	全くない	たまに	ときどき	ほとんど	いつも
>	○	○	○	○	○

____と私は、互いに好感を抱きあっている。 *

	全くない	たまに	ときどき	ほとんど	いつも
>	○	○	○	○	○

____と私の間で、課題から得られるものについての懸念がある。 *

	全くない	たまに	ときどき	ほとんど	いつも
>	○	○	○	○	○

私は、私の援助能力を信頼しきっている。 *

	全くない	たまに	ときどき	ほとんど	いつも
>	○	○	○	○	○

【資料 6.2】WAI（日本語版）

____と私は、目標を合意したうえで課題を行っている。 *

	全くない	たまに	ときどき	ほとんど	いつも
>	○	○	○	○	○

私は____と一緒に課題するのが楽しいと思っている。 *

	全くない	たまに	ときどき	ほとんど	いつも
>	○	○	○	○	○

課題における____の目標について、____と私の間で共通の認識がある。 *

	全くない	たまに	ときどき	ほとんど	いつも
>	○	○	○	○	○

____と私は、互いに信頼し合っている。 *

	全くない	たまに	ときどき	ほとんど	いつも
>	○	○	○	○	○

____と私は、____が抱えている問題についての考えが異なっている。 *

	全くない	たまに	ときどき	ほとんど	いつも
>	○	○	○	○	○

____と私は、____がどのように変わればよいか、十分理解を深めることができた。 *

	全くない	たまに	ときどき	ほとんど	いつも
>	○	○	○	○	○

私は、課題の方法は____にとって良い方法だと思っている。 *

	全くない	たまに	ときどき	ほとんど	いつも
>	○	○	○	○	○

Appendix B

Ethics

B.1 Ethics Approval

様式9

課題番号第　芸021-6　号
令和　3　年　9　月　27　日

研 究 倫 理 審 査 結 果 通 知 書

申請者(研究責任者)
李 昇姫　殿

芸術系長
野 中 勝 利
(公印省略)

令和3年7月15日付けで申請のあった研究倫理について、審査の結果、下記のとおり判定したので通知します。

記

1 課題名
人間とロボットのインタラクションにおける非言語フィードバックの評価

2 判 定
承 認

3 理 由

B.2 Research Instructions: Japanese Version

【資料 2.1】研究についての説明（日本語版）

研究についての説明

① **研究の説明**

ア. 本研究の目的は、動きと光でフィードバックを表現できる球体型・片目ロボットを訓練することによって、ロボットが人に与える信頼度と感情の変化の研究です。

イ. 最初に、ロボットに対する不安の程度を調べる RoSAS（Robotic Social Attributes Scale）の調査を行います。

ウ. 試験中のインタラクションを記録するために、手の動きと心拍数と電気皮膚反応を計測するスマートブレスレットを装着していただきます。スマートブレスレットの装着に違和感がある場合は、装着しないように申し出ることが可能です。

エ. 本実験の課題は、人、コンピュータ、ロボットの３者で一緒に迷路を解くことです。タブレットを用いて、迷路の道を選択するゲームです。本実験の前に十分な練習の時間があります。

オ. 課題は、５回の 3 セットで迷路を共同解決します。相手はセットごとにランダムに割り当てられます。解決相手は出口の位置を大まかに知っているだけで、あなたを出口まで案内しようとします。 受け取った指示に従うかどうかはあなた次第です。各セットの所要時間は約５分です。

カ. 各セット後に 2 つの評価を行います。ロボットとのインタラクションにおけるあなたの感情を測る SAM（Self-Assessment Manikin）や、ロボットとあなたの共同作業への満足度を測る WAI（Working Alliance Inventory）などのテストを行います。

キ. 最後に、もう一度 RoSAS（Robotic Social Attributes Scale）の調査を行います。又は、国籍、年齢、性別などについて回答してもらいます。

ク. その後、実験についての意見を聞かせてもらいます。

ケ. 休憩を取りたい、実験をやめたいという場合は、いつでも実験を中止することができますので研究分担者にお声かけください。

② **感染症拡大防止対策に関すること**

ア. 感染症予防のため、実験前に検温を行います。

イ. 実験は換気の良い部屋で行い、研究協力者との間には常に 2m の距離を保ちます。

ウ. 実験に使用するデバイスや家具はすべて消毒し、研究協力者と研究分担者は常にマスク及びフェイスシールドを着用します。

③ **倫理的配慮に関すること**

ア. 実験への同意の有無および得られた結果に関わらず、研究協力者が不利益を被ることはありません。

④ **本人の自由意思による同意であること**

ア. 研究協力者は、実験に協力しない自由があります。

⑤ **同意後も不利益を受けず随時撤回できること**

ア. 研究協力者は実験実施中でもいつでも実験協力の同意を撤回することができます。それによって、研究協力者が不利益を被ることはありません。

【資料 2.1】研究についての説明（日本語版）

⑥ **同意しない場合でも不利益を受けないこと**

ア. 研究協力者は実験協力に同意しない場合でも、研究協力者が特に不利益を被ることはありません。

⑦ **個人情報は保護されること**

ア. データは第三者によっては各個人を特定できない形で扱います。

イ. 個人を特定できるような形でデータを公表することはありません。

説明者	所属	人間総合科学研究科　感性認知脳科学専攻	
	氏名	ONCHI SUGUIMITZU Diego Eiji	㊞
	連絡先	s1930389@s.tsukuba.ac.jp	
研究責任者	所属	芸術系	
	氏名	李昇姫	㊞
	連絡先	lee.seunghee.gn@u.tsukuba.ac.jp	

この研究は筑波大学芸術系研究倫理委員会の承認を得て、研究協力者の皆様に不利益がないよう万全の注意を払って行われています。研究への協力に際してご意見ご質問などございましたら、気軽に研究責任者にお尋ね下さい。あるいは、芸術系研究倫理委員会までご相談下さい。

【電話：029-853-2571（体育芸術エリア支援室研究支援）e-mail：tg-kenkyurinri@un.tsukuba.ac.jp】

B.3 Research Instructions: English Version

【資料 2.2】研究についての説明（英語版）

Research Instructions

① **About the research**
  a. The purpose of this research is to study how a single-eyed spherical robot that can express feedback with movement and lights affects the interaction between humans and robots during interaction.
  b. First, you will be asked to fill the **Robotic Social Attributes Scale (RoSAS)**, a survey that measures how comfortable you are interacting with robots.
  c. You will be asked to wear a hand-motion, heart rate, and galvanic skin response smart-bracelet to record how you interact with the robot. If you feel uncomfortable wearing the smart-bracelet, you can request not to wear it.
  d. The task for this experiment is to solve a maze together with a person, computer, and the robot. You will use a tablet to solve the maze. You can practice this until you feel comfortable.
  e. You will solve the maze in 3 sets of 5 rounds. Your partner will be randomly assigned for each set. Your partner only has a general idea of where the exit is and will try to guide you to the exit. It is up to you if you want to follow the instructions you receive. Each set will take around 5 minutes to complete.
  f. After finishing each set, you will be asked to fill 2 surveys: (1) **Self-Assessment Manikin (SAM)**, a survey that measures your emotional impression about the interaction using pictures, and (2) **Working Alliance Inventory (WAI)**, a survey that measures how good you think the interaction was.
  g. Finally, you will be asked to fill the **Robotic Social Attributes Scale (RoSAS)** once more. You will also be asked about your nationality, age, and gender.
  h. Once the experiment is over, we will talk about the experience.
  i. Feel free to ask, at any time, if you would like to take a rest, or stop the experiment.

② **Health considerations**
  a. As a measure to prevent the spread of infectious diseases, we will measure your body temperature before starting the experiment.
  b. The experiment will be conducted in a well-ventilated room and a distance of 2m will be kept between you and the instructor at all times.
  c. All devices and furniture used in the experiment will be disinfected and both you and the instructor will wear a mask and face-shield at all times.

③ **Ethical considerations**
  a. You will not suffer any disadvantages from the results obtained in the experiment and whether you agree to participate in the experiment.

④ **Agreeing to participate is completely voluntary**
  a. You are free to decide whether you want to participate or not without suffering any negative consequences.

【資料 2.2】研究についての説明（英語版）

⑤ **Cancellation of the agreement will not cause any penalties**
   a. You can stop the experiment and cancel your participation at any time without any negative consequences.

⑥ **If you decide not to participate, there are no penalties**
   a. You can decide not to participate without any negative consequences.

⑦ **Your personal information is protected**
   a. The data collected is not linked with any identifying information about you.
   b. We do not collect any personal information that can identify you.

Explained by	Department	Comprehensive Human Sciences Kansei, Behavioral, and Brain Sciences	
	Name	ONCHI SUGUIMITZU Diego Eiji	㊞
	Contact	s1930389@s.tsukuba.ac.jp	
Researcher Responsible	Faculty	Art and Design	
	Name	LEE Seung Hee	㊞
	Contact	lee.seunghee.gn@u.tsukuba.ac.jp	

This research is carried out with the utmost care so that there is no disadvantage to the participants, with the approval of the Research Ethics Committee of Art and Design, University of Tsukuba. If you have any comments or questions concerning your participation in the research, feel free to ask the researcher. Alternatively, please consult with the Research Ethics Committee of Art and Design.

【 Tel: 029-853-2571 (Research Support Office of Sports and Art Area) e-mail: tg-kenkyurinri@un.tsukuba.ac.jp】

B.4 Agreement Form: Japanese Version

【資料 3.1】同意書（日本語版）

同 意 書

筑波大学芸術系長　殿

私は、「人間とロボットのインタラクションにおける非言語フィードバックの評価」の研究について、その目的、方法、その成果及び危険性とその対処法について充分な説明を受けました。また、本研究への協力に同意しなくても何ら不利益を受けないことも確認した上で、被験者になることに 同意します。

ただし、この同意は、あくまでも私自身の自由意思によるものであり、不利益を受けず、随時撤回できるものであることを確認します。

令和　　年　　月　　日

氏　名

（自筆署名又は記名押印）

「人間とロボットのインタラクションにおける非言語フィードバックの評価」の研究について、書面及び口頭により令和　　年　　月　　日に説明を行い、上記のとおり同意を得ました。

研究責任者	所属	芸術系	
	氏名	李昇姫	㊞
	連絡先	lee.seunghee.gn@u.tsukuba.ac.jp	
説明者	所属	人間総合科学研究科　感性認知脳科学専攻	
	氏名	ONCHI SUGUIMITZU Diego Eiji	㊞
	連絡先	s1930389@s.tsukuba.ac.jp	

B.5 Agreement Form: English Version

【資料 3.2】同意書（英語版）

Agreement Form

To Director of Art and Design, University of Tsukuba,

I agree that I have received an adequate explanation about the objective, methodology, results, risks, and coping methods about the study: EFFECTS OF NON-VERBAL FEEDBACK DURING HUMAN-ROBOT INTERACTION. Also, I agree to become a participant after understanding that I will not suffer any disadvantages if I decide not to cooperate in this research.

Nevertheless, this agreement is based upon my own free will, and I understand that it can be withdrawn at any time without suffering any disadvantages.

Date (YYYY/MM/DD):

________/____/____

NAME: ________________________________

(Signature or Seal)

I received the aforementioned agreement after giving a textual and verbal explanation about the study: EFFECTS OF NON-VERBAL FEEDBACK DURING HUMAN-ROBOT INTERACTION on _____/____/____.

Researcher Responsible	Faculty	Art and Design	
	Name	LEE Seung Hee	㊞
	Contact	lee.seunghee.gn@u.tsukuba.ac.jp	
Explained by	Department	Comprehensive Human Sciences Kansei, Behavioral, and Brain Sciences	
	Name	ONCHI SUGUIMITZU Diego Eiji	㊞
	Contact	s1930389@s.tsukuba.ac.jp	

Appendix C

Electrodermal Analysis per Participant

C.1 EDA of Participant 1

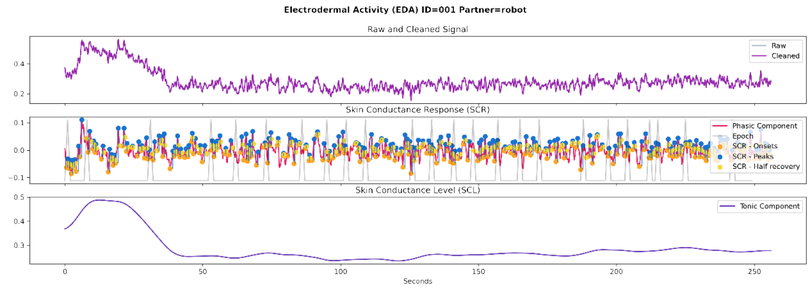


FIGURE C.1: Electrodermal analysis of participant 001 when interacting with the robot.

Source: own.

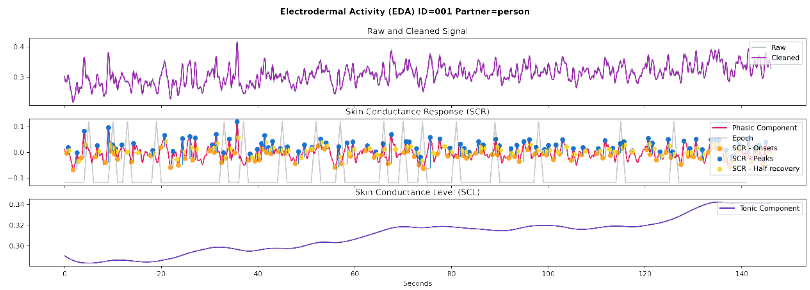


FIGURE C.2: Electrodermal analysis of participant 001 when interacting with the person.

Source: own.

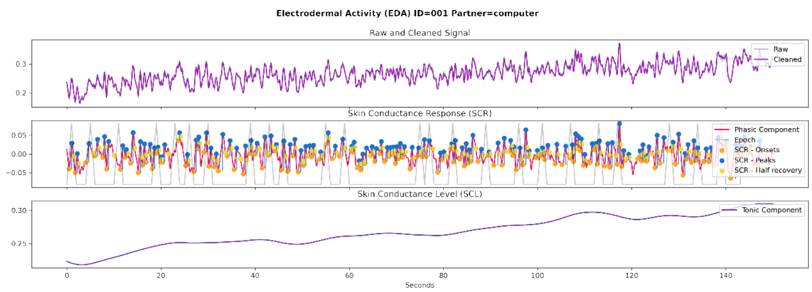


FIGURE C.3: Electrodermal analysis of participant 001 when interacting with the computer.

Source: own.

C.2 EDA of Participant 2

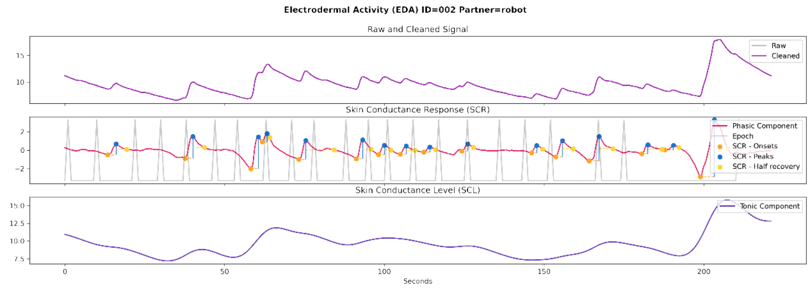


FIGURE C.4: Electrodermal analysis of participant 002 when interacting with the robot.

Source: own.

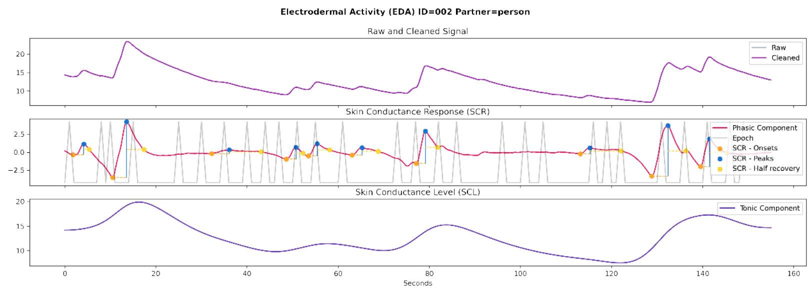


FIGURE C.5: Electrodermal analysis of participant 002 when interacting with the person.

Source: own.

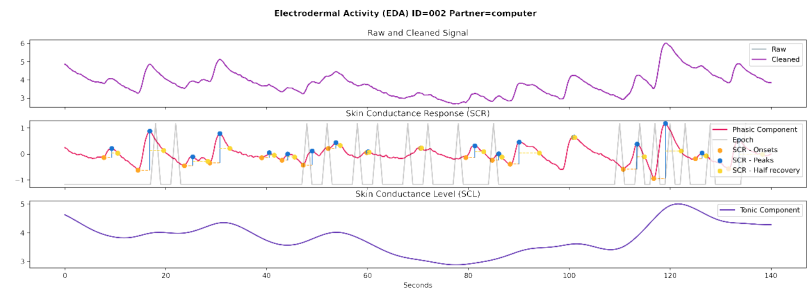


FIGURE C.6: Electrodermal analysis of participant 002 when interacting with the computer.

Source: own.

C.3 EDA of Participant 3

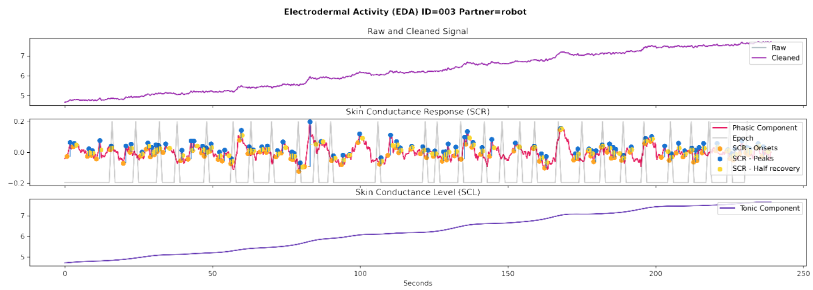


FIGURE C.7: Electrodermal analysis of participant 003 when interacting with the robot.

Source: own.

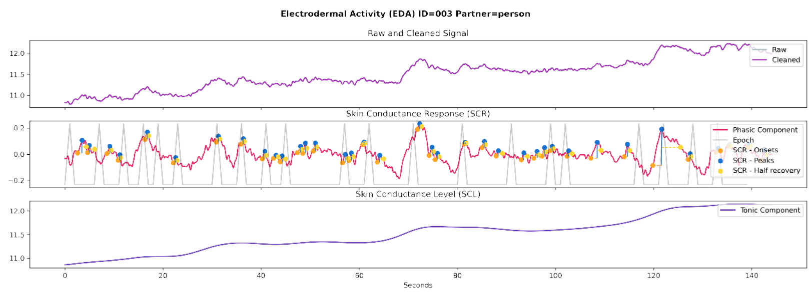


FIGURE C.8: Electrodermal analysis of participant 003 when interacting with the person.

Source: own.

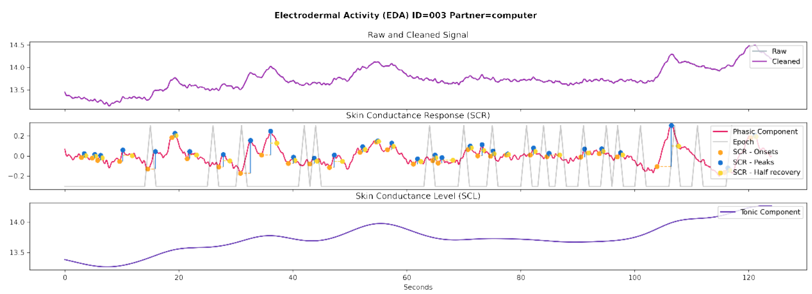


FIGURE C.9: Electrodermal analysis of participant 003 when interacting with the computer.

Source: own.

C.4 EDA of Participant 4

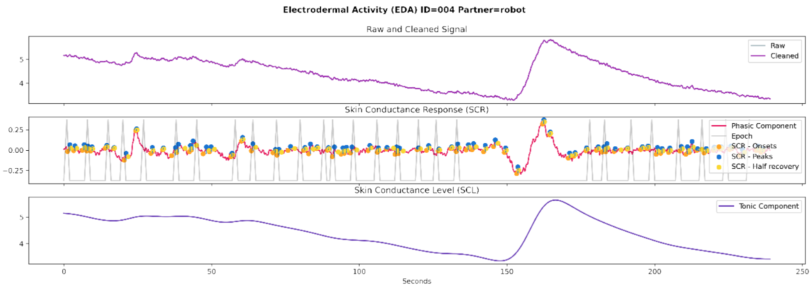


FIGURE C.10: Electrodermal analysis of participant 004 when interacting with the robot.

Source: own.

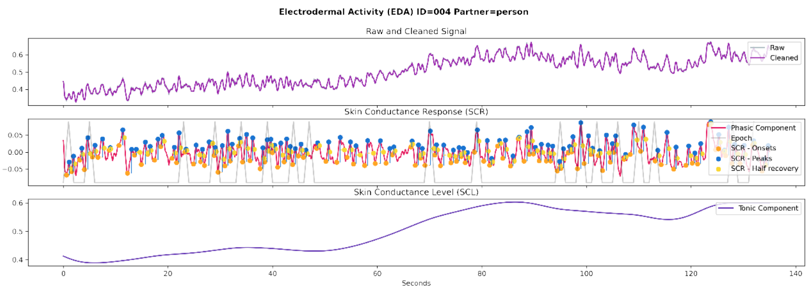


FIGURE C.11: Electrodermal analysis of participant 004 when interacting with the person.

Source: own.

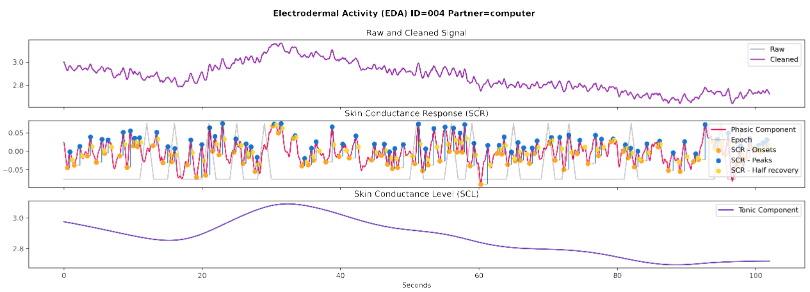


FIGURE C.12: Electrodermal analysis of participant 004 when interacting with the computer.

Source: own.

C.5 EDA of Participant 5

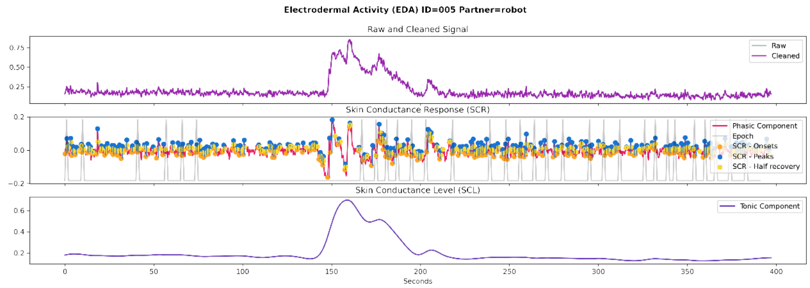


FIGURE C.13: Electrodermal analysis of participant 005 when interacting with the robot.

Source: own.

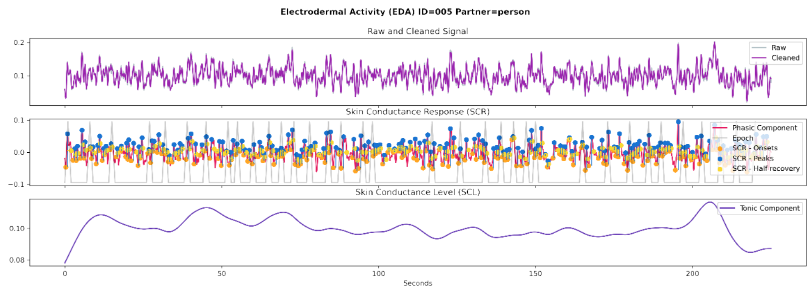


FIGURE C.14: Electrodermal analysis of participant 005 when interacting with the person.

Source: own.

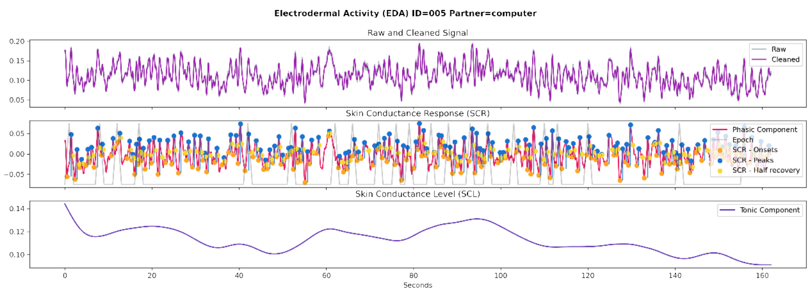


FIGURE C.15: Electrodermal analysis of participant 005 when interacting with the computer.

Source: own.

C.6 EDA of Participant 6

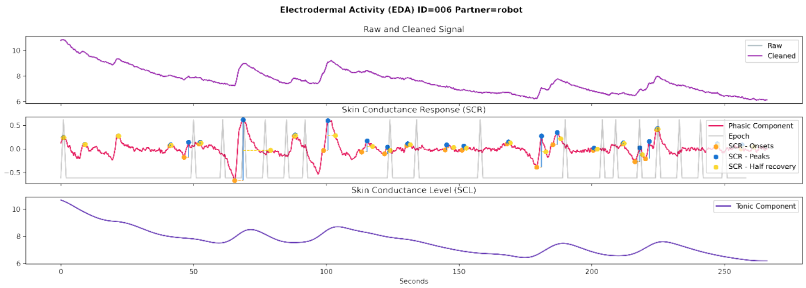


FIGURE C.16: Electrodermal analysis of participant 006 when interacting with the robot.

Source: own.

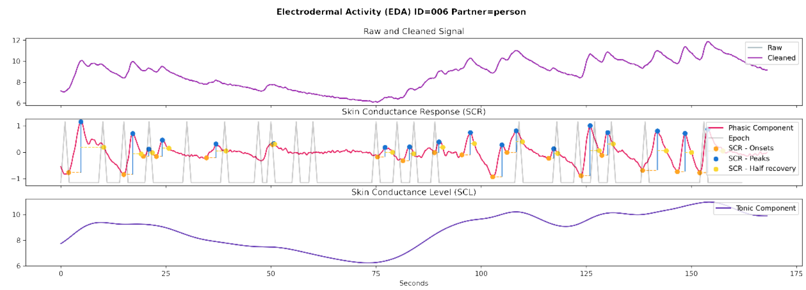


FIGURE C.17: Electrodermal analysis of participant 006 when interacting with the person.

Source: own.

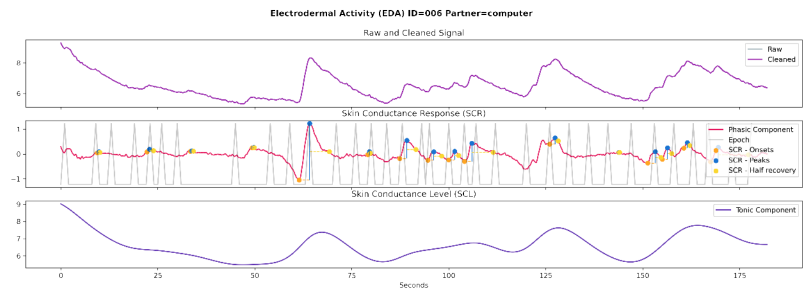


FIGURE C.18: Electrodermal analysis of participant 006 when interacting with the computer.

Source: own.

C.7 EDA of Participant 7

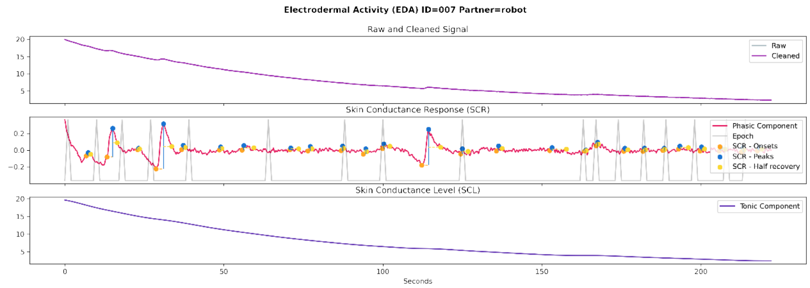


FIGURE C.19: Electrodermal analysis of participant 007 when interacting with the robot.

Source: own.

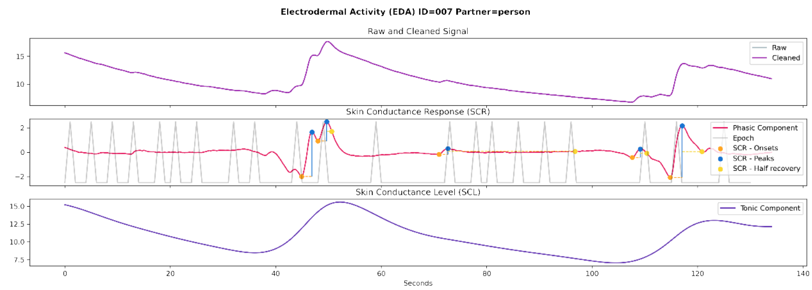


FIGURE C.20: Electrodermal analysis of participant 007 when interacting with the person.

Source: own.

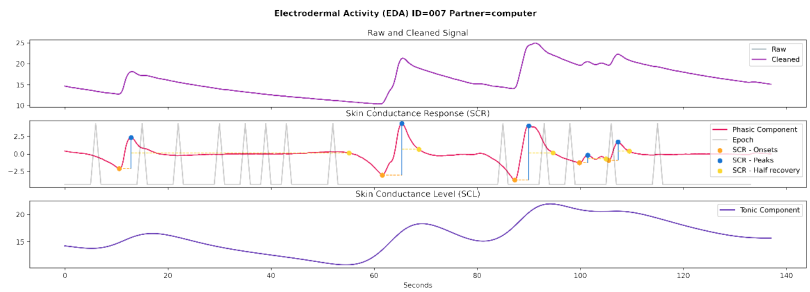


FIGURE C.21: Electrodermal analysis of participant 007 when interacting with the computer.

Source: own.

C.8 EDA of Participant 8

Electrodermal analysis of this participant when interacting with the computer was not available due to too much noise when recording.

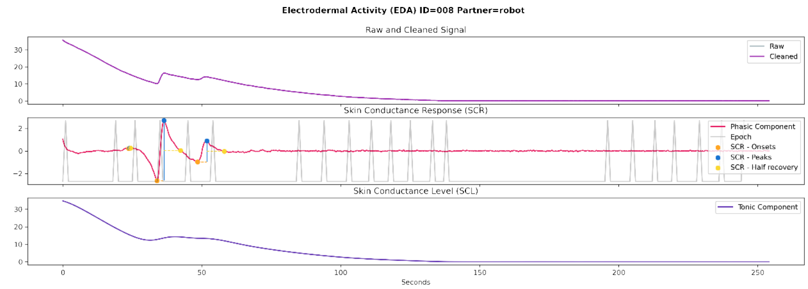


FIGURE C.22: Electrodermal analysis of participant 008 when interacting with the robot.

Source: own.

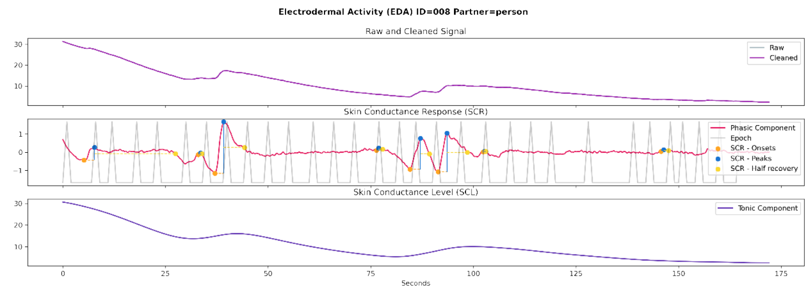


FIGURE C.23: Electrodermal analysis of participant 008 when interacting with the person.

Source: own.

C.9 EDA of Participant 9

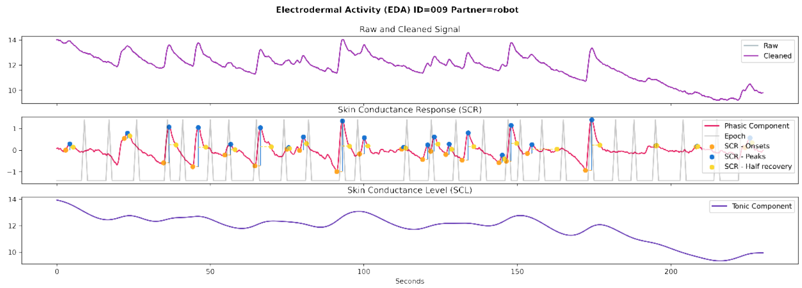


FIGURE C.24: Electrodermal analysis of participant 009 when interacting with the robot.

Source: own.

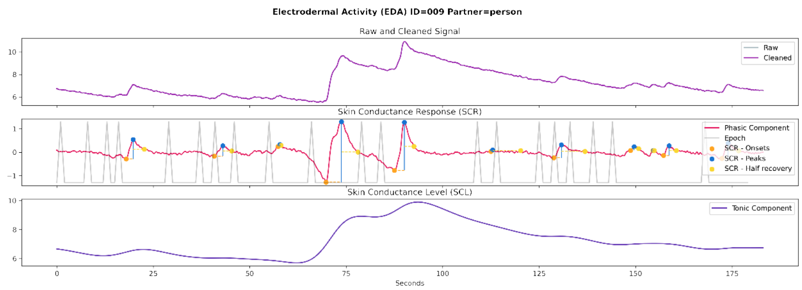


FIGURE C.25: Electrodermal analysis of participant 009 when interacting with the person.

Source: own.

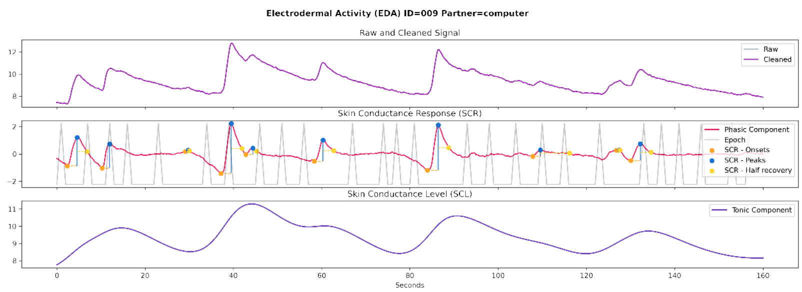


FIGURE C.26: Electrodermal analysis of participant 009 when interacting with the computer.

Source: own.

C.10 EDA of Participant 10

Electrodermal analysis of this participant when interacting with the robot was not available due to too much noise when recording. Electrodermal analysis of this participant when interacting with the computer was not available due to too much noise when recording.

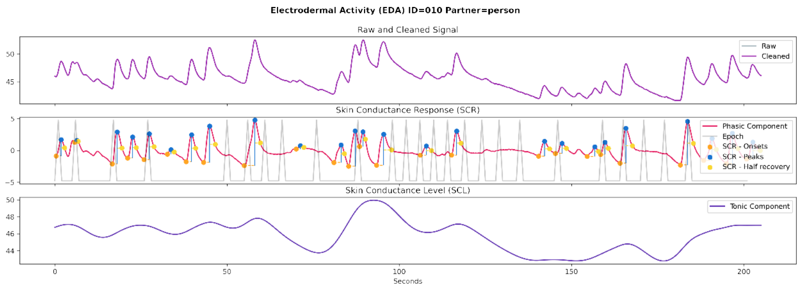


FIGURE C.27: Electrodermal analysis of participant 010 when interacting with the person.

Source: own.

C.11 EDA of Participant 11

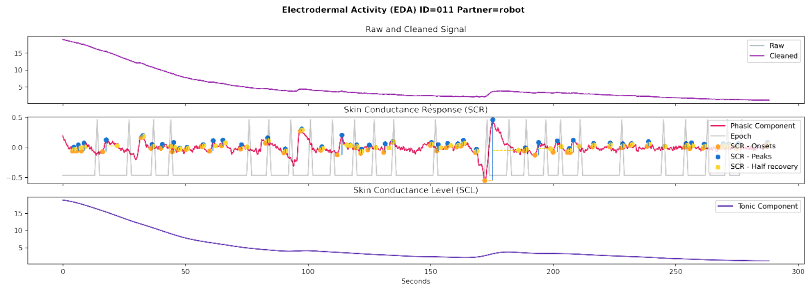


FIGURE C.28: Electrodermal analysis of participant 011 when interacting with the robot.

Source: own.

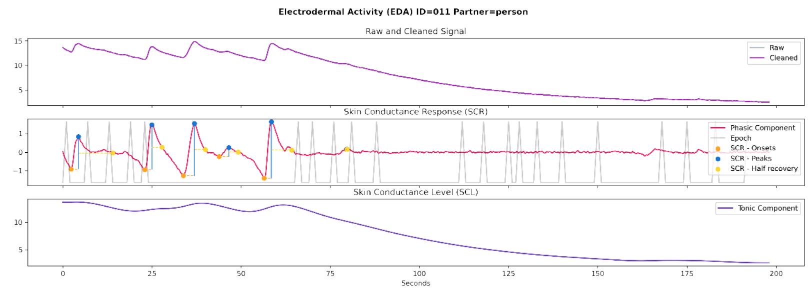


FIGURE C.29: Electrodermal analysis of participant 011 when interacting with the person.

Source: own.

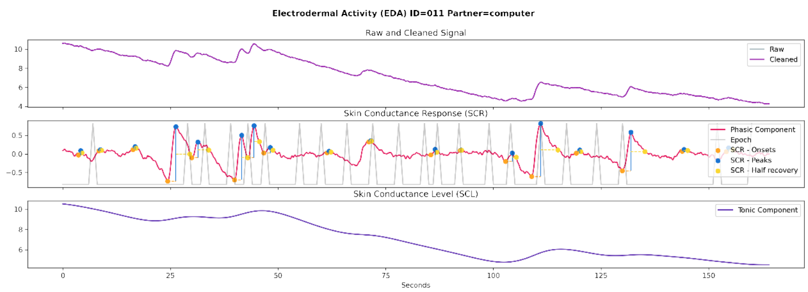


FIGURE C.30: Electrodermal analysis of participant 011 when interacting with the computer.

Source: own.

C.12 EDA of Participant 12

Electrodermal analysis of this participant when interacting with the computer was not available due to too much noise when recording.

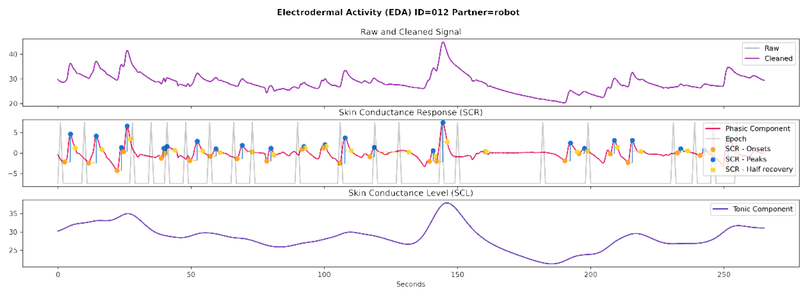


FIGURE C.31: Electrodermal analysis of participant 012 when interacting with the robot.

Source: own.

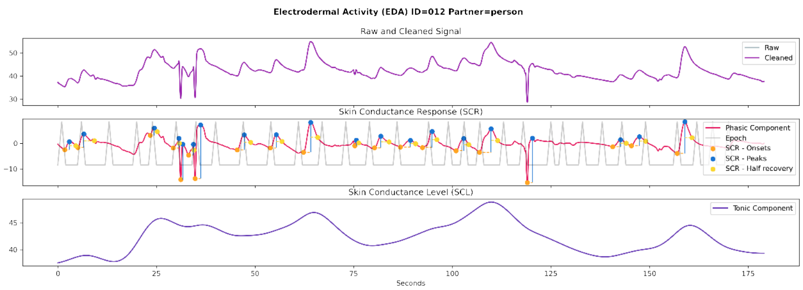


FIGURE C.32: Electrodermal analysis of participant 012 when interacting with the person.

Source: own.

C.13 EDA of Participant 13

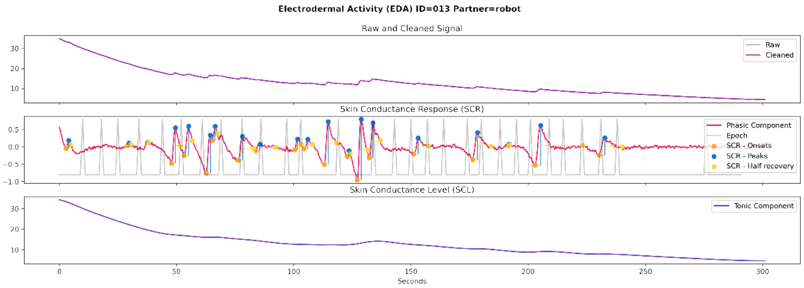


FIGURE C.33: Electrodermal analysis of participant 013 when interacting with the robot.

Source: own.

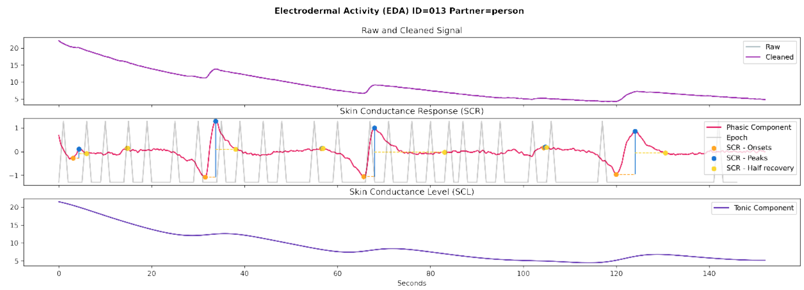


FIGURE C.34: Electrodermal analysis of participant 013 when interacting with the person.

Source: own.

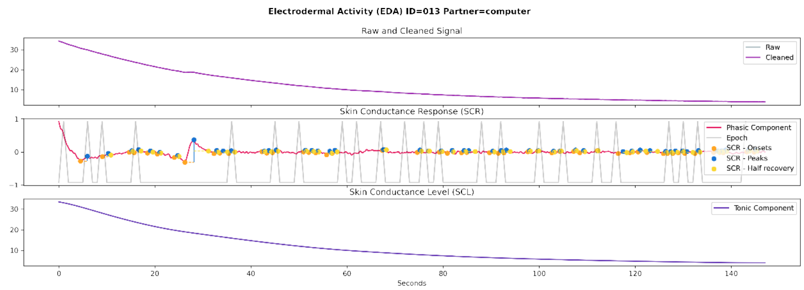


FIGURE C.35: Electrodermal analysis of participant 013 when interacting with the computer.

Source: own.

C.14 EDA of Participant 14

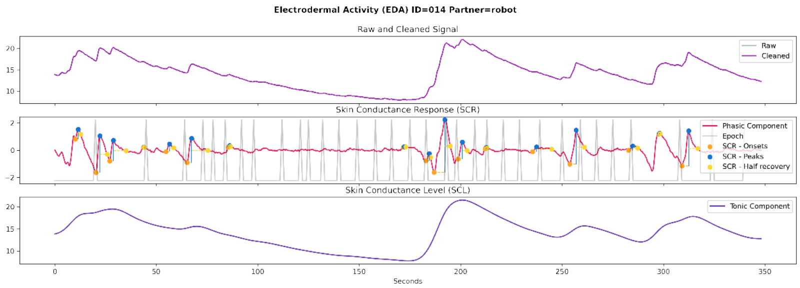


FIGURE C.36: Electrodermal analysis of participant 014 when interacting with the robot.

Source: own.

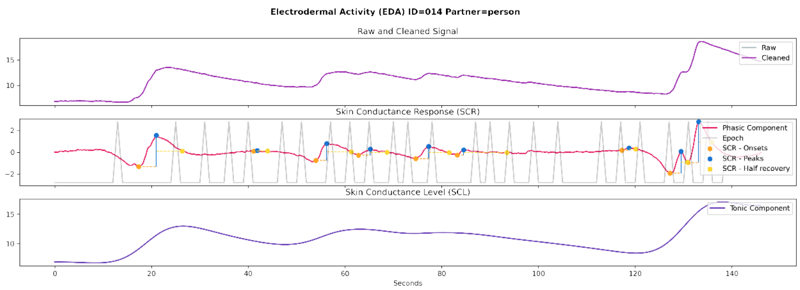


FIGURE C.37: Electrodermal analysis of participant 014 when interacting with the person.

Source: own.

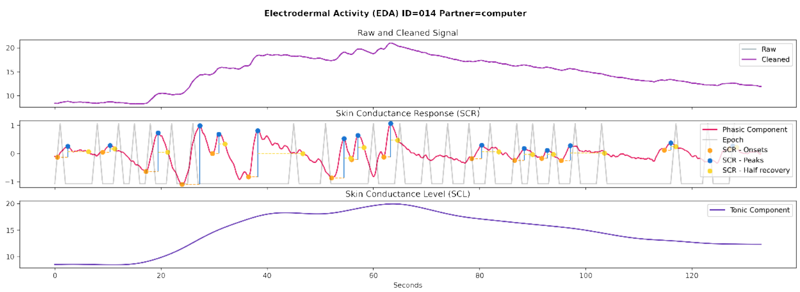


FIGURE C.38: Electrodermal analysis of participant 014 when interacting with the computer.

Source: own.

C.15 EDA of Participant 15

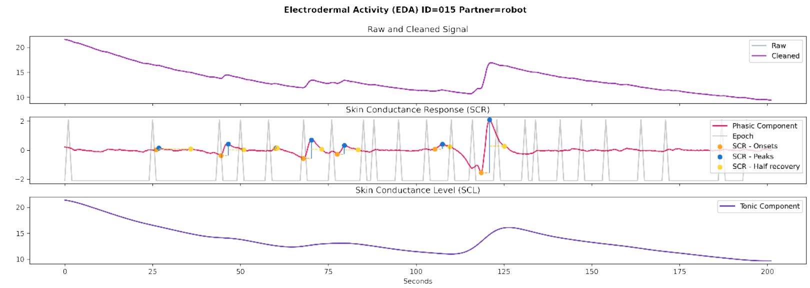


FIGURE C.39: Electrodermal analysis of participant 015 when interacting with the robot.

Source: own.

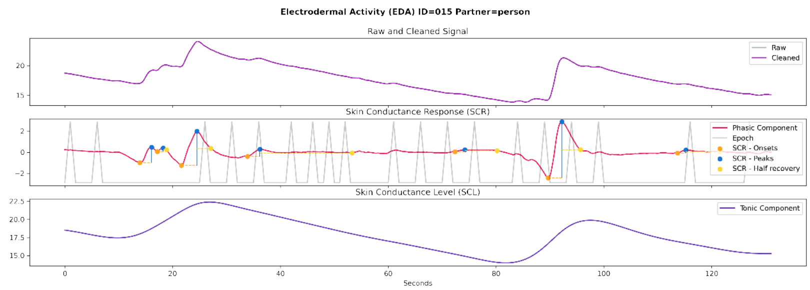


FIGURE C.40: Electrodermal analysis of participant 015 when interacting with the person.

Source: own.

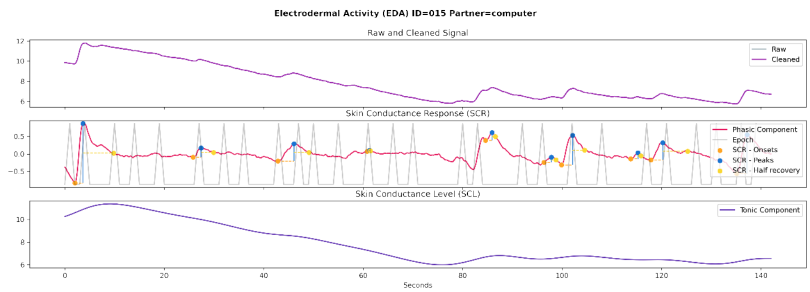


FIGURE C.41: Electrodermal analysis of participant 015 when interacting with the computer.

Source: own.

C.16 EDA of Participant 16

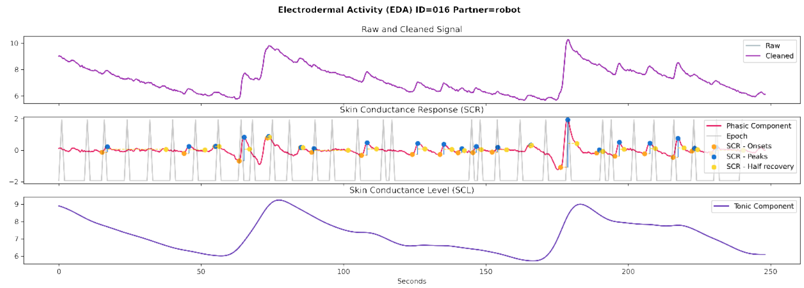


FIGURE C.42: Electrodermal analysis of participant 016 when interacting with the robot.

Source: own.

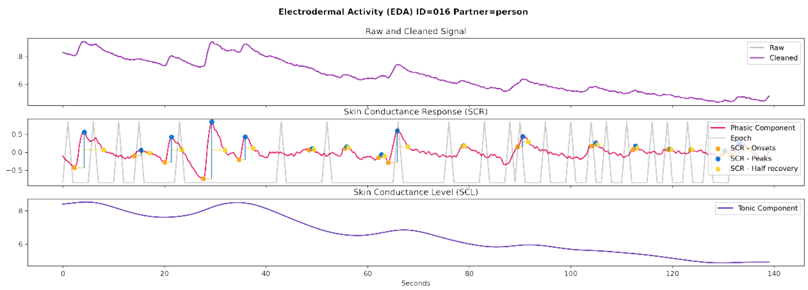


FIGURE C.43: Electrodermal analysis of participant 016 when interacting with the person.

Source: own.

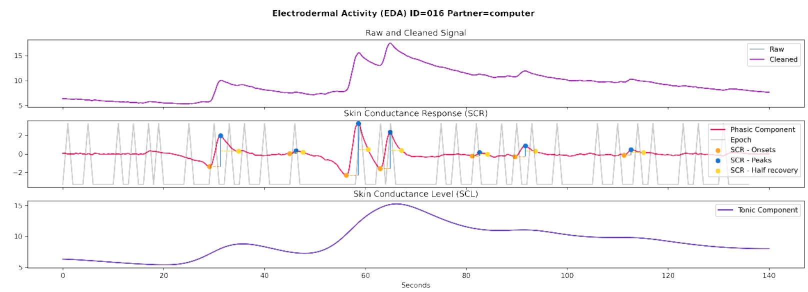


FIGURE C.44: Electrodermal analysis of participant 016 when interacting with the computer.

Source: own.

C.17 EDA of Participant 17

Electrodermal analysis of this participant when interacting with the person was not available due to too much noise when recording.

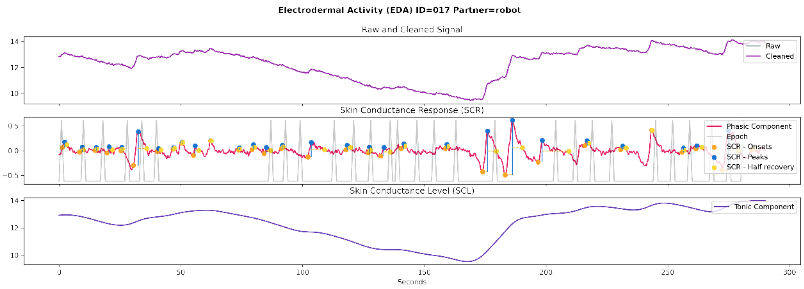


FIGURE C.45: Electrodermal analysis of participant 017 when interacting with the robot.

Source: own.

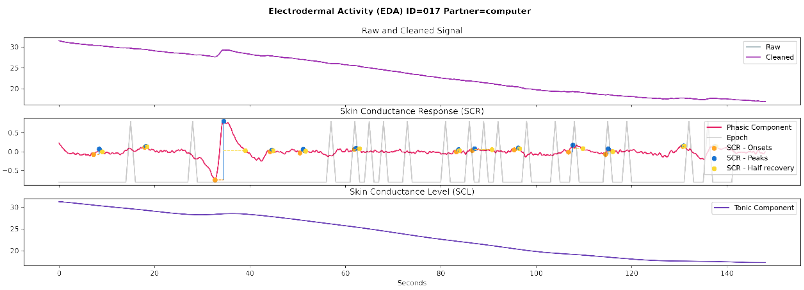


FIGURE C.46: Electrodermal analysis of participant 017 when interacting with the computer.

Source: own.

C.18 EDA of Participant 18

Electrodermal analysis of this participant when interacting with the person was not available due to too much noise when recording. Electrodermal analysis of this participant when interacting with the computer was not available due to too much noise when recording.

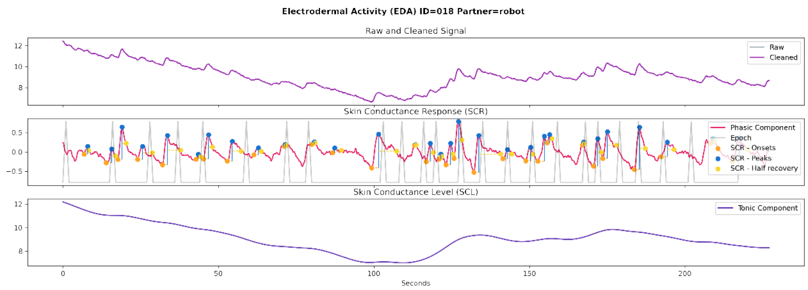


FIGURE C.47: Electrodermal analysis of participant 018 when interacting with the robot.

Source: own.

C.19 EDA of Participant 19

Electrodermal analysis of this participant when interacting with the robot was not available due to too much noise when recording.

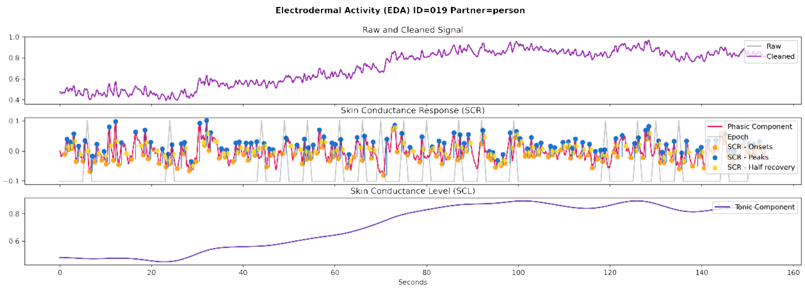


FIGURE C.48: Electrodermal analysis of participant 019 when interacting with the person.

Source: own.

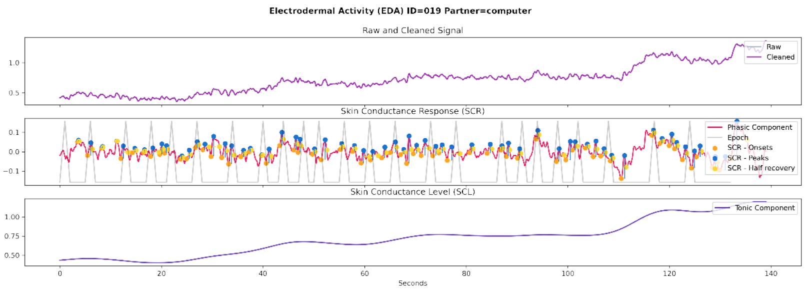


FIGURE C.49: Electrodermal analysis of participant 019 when interacting with the computer.

Source: own.

C.20 EDA of Participant 20

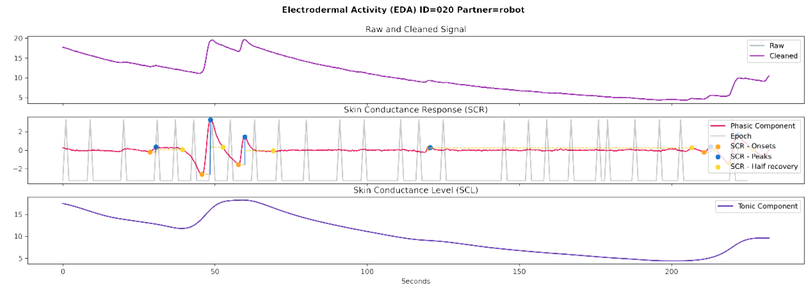


FIGURE C.50: Electrodermal analysis of participant 020 when interacting with the robot.

Source: own.

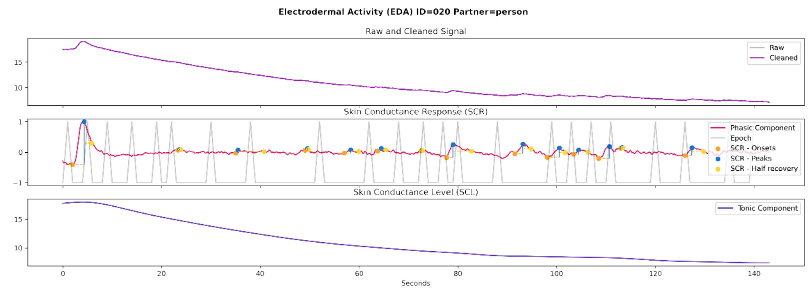


FIGURE C.51: Electrodermal analysis of participant 020 when interacting with the person.

Source: own.

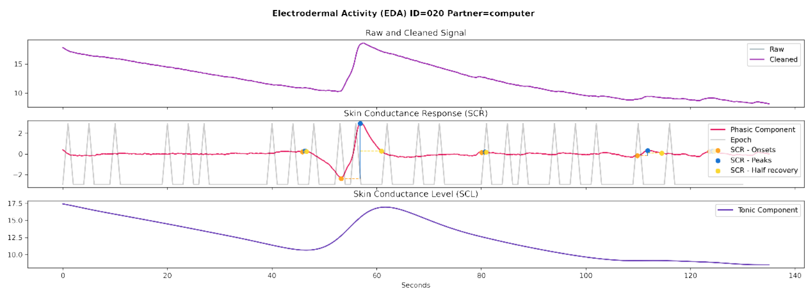


FIGURE C.52: Electrodermal analysis of participant 020 when interacting with the computer.

Source: own.

C.21 EDA of Participant 21

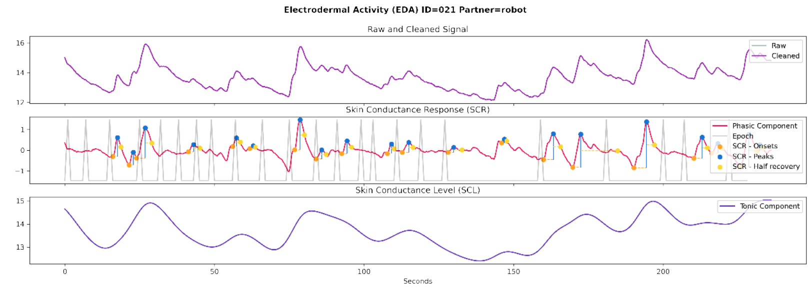


FIGURE C.53: Electrodermal analysis of participant 021 when interacting with the robot.

Source: own.

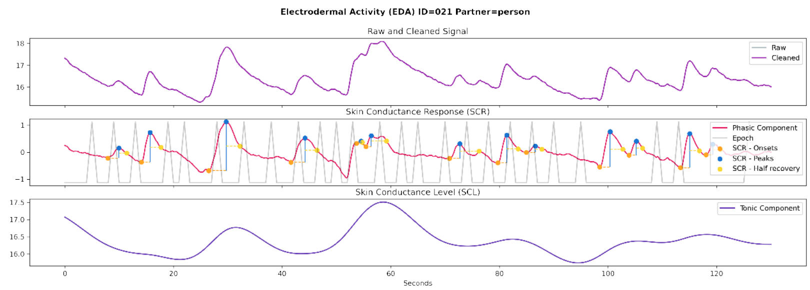


FIGURE C.54: Electrodermal analysis of participant 021 when interacting with the person.

Source: own.

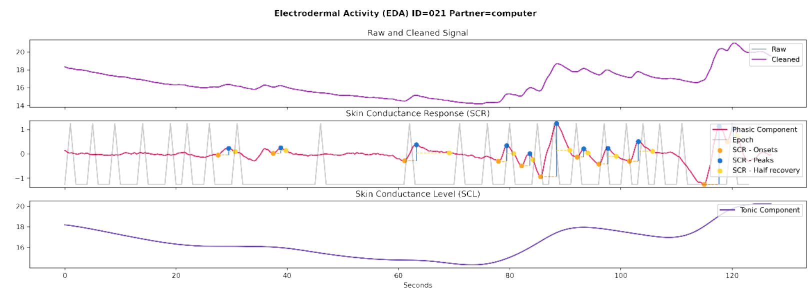


FIGURE C.55: Electrodermal analysis of participant 021 when interacting with the computer.

Source: own.

C.22 EDA of Participant 22

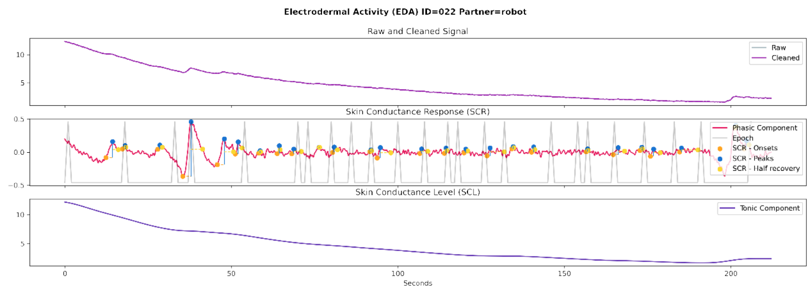


FIGURE C.56: Electrodermal analysis of participant 022 when interacting with the robot.

Source: own.

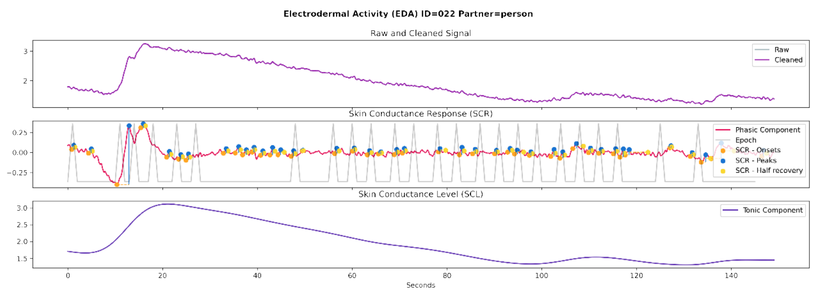


FIGURE C.57: Electrodermal analysis of participant 022 when interacting with the person.

Source: own.

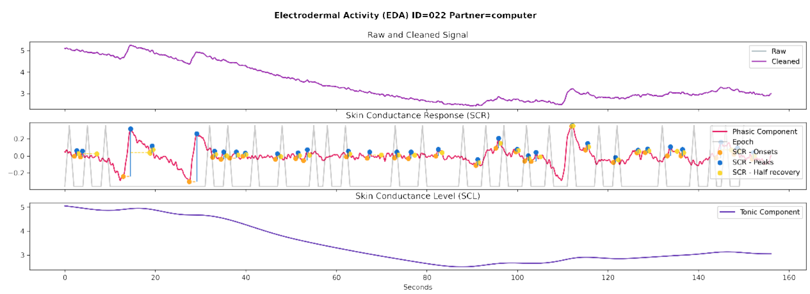


FIGURE C.58: Electrodermal analysis of participant 022 when interacting with the computer.

Source: own.

C.23 EDA of Participant 23

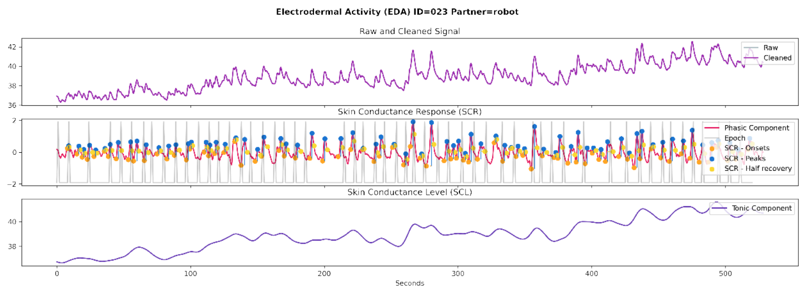


FIGURE C.59: Electrodermal analysis of participant 023 when interacting with the robot.

Source: own.

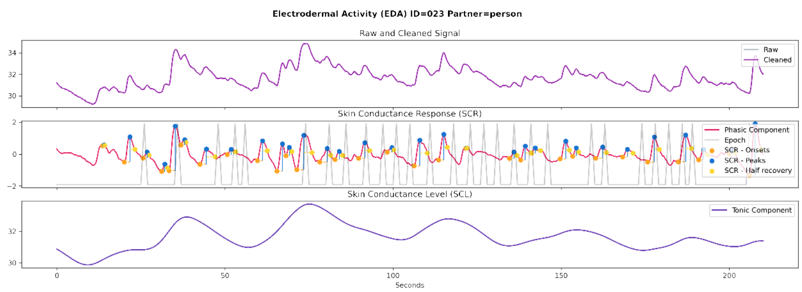


FIGURE C.60: Electrodermal analysis of participant 023 when interacting with the person.

Source: own.

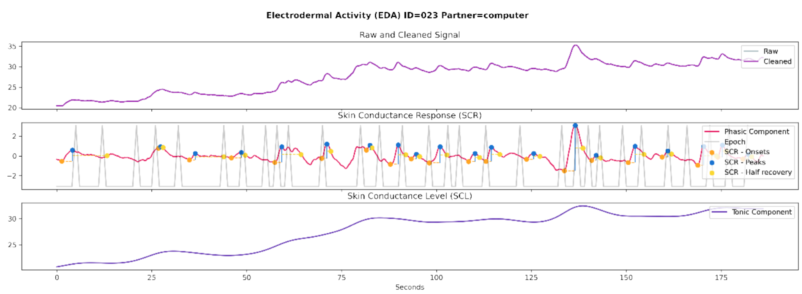


FIGURE C.61: Electrodermal analysis of participant 023 when interacting with the computer.

Source: own.

C.24 EDA of Participant 24

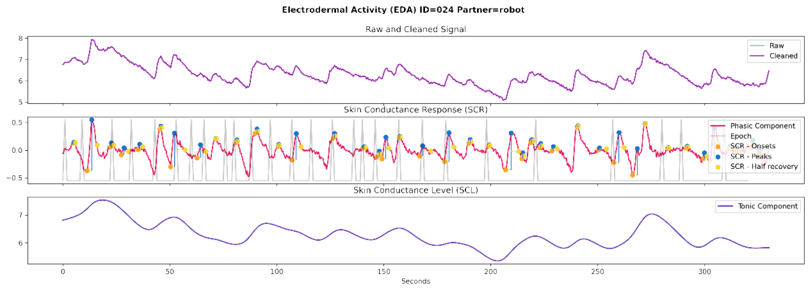


FIGURE C.62: Electrodermal analysis of participant 024 when interacting with the robot.

Source: own.

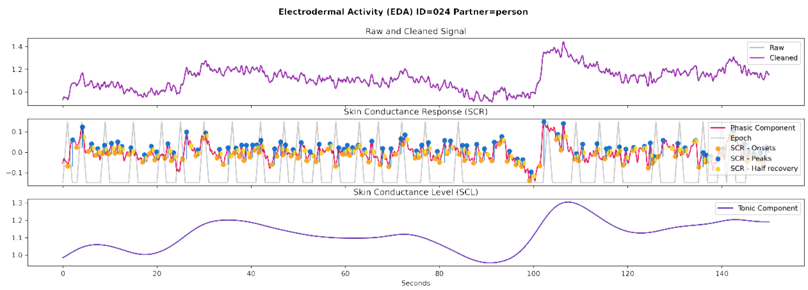


FIGURE C.63: Electrodermal analysis of participant 024 when interacting with the person.

Source: own.

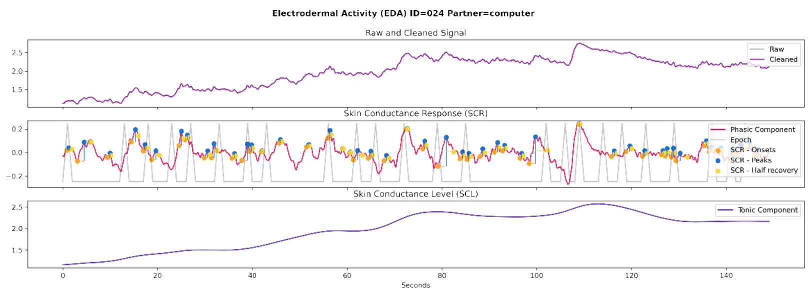


FIGURE C.64: Electrodermal analysis of participant 024 when interacting with the computer.

Source: own.

Appendix D

Source Code

D.1 Module: robot

D.1.1 License

D.1.2 *robot/robot.go*

```go
package main

import (
	"bufio"
	"context"
	"encoding/json"
	"fmt"
	"image"
	"image/color"
	"image/jpeg"
	"log"
	"math"
	"math/rand"
	"os"
	"sync"
	"time"

	"net/http"

	"module/anim"
	"module/body"
	"module/eye"
	"module/tracker.git"
)

func main() {
	e, err := eye.New()
	if err != nil {
		log.Fatal(err)
	}
	defer e.Close()

	b, err := body.New(14, 15)
	if err != nil {
		log.Fatal(err)
	}
	defer b.Close()

	cIdle := color.CMYK{255, 0, 0, 150}
	cWake := color.CMYK{255, 0, 0, 10}
	cStand := color.CMYK{255, 0, 0, 65}
	e.Color(cIdle)
	e.Offset(b.Angle() - 0.72)

	anim.Eye = e
	anim.Body = b
	list, err := anim.ReadFile("./anim.yaml")
	if err != nil {
		log.Fatal(err)
	}
	fmt.Println("loaded animations:")
	fmt.Printf("list = %+v\n", list)
```

```

	w, err := anim.Play("think")
	if err != nil {
		log.Fatal(err)
	}
	w.Wait()

	b.SetSpeed(0.2)

	var ws sync.WaitGroup
	done := make(chan struct{})
	ws.Add(1)
	go func() {
		defer ws.Done()
		t := time.NewTicker(20 * time.Millisecond)
		for {
			select {
			case <-done:
				return
			case <-t.C:
			}
			e.Offset(b.Angle() - 0.72)
		}
	}()

	tracker, err := tracker.New(640, 480)
	if err != nil {
		panic(err)
	}
	defer tracker.Close()

	fmt.Println("Press [ENTER] to close")

	reqImg := make(chan struct{})
	resImg := make(chan image.Image)

	pause := make(chan struct{})
	pauseOut := make(chan struct{})

	colorIdle := cIdle
	faceFound := make(chan struct{})
	faceGo := make(chan struct{})
	ws.Add(1)
	go func() {
		defer ws.Done()
		for {
			select {
			case <-done:
				return
			case <-pause:
				<-pauseOut
				continue
			default:
				start := time.Now()
				angle := math.Pi - b.Angle()
				x, y, found := tracker.Detect(angle)
				select {
				case <-reqImg:
					img := tracker.Image()
					resImg <- img
				default:
				}

				if found {
					x /= 6
					y /= 6
					y += 0.01

					select {
					case <-done:
						return
					case <-pause:
						<-pauseOut
						continue
					case faceFound <- struct{}{}:
					}

					select {
					case <-done:
						return
					case <-pause:
						<-pauseOut
						continue
					case <-faceGo:
					}
					b.Move(x, y, body.MoveRelative)
				} else {
					e.Color(colorIdle)
				}

				select {
				case <-done:
					return
```

```
				case <-pause:
					<-pauseOut
					continue
				default:
				}

				clear()
				fmt.Printf("face (%v): %.2f, %.2f (angle: %.2f)", found, x, y, b.Angle()/math.Pi*180)

				elapsed := time.Now().Sub(start)
				fps := 1.0 / elapsed.Seconds()
				fmt.Printf("(%v, %.0f fps)", elapsed, fps)
			}

		}
	}()

	ws.Add(1)
	go func() {
		defer ws.Done()
		for {
			select {
			case <-done:
				return
			case <-pause:
				<-pauseOut
			case <-time.After(10 * time.Second):
				clear()
				fmt.Printf("recentering")
				b.Move(0, 0, body.MoveAbsolute)
				colorIdle = cIdle
			case <-faceFound:
				colorIdle = cStand
				e.Color(cWake)
				select {
				case <-done:
					return
				case faceGo <- struct{}{}:
				}
			}
		}
	}()

	ws.Add(1)
	go func() {
		defer ws.Done()
		for {
			t := time.NewTicker(time.Duration(rand.Intn(7500)+2857) * time.Millisecond)
			select {
			case <-done:
				return
			case <-pause:
				<-pauseOut
			case <-t.C:
			}
			t.Stop()
			e.Blink(1.0)
		}
	}()

	openRequests := func(handler func(w http.ResponseWriter, r *http.Request)) func(w http.ResponseWriter, r *http.Request) {
		return func(w http.ResponseWriter, r *http.Request) {
			w.Header().Set("Access-Control-Allow-Origin", "*")
			w.Header().Set("Access-Control-Allow-Credentials", "true")
			w.Header().Set("Access-Control-Allow-Headers", "Authorization, Content-Type")
			w.Header().Set("Access-Control-Allow-Method", "GET, POST")

			handler(w, r)
		}
	}

	http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
		reqImg <- struct{}{}
		img := <-resImg
		if err := jpeg.Encode(w, img, &jpeg.Options{Quality: 30}); err != nil {
			panic(err)
		}
	})

	lightOn := true
	lightHandler := func(w http.ResponseWriter, r *http.Request) {
		lightOn = !lightOn
		e.TurnOn(lightOn)

		w.WriteHeader(http.StatusOK)
	}
	http.HandleFunc("/light", openRequests(lightHandler))

	thinkHandler := func(w http.ResponseWriter, r *http.Request) {
		for i := 0; i < 3; i++ {
			pause <- struct{}{}
		}
```

```
			n := rand.Intn(len(list))
			fmt.Printf("playing = %+v\n", list[n])
			wait, err := anim.Play(list[n])
			if err != nil {
				log.Fatal(err)
			}
			wait.Wait()
			for i := 0; i < 6; i++ {
				select {
				case pauseOut <- struct{}{}:
					continue
				default:
				}
			}
			time.Sleep(1 * time.Second)

			w.WriteHeader(http.StatusOK)
		}
		http.HandleFunc("/think", openRequests(thinkHandler))

		posHandler := func(w http.ResponseWriter, r *http.Request) {
			x, y := b.Position()
			b, err := json.Marshal(struct {
				X float64 `json:"x"`
				Y float64 `json:"y"`
			}{
				X: x,
				Y: y,
			})
			if err != nil {
				panic(err)
			}

			w.Write(b)
		}
		http.HandleFunc("/position", openRequests(posHandler))

		srv := http.Server{
			Addr: ":8080",
		}
		ws.Add(1)
		go func() {
			defer ws.Done()
			fmt.Println("Listening on port: 8080")
			if err := srv.ListenAndServe(); err != http.ErrServerClosed {
				panic(err)
			}
		}()

		bufio.NewReader(os.Stdin).ReadString('\n')
		fmt.Println("closing")
		close(done)
		if err := srv.Shutdown(context.Background()); err != nil {
			panic(err)
		}
		ws.Wait()
		e.Color(color.NRGBA{255, 0, 0, 255})
		time.Sleep(500 * time.Millisecond)
}

func clear() {
		fmt.Printf("\r                                                                                              \r")
}
```

D.2 Module: servo

D.2.1 License

D.2.2 *servo/servo.go*

```go
package servo

import (
	"fmt"
	"strings"
	"sync"
	"time"
)

type flag uint8

// is check if the given bits are set in the flag.
func (f flag) is(bits flag) bool {
	return f&bits != 0
}

// String implements the Stringer interface.
func (f flag) String() string {
	if f == 0 {
		return "( NONE )"
	}

	s := new(strings.Builder)

	fmt.Fprintf(s, "(")

	if f.is(Centered) {
		fmt.Fprintf(s, " Centered")
	}
	if f.is(Normalized) {
		fmt.Fprintf(s, " Normalized")
	}

	fmt.Fprintf(s, " )")

	return s.String()
}

const (
	// Centered sets the range of the servo from -90 to 90 degrees.
	// Together with Normalized, the range of the servo is set to -1 to 1.
	Centered flag = (1 << iota)
	// Normalized sets the range of the servo from 0 to 2.
	// Together with Centered, the range of the servo is set to -1 to 1.
	Normalized
)

// Servo is a struct that holds all the information necessary to control a
// servo motor. Use the function servo.New(gpio) for correct
// initialization. Servo is designed to be concurrent-safe.
type Servo struct {
	// pin is the GPIO pin number of the Raspberry Pi. Check that the pin is
```

```
	// controllable with pi-blaster.
	//
	// CAUTION: Incorrect pin assignment might cause damage to your Raspberry
	// Pi.
	pin gpio
	// Name is an optional value to assign a meaningful name to the servo.
	Name string
	// Flags is a bit flag that sets various configuration parameters.
	//
	// servo.Centered sets the range of the servo from -90 to 90 degrees.
	//
	// servo.Normalized sets the range of the servo from 0 to 2.
	// Together with servo.Centered, the range of the servo is set to -1 to 1.
	Flags flag

	// MinPulse is the minimum pwm pulse of the servo. (default 0.05 s)
	// MaxPulse is the maximum pwm pulse of the servo. (default 0.25 s)
	// These calibration variables should be immutables once the servo is
	// connected..
	MinPulse, MaxPulse float64

	target, position float64
	deltaT           time.Time
	lastPWM          pwm

	step, maxStep float64

	idle     bool
	finished *sync.Cond
	lock     *sync.RWMutex
}

// updateRate is set to 3ms/degree, an approximate on 0.19s/60degrees.

// String implements the Stringer interface.
// It returns a string in the following format:
//
// servo "NAME" connected to gpio(GPIO_PIN) [flags: ( FLAGS_SET )]
//
// where NAME is the verbose name (default: fmt.Sprintf("Servo%d", GPIO)),
// GPIO_PIN is the connection pin of the servo, and FLAGS_SET is the list of
// flags set (default: NONE).
func (s *Servo) String() string {
	return fmt.Sprintf("servo %q connected to gpio(%d) [flags: %v]", s.Name, s.pin, s.Flags)
}

// New creates a new Servo struct with default values, connected at a GPIO pin
// of the Raspberry Pi. You should check that the pin is controllable with pi-blaster.
//
// CAUTION: Incorrect pin assignment might cause damage to your Raspberry
// Pi.
func New(GPIO int) (s *Servo) {
	// maxS is the maximun degrees/s for a tipical servo of speed
	// 0.19s/60degrees.
	const maxS = 315.7

	s = &Servo{
		pin:      gpio(GPIO),
		Name:     fmt.Sprintf("Servo%d", GPIO),
		maxStep:  maxS,
		step:     maxS,
		MinPulse: 0.05,
		MaxPulse: 0.25,

		idle:     true,
		finished: sync.NewCond(&sync.Mutex{}),
		lock:     new(sync.RWMutex),
	}

	return s
}

// Connect connects the servo to the pi-blaster daemon.
func (s *Servo) Connect() error {
	_blaster.subscribe(s)

	return nil
}

// Close cleans up the state of the servo and deactivates the corresponding
// GPIO pin.
func (s *Servo) Close() {
	_blaster.unsubscribe(s)
}

// Position returns the current angle of the servo, adjusted for its Flags.
func (s *Servo) Position() float64 {
	s.lock.RLock()
	defer s.lock.RUnlock()

	p := s.position
	if s.Flags.is(Centered) {
		p -= 90
```

```go
	}
	if s.Flags.is(Normalized) {
		p /= 90
	}

	return p
}

// Waiter implements the Wait function.
type Waiter interface {
	// Wait waits for the servo to finish moving.
	Wait()
}

// MoveTo sets a target angle for the servo to move. The magnitude of the target
// depends on the servo's Flags. The target is automatically clamped to the set
// range. If called concurrently, the target position is overridden by the last
// goroutine (usually non-deterministic).
func (s *Servo) MoveTo(target float64) (wait Waiter) {
	s.moveTo(target)
	return s
}

func (s *Servo) moveTo(target float64) {
	if s.Flags.is(Normalized) {
		target *= 90
	}
	if s.Flags.is(Centered) {
		target += 90
	}

	s.lock.Lock()
	defer s.lock.Unlock()

	if s.step == 0.0 {
		s.target = s.position
	} else {
		s.target = clamp(target, 0, 180)
	}
	s.deltaT = time.Now()
	s.idle = false
}

// SetSpeed changes the speed of the servo from (still) 0.0 to 1.0 (max speed).
// Setting a speed of 0.0 effectively sets the target position to the current
// position and the servo will not move.
func (s *Servo) SetSpeed(percentage float64) {
	s.lock.Lock()
	defer s.lock.Unlock()

	s.step = s.maxStep * clamp(percentage, 0.0, 1.0)
}

// Stop stops moving the servo. This effectively sets the target position to
// the stopped position of the servo.
func (s *Servo) Stop() {
	s.lock.Lock()
	defer s.lock.Unlock()

	s.target = s.position
	s.idle = true
	s.finished.L.Lock()
	s.finished.Broadcast()
	s.finished.L.Unlock()
}

// SetPosition immediately sets the angle the servo.
func (s *Servo) SetPosition(position float64) {
	if s.Flags.is(Normalized) {
		position *= 90
	}
	if s.Flags.is(Centered) {
		position += 90
	}

	s.lock.Lock()
	defer s.lock.Unlock()

	s.position = clamp(position, 0, 180)
	s.target = s.position
	s.idle = false
}

// pwm linearly interpolates an angle based on the start, finish, and
// duration of the movement, and returns the gpio pin and adjusted pwm for the
// current time.
func (s *Servo) pwm() (gpio, pwm) {
	ok := false
	s.lock.RLock()
	p := s.position
	_pwm := s.lastPWM

	defer func() {
```

```
		if !ok {
			s.lock.Lock()
			s.position = p
			s.lastPWM = _pwm
			s.deltaT = time.Now()

			if p == s.target {
				s.idle = true
				s.finished.L.Lock()
				s.finished.Broadcast()
				s.finished.L.Unlock()
			}
			s.lock.Unlock()
		}
	}()
	defer s.lock.RUnlock()

	if s.position == s.target && s.idle {
		ok = true
		return s.pin, _pwm
	}

	delta := time.Since(s.deltaT).Seconds() * s.step
	if s.target < s.position {
		p = s.position - delta
		if p <= s.target {
			p = s.target
		}
	} else {
		p = s.position + delta
		if p >= s.target {
			p = s.target
		}
	}

	_pwm = pwm(remap(p, 0, 180, s.MinPulse, s.MaxPulse))

	return s.pin, _pwm
}

// isIdle checks if the servo is not moving.
func (s *Servo) isIdle() bool {
	s.lock.RLock()
	defer s.lock.RUnlock()

	return s.idle
}

// Wait waits for the servo to stop moving. It is concurrent-safe.
func (s *Servo) Wait() {
	s.finished.L.Lock()
	defer s.finished.L.Unlock()

	for !s.isIdle() {
		s.finished.Wait()
	}
}

func clamp(value, min, max float64) float64 {
	if value < min {
		value = min
	}
	if value > max {
		value = max
	}
	return value
}

func remap(value, min, max, toMin, toMax float64) float64 {
	return (value-min)/(max-min)*(toMax-toMin) + toMin
}
```

D.2.3 `servo/blaster.go`

```
package servo

import (
	"fmt"
	"io/ioutil"
	"log"
	"math"
	"os"
	"os/exec"
	"strings"
	"sync"
	"time"
)
```

```go

type blaster struct {
	disabled bool
	buffer   chan string
	done     chan struct{}
	servos   chan servoPkg
	_servos  map[gpio]*Servo

	rate chan time.Duration

	ws *sync.WaitGroup
}

var _blaster *blaster

type gpio int
type pwm float64

type servoPkg struct {
	servo *Servo
	add   bool
}

func init() {
	_blaster = &blaster{
		buffer:  make(chan string),
		done:    make(chan struct{}),
		servos:  make(chan servoPkg),
		rate:    make(chan time.Duration),
		_servos: make(map[gpio]*Servo),
	}

	if err := _blaster.start(); err != nil {
		if err == errPiBlasterNotFound {
			log.Println("WARNING:", err, "\n\t(servo will continue with pi-blaster disabled)")
			noPiBlaster()
			if err := _blaster.start(); err != nil {
				panic(err)
			}
		} else {
			panic(err)
		}
	}
}

// noPiBlaster stops this package from sending text to /dev/pi-blaster. Useful
// for debugging in devices without pi-blaster installed.
func noPiBlaster() {
	_blaster.disabled = true
}

// hasBlaster checks if pi-blaster is running in the system. It depends on
// /bin/sh and pgrep.
func hasBlaster() bool {
	cmd := exec.Command("/bin/sh", "-c", "pgrep pi-blaster")
	if err := cmd.Run(); err != nil {
		return false
	}
	return true
}

var (
	// errPiBlasterNotFound is thrown when an instance of pi-blaster could not
	// be found on the system.
	errPiBlasterNotFound = fmt.Errorf("pi-blaster was not found running: start pi-blaster to avoid this error")
)

// start runs a goroutine to send data to pi-blaster. If NoPiBlaster was
// called, the data is sent to ioutil.Discard.
func (b *blaster) start() error {
	if !b.disabled && !hasBlaster() {
		return errPiBlasterNotFound
	}

	b.manager(b.done)

	return nil
}

// manager keeps track of changes to servos and flushes the data to pi-blaster.
// The flush will happen only if there was a change in the servos data.
// Everytime the data is flushed, the variable is emptied.
func (b *blaster) manager(done <-chan struct{}) {
	data := make(map[gpio]pwm)

	updateCh := time.NewTicker(3 * time.Millisecond)
	flushCh := time.NewTicker(40 * time.Millisecond)

	var ws sync.WaitGroup
	b.ws = &ws
	b.ws.Add(1)

	go func() {
```

```
			defer b.ws.Done()
			for {
				select {
				case <-done:
					return
				case pkg := <-b.servos:
					servo := pkg.servo
					if pkg.add {
						b._servos[servo.pin] = servo
					} else {
						delete(b._servos, servo.pin)
						data[servo.pin] = 0.0
					}
					updateCh.Stop()
					factor := math.Log10(float64(len(b._servos)+1))*3 + 1
					updateCh = time.NewTicker(time.Duration(factor) * 3 * time.Millisecond)
				case <-updateCh.C:
					for _, servo := range b._servos {
						if !servo.isIdle() {
							pin, pwm := servo.pwm()
							data[pin] = pwm
						}
					}
				case rate := <-b.rate:
					flushCh.Stop()
					flushCh = time.NewTicker(rate)
				case <-flushCh.C:
					if len(data) != 0 {
						b.flush(data)
						data = make(map[gpio]pwm)
					}
				}
			}
		}()
}

// subscribe adds a Servo reference to the manager.
func (b *blaster) subscribe(servo *Servo) {
		b.servos <- servoPkg{servo, true}
}

// unsubscribe removes a Servo reference from the manager.
func (b *blaster) unsubscribe(servo *Servo) {
		b.servos <- servoPkg{servo, false}
}

// Rate changes the rate that data is flushed to pi-blaster (default: 40ms).
// This can be changed on-the-fly.
func Rate(r time.Duration) {
		_blaster.rate <- r
}

// Close cleans up the servo package. Make sure to call this in your main
// goroutine.
func Close() {
		if _blaster == nil {
			return
		}
		_blaster.close()
}

// close stops blaster if it was started.
func (b *blaster) close() {
		b.write("*=0.0")
		close(b.done)
		b.ws.Wait()
}

// flush parses the data into "PIN=PWM PIN=PWM" format.
func (b *blaster) flush(data map[gpio]pwm) {
		s := new(strings.Builder)

		for pin, pwm := range data {
			fmt.Fprintf(s, " %d=%.6f", pin, pwm)
		}

		if s.Len() == 0 {
			return
		}

		b.write(s.String())
}

// write sends a string s to the designated io.Writer.
func (b *blaster) write(s string) {
		w := ioutil.Discard

		if !b.disabled {
			const pipepath = "/dev/pi-blaster"
			f, err := os.OpenFile(pipepath,
				os.O_WRONLY, os.ModeNamedPipe)
			if err != nil {
				panic(err)
```

```
		}
		defer f.Close()
		w = f
	}

	fmt.Fprintf(w, "%s\n", s)
	//fmt.Fprintf(os.Stdout, "%s\n", s)
}
```

D.2.4 *servo/servo_test.go*

```
// +build !live

package servo

import (
	"fmt"
	"sync"
	"testing"
	"time"
)

func init() {
	if hasBlaster() {
		fmt.Println("Found pi-blaster running.")
		fmt.Println("The test will not send anything to pi-blaster.")
		noPiBlaster()
	}
}

func TestServo(t *testing.T) {
	s := &Servo{
		Flags: Centered | Normalized,
	}

	if !s.Flags.is(Centered) {
		t.Error("Flags was not set to Centered")
	}
	if !s.Flags.is(Normalized) {
		t.Error("Flags was not set to Normalized")
	}
}

func TestConnect(t *testing.T) {
	const gpio = 99
	s := New(gpio)
	err := s.Connect()
	if err != nil {
		t.Fatal(err)
	}
	defer s.Close()

	if s.pin != gpio {
		t.Errorf("GPIO does not match, got: %d, want: %d", s.pin, gpio)
	}
	name := fmt.Sprintf("Servo%d", gpio)
	if s.Name != name {
		t.Errorf("Name does not match, got: %q, want: %q", s.Name, name)
	}
}

func TestServo_Position(t *testing.T) {
	const gpio = 99
	s := New(gpio)
	err := s.Connect()
	if err != nil {
		t.Fatal(err)
	}
	defer s.Close()

	const want = 59.6
	s.position = want
	got := s.Position()
	if got != want {
		t.Errorf("positions do not match, got: %.2f, want: %.2f", got, want)
	}

	t.Run("Centered", func(t *testing.T) {
		s.Flags = Centered
		got := s.Position()
		if got != want-90 {
			t.Errorf("positions do not match, got: %.2f, want: %.2f", got, want-90)
		}
	})

	t.Run("Normalized", func(t *testing.T) {
		s.Flags = Normalized
```

```go
		got := s.Position()
		if got != want/90 {
			t.Errorf("positions do not match, got: %.2f, want: %.2f", got, want/90)
		}
	})

	t.Run("Centered | Normalized", func(t *testing.T) {
		s.Flags = Centered | Normalized
		got := s.Position()
		if got != (want-90)/90 {
			t.Errorf("positions do not match, got: %.2f, want: %.2f", got, (want-90)/90)
		}
	})
}

func TestServo_MoveTo(t *testing.T) {
	// map[input]want
	tests := map[float64]float64{
		0:    0,
		10:   10,
		200:  180,
		-200: 0,
	}

	const gpio = 99
	s := New(gpio)
	err := s.Connect()
	if err != nil {
		t.Fatal(err)
	}
	defer s.Close()

	for input, want := range tests {
		s.moveTo(input)
		got := s.target
		if got != want {
			t.Errorf("Servo.moveTo(%.2f) -> got: %.2f, want: %.2f", input, got, want)
		}
	}

	t.Run("Concurrent", func(t *testing.T) {
		var wg sync.WaitGroup

		wg.Add(5)
		for i := 0; i < 5; i++ {
			go func(i int) {
				defer wg.Done()
				for j := 0; j < 30; j++ {
					s.moveTo(float64(i + j))
				}
			}(i)

		}
		wg.Wait()
	})
}

func TestServo_Reach(t *testing.T) {
	const gpio = 99
	s := New(gpio)
	err := s.Connect()
	if err != nil {
		t.Fatal(err)
	}
	defer s.Close()
	done := make(chan struct{})

	// Move to 180 degrees, but override concurrently to 0 when it reaches 110
	// degrees.
	s.moveTo(180)

	var wg sync.WaitGroup

	wg.Add(1)
	go func() {
		defer wg.Done()
		defer close(done)
		s.Wait()
	}()

	wg.Add(1)
	go func() {
		defer wg.Done()
		b := true
		for {
			select {
			case <-done:
				want := 0.0
				got := s.Position()
				if got != want {
					t.Errorf("Servo.moveTo(%.2f) -> got: %.2f, want: %.2f", 0.0, got, want)
				}
				return
```

```
					default:
						if b && s.Position() >= 110 {
							s.moveTo(0)
							b = false
						}
					}
				}
	}()

	<-done
	wg.Wait()
}

func BenchmarkServo_Reach(b *testing.B) {
	n := 100
	degrees := 2.0
	servos := make([]*Servo, 0, n)

	for i := 0; i < n; i++ {
		s := New(i)
		err := s.Connect()
		if err != nil {
			b.Fatalf("servos[%d] -> %v", i, err)
		}
		defer s.Close()
		servos = append(servos, s)
	}

	var wg sync.WaitGroup
	wg.Add(n)

	b.Logf("This benchmark should read aprox %.0f ns/op", 0.19/60.0*degrees*float64(time.Second))

	b.ResetTimer()
	for j := 0; j < n; j++ {
		go func(j int) {
			defer wg.Done()

			for i := 0; i < b.N; i++ {
				servos[j].position = 0
				servos[j].moveTo(degrees)
				servos[j].Wait()
			}
		}(j)
	}
	wg.Wait()
}

func BenchmarkServo_PWM(b *testing.B) {
	servo := New(1)
	err := servo.Connect()
	if err != nil {
		b.Fatalf("%v -> %v", servo, err)
	}
	defer servo.Close()

	servo.position = 0
	servo.moveTo(180)

	var wg sync.WaitGroup
	wg.Add(100)

	b.ResetTimer()
	for j := 0; j < 100; j++ {
		go func(j int) {
			defer wg.Done()

			for i := 0; i < b.N; i++ {
				servo.pwm()
			}
		}(j)
	}
	wg.Wait()

}

func TestServo_Stop(t *testing.T) {
	const gpio = 99
	s := New(gpio)
	err := s.Connect()
	if err != nil {
		t.Fatal(err)
	}
	defer s.Close()
	done := make(chan struct{})

	// Move to 180 degrees, but override concurrently to 0 when it reaches 110
	// degrees.
	s.moveTo(180)

	var wg sync.WaitGroup

	wg.Add(1)
```

```go
	go func() {
		defer wg.Done()
		defer close(done)
		s.Wait()
	}()

	wg.Add(1)
	go func() {
		defer wg.Done()
		b := true
		for {
			select {
			case <-done:
				got := s.Position()
				if got == 180 {
					t.Errorf("Servo.Stop() failed to stop -> got: %.2f", got)
				}
				t.Logf("Servo.Stop() stopped at: %.2f (requested: %.2f)", got, 110.0)
				return
			default:
				if b && s.Position() >= 110 {
					s.Stop()
					b = false
				}
			}
		}
	}()

	<-done
	wg.Wait()
}

func TestServo_Wait(t *testing.T) {
	const gpio = 99
	s := New(gpio)
	err := s.Connect()
	if err != nil {
		t.Fatal(err)
	}
	defer s.Close()

	// Move to 180 degrees and wait until finished.
	degrees := 180.0
	s.moveTo(degrees)

	var wg sync.WaitGroup

	wg.Add(1)
	// Test a concurrent waiter.
	go func() {
		defer wg.Done()
		s.Wait()
	}()

	start := time.Now()
	s.Wait()
	elapsed := time.Since(start)

	_t := time.Duration(degrees/s.step*1000) * time.Millisecond
	const tolerance = 50 * time.Millisecond
	min := _t - tolerance
	max := _t + tolerance
	if elapsed < min || elapsed > max {
		t.Errorf("it should take between %v and %v to move %.2f degrees, got: %v", min, max, degrees, elapsed)
	}

	wg.Wait()

	done := make(chan struct{})

	go func() {
		defer close(done)
		s.moveTo(degrees)
		<-time.After(500 * time.Millisecond)
		s.Wait()
	}()

	select {
	case <-time.After(1 * time.Second):
		t.Errorf("Wait timeout after 1 second")
	case <-done:
	}
}

func TestClamp(t *testing.T) {
	// map[input]want
	tests := map[float64]float64{
		0:   0,
		10:  1,
		-10: -1,
		0.5: 0.5,
	}

```

```
	for input, want := range tests {
		got := clamp(input, -1, 1)
		if got != want {
			t.Errorf("clam(%.2f, -1, 1) -> got: %.2f, want: %.2f", input, got, want)
		}
	}
}
```

D.2.5 *servo/blaster.go*

```
package servo

import (
	"fmt"
	"io/ioutil"
	"log"
	"math"
	"os"
	"os/exec"
	"strings"
	"sync"
	"time"
)

type blaster struct {
	disabled bool
	buffer   chan string
	done     chan struct{}
	servos   chan servoPkg
	_servos  map[gpio]*Servo

	rate chan time.Duration

	ws *sync.WaitGroup
}

var _blaster *blaster

type gpio int
type pwm float64

type servoPkg struct {
	servo *Servo
	add   bool
}

func init() {
	_blaster = &blaster{
		buffer:  make(chan string),
		done:    make(chan struct{}),
		servos:  make(chan servoPkg),
		rate:    make(chan time.Duration),
		_servos: make(map[gpio]*Servo),
	}

	if err := _blaster.start(); err != nil {
		if err == errPiBlasterNotFound {
			log.Println("WARNING:", err, "\n\t(servo will continue with pi-blaster disabled)")
			noPiBlaster()
			if err := _blaster.start(); err != nil {
				panic(err)
			}
		} else {
			panic(err)
		}
	}
}

// noPiBlaster stops this package from sending text to /dev/pi-blaster. Useful
// for debugging in devices without pi-blaster installed.
func noPiBlaster() {
	_blaster.disabled = true
}

// hasBlaster checks if pi-blaster is running in the system. It depends on
// /bin/sh and pgrep.
func hasBlaster() bool {
	cmd := exec.Command("/bin/sh", "-c", "pgrep pi-blaster")
	if err := cmd.Run(); err != nil {
		return false
	}
	return true
}

var (
	// errPiBlasterNotFound is thrown when an instance of pi-blaster could not
	// be found on the system.
```

```go
	errPiBlasterNotFound = fmt.Errorf("pi-blaster was not found running: start pi-blaster to avoid this error")
)

// start runs a goroutine to send data to pi-blaster. If NoPiBlaster was
// called, the data is sent to ioutil.Discard.
func (b *blaster) start() error {
	if !b.disabled && !hasBlaster() {
		return errPiBlasterNotFound
	}

	b.manager(b.done)

	return nil
}

// manager keeps track of changes to servos and flushes the data to pi-blaster.
// The flush will happen only if there was a change in the servos data.
// Everytime the data is flushed, the variable is emptied.
func (b *blaster) manager(done <-chan struct{}) {
	data := make(map[gpio]pwm)

	updateCh := time.NewTicker(3 * time.Millisecond)
	flushCh := time.NewTicker(40 * time.Millisecond)

	var ws sync.WaitGroup
	b.ws = &ws
	b.ws.Add(1)

	go func() {
		defer b.ws.Done()
		for {
			select {
			case <-done:
				return
			case pkg := <-b.servos:
				servo := pkg.servo
				if pkg.add {
					b._servos[servo.pin] = servo
				} else {
					delete(b._servos, servo.pin)
					data[servo.pin] = 0.0
				}
				updateCh.Stop()
				factor := math.Log10(float64(len(b._servos)+1))*3 + 1
				updateCh = time.NewTicker(time.Duration(factor) * 3 * time.Millisecond)
			case <-updateCh.C:
				for _, servo := range b._servos {
					if !servo.isIdle() {
						pin, pwm := servo.pwm()
						data[pin] = pwm
					}
				}
			case rate := <-b.rate:
				flushCh.Stop()
				flushCh = time.NewTicker(rate)
			case <-flushCh.C:
				if len(data) != 0 {
					b.flush(data)
					data = make(map[gpio]pwm)
				}
			}
		}
	}()
}

// subscribe adds a Servo reference to the manager.
func (b *blaster) subscribe(servo *Servo) {
	b.servos <- servoPkg{servo, true}
}

// unsubscribe removes a Servo reference from the manager.
func (b *blaster) unsubscribe(servo *Servo) {
	b.servos <- servoPkg{servo, false}
}

// Rate changes the rate that data is flushed to pi-blaster (default: 40ms).
// This can be changed on-the-fly.
func Rate(r time.Duration) {
	_blaster.rate <- r
}

// Close cleans up the servo package. Make sure to call this in your main
// goroutine.
func Close() {
	if _blaster == nil {
		return
	}
	_blaster.close()
}

// close stops blaster if it was started.
func (b *blaster) close() {
	b.write("*=0.0")
```

```
		close(b.done)
		b.ws.Wait()
}

// flush parses the data into "PIN=PWM PIN=PWM" format.
func (b *blaster) flush(data map[gpio]pwm) {
		s := new(strings.Builder)

		for pin, pwm := range data {
				fmt.Fprintf(s, " %d=%.6f", pin, pwm)
		}

		if s.Len() == 0 {
				return
		}

		b.write(s.String())
}

// write sends a string s to the designated io.Writer.
func (b *blaster) write(s string) {
		w := ioutil.Discard

		if !b.disabled {
				const pipepath = "/dev/pi-blaster"
				f, err := os.OpenFile(pipepath,
						os.O_WRONLY, os.ModeNamedPipe)
				if err != nil {
						panic(err)
				}
				defer f.Close()
				w = f
		}

		fmt.Fprintf(w, "%s\n", s)
		//fmt.Fprintf(os.Stdout, "%s\n", s)
}
```

D.2.6 servo/example_test.go

```
//+build !live

package servo_test

import (
		"fmt"
		"log"

		"github.com/cgxeiji/servo"
)

func Example() {
		// Use servo.Close() to close the connection of all servos and pi-blaster.
		defer servo.Close()

		// If you want to move the servos, make sure that pi-blaster is running.
		// For example, start pi-blaster as:
		// $ sudo pi-blaster --gpio 14 --pcm

		// Create a new servo connected to gpio 14.
		myServo := servo.New(14)
		// (optional) Initialize the servo with your preferred values.
		// myServo.Flags = servo.Normalized | servo.Centered
		myServo.MinPulse = 0.05 // Set the minimum pwm pulse width (default: 0.05).
		myServo.MaxPulse = 0.25 // Set the maximum pwm pulse width (default: 0.25).
		myServo.SetPosition(90) // Set the initial position to 90 degrees.
		myServo.SetSpeed(0.2)   // Set the speed to 20% (default: 1.0).
		// NOTE: The maximum speed of the servo is 0.19s/60degrees.
		// (optional) Set a verbose name.
		myServo.Name = "My Servo"

		// Print the information of the servo.
		fmt.Println(myServo)

		// Connect the servo to the daemon.
		err := myServo.Connect()
		if err != nil {
				log.Fatal(err)
		}

		// (optional) Use myServo.Close() to close the connection to a specific
		// servo. You still need to close the connection to pi-blaster with
		// `servo.Close()`.
		defer myServo.Close()

		myServo.SetSpeed(0.5) // Set the speed to half. This is concurrent-safe.
		myServo.MoveTo(180)   // This is a non-blocking call.
```

```go

	/* do some work */

	myServo.Wait() // Call Wait() to sync with the servo.

	// MoveTo() returns a Waiter interface that can be used to move and wait on
	// the same line.
	myServo.MoveTo(0).Wait() // This is a blocking call.

	// Output:
	// servo "My Servo" connected to gpio(14) [flags: ( NONE )]
}
```

D.2.7 servo/package_test.go

```go
// +build !live

package servo_test

import (
	"sync"
	"testing"
	"time"

	"github.com/cgxeiji/servo"
)

func initServo(t *testing.T) *servo.Servo {
	s := servo.New(99)
	err := s.Connect()
	if err != nil {
		t.Fatal(err)
	}

	s.Name = "Tester"
	return s
}

func TestExportConnect(t *testing.T) {
	s := initServo(t)
	defer s.Close()

	want := `servo "Tester" connected to gpio(99) [flags: ( NONE )]`
	got := s.String()

	if got != want {
		t.Errorf("error connecting servo\ngot:\n%v\nwant:\n%v", got, want)
	}
}

func TestExportServo_MoveTo(t *testing.T) {
	s := initServo(t)
	defer s.Close()

	var wg sync.WaitGroup

	// Move to 180 degrees and wait until finished.
	degrees := 180.0
	s.MoveTo(degrees)

	wg.Add(1)
	// Test a concurrent waiter.
	go func() {
		defer wg.Done()
		s.Wait()
	}()

	start := time.Now()
	s.Wait()
	elapsed := time.Since(start)

	_t := time.Duration(degrees/315.7*1000) * time.Millisecond
	const tolerance = 50 * time.Millisecond
	min := _t - tolerance
	max := _t + tolerance

	if elapsed < min || elapsed > max {
		t.Errorf("it should take between %v and %v to move %.2f degrees, got: %v", min, max, degrees, elapsed)
	}

	got := s.Position()
	if got != degrees {
		t.Errorf("did not move to %.2f, got: %.2f", degrees, got)
	}

	wg.Wait()
}
```

D.2.8 *servo/live_test.go*

```go
// +build live

package servo_test

import (
	"testing"
	"time"

	"github.com/cgxeiji/servo"
)

func init() {
	if !servo.HasBlaster() {
		panic("start pi-blaster before running the live test!")
	}
}

func TestLive(t *testing.T) {
	test, err := servo.Connect(14)
	if err != nil {
		t.Fatalf("Could not connect servo to pin 14, got:\n%v", err)
	}
	defer func() {
		test.Speed(0.05)
		test.MoveTo(90)
		test.Wait()
		test.Close()
	}()

	test.MoveTo(180)
	start := time.Now()
	test.Wait()
	elapsed := time.Since(start)

	_t := time.Duration(570) * time.Millisecond
	const tolerance = 50 * time.Millisecond
	min := _t - tolerance
	max := _t + tolerance

	t.Logf("took %v to move %.2f degrees", elapsed, 180.0)
	if elapsed < min || elapsed > max {
		t.Errorf("it should take between %v and %v to move %.2f degrees, got: %v", min, max, 180.0, elapsed)
	}
	if test.Position() != 180 {
		t.Errorf("servo position got: %.2f, want: %.2f", test.Position(), 180.0)
	}

	time.Sleep(500 * time.Millisecond)
	test.Speed(0.5)

	test.MoveTo(0)
	test.MoveTo(90)
	test.MoveTo(0)
	test.Wait()
	if test.Position() != 0 {
		t.Errorf("servo position got: %.2f, want: %.2f", test.Position(), 0.0)
	}
	time.Sleep(500 * time.Millisecond)
}

func TestLive2(t *testing.T) {
	test, err := servo.Connect(15)
	if err != nil {
		t.Fatalf("Could not connect servo to pin 15, got:\n%v", err)
	}
	defer func() {
		test.Speed(0.05)
		test.MoveTo(90)
		test.Wait()
		test.Close()
	}()

	test.Speed(0.5)
	test.MoveTo(180)
	start := time.Now()
	test.Wait()
	elapsed := time.Since(start)

	_t := time.Duration(570*2) * time.Millisecond
	const tolerance = 50 * time.Millisecond
	min := _t - tolerance
	max := _t + tolerance

	t.Logf("took %v to move %.2f degrees", elapsed, 180.0)
	if elapsed < min || elapsed > max {
		t.Errorf("it should take between %v and %v to move %.2f degrees, got: %v", min, max, 180.0, elapsed)
	}
	if test.Position() != 180 {
		t.Errorf("servo position got: %.2f, want: %.2f", test.Position(), 180.0)
```

```
	}

	time.Sleep(500 * time.Millisecond)
	test.Speed(0.25)

	test.MoveTo(0)
	test.MoveTo(90)
	test.MoveTo(0)
	test.Wait()
	if test.Position() != 0 {
		t.Errorf("servo position got: %.2f, want: %.2f", test.Position(), 0.0)
	}
	time.Sleep(500 * time.Millisecond)
}
```

D.2.9 servo/export_test.go

```
package servo

var HasBlaster = hasBlaster
```

D.2.10 servo/stress_test.go

```
// +build !race,!live

package servo

import (
	"fmt"
	"sync"
	"testing"
	"time"
)

func TestStress(t *testing.T) {
	degrees := 180.0
	_t := time.Duration(degrees/315.7*1000) * time.Millisecond
	const tolerance = 50 * time.Millisecond
	min := _t - tolerance
	max := _t + tolerance

	for n := 100; n <= 10000; n *= 10 {
		t.Run(fmt.Sprintf("%dServos", n), func(t *testing.T) {
			servos := make([]*Servo, 0, n)
			times := make([]time.Duration, 0, n)

			for i := 0; i < n; i++ {
				s := New(i)
				err := s.Connect()
				if err != nil {
					t.Fatalf("servos[%d] -> %v", i, err)
				}
				defer s.Close()
				servos = append(servos, s)
			}

			var wg sync.WaitGroup
			timeout := make(chan time.Duration)

			wg.Add(n)

			for i := 0; i < n; i++ {
				go func(i int) {
					defer wg.Done()
					times := make([]time.Duration, 0, 3)
					tests := []float64{180, 0, 180}

					for _, d := range tests {
						start := time.Now()
						servos[i].moveTo(d)
						servos[i].Wait()
						elapsed := time.Since(start)
						if elapsed < min || elapsed > max {
							times = append(times, elapsed)
						}
					}
					var sum time.Duration
					for _, t := range times {
						sum += t
					}
					if sum > 0 {
						timeout <- sum / time.Duration(len(times))
```

```go
					}
				}(i)
			}

			go func() {
				wg.Wait()
				close(timeout)
			}()

			for t := range timeout {
				times = append(times, t)
			}

			fn := len(times)

			if fn != 0 {
				sum := time.Duration(0)
				for _, t := range times {
					sum += t
				}
				mean := sum / time.Duration(fn)

				ratio := float64(fn) / float64(n)
				if n > 100 {
					t.Logf("%d out of %d (%.2f%%) servos failed with a mean time of %v (it should take between %v and %v to move %.2f degrees)",
						fn, n, ratio*100.0, mean, min, max, degrees)
				} else {
					t.Errorf("%d out of %d (%.2f%%) servos failed with a mean time of %v (it should take between %v and %v to move %.2f degrees)",
						fn, n, ratio*100.0, mean, min, max, degrees)
				}
			}
		})
	}
}
```

D.3 Module: ring

D.3.1 License

D.3.2 *ring/ring.go*

```go
package ring

import (
	"fmt"
	"image/color"
	"math"
	"os"

	ws2811 "github.com/rpi-ws281x/rpi-ws281x-go"
)

// Ring represents the WS2811 LED device.
type Ring struct {
	device    *ws2811.WS2811
	layers    []Pixeler
	ledArc    float64
	ledOffset int
	offset    float64
	opt       *Options
}

// Pixeler is an interface that returns the color of a pixel at a specific
// location, with a set resolution.
type Pixeler interface {
	Pixel(int) color.Color
	Options() *LayerOptions
}

// Options is the list of ring options.
type Options struct {
	// LedCount is the number of LEDs in the ring.
	LedCount int
	// MinBrightness is the minimum output of the LED> Goes from 0 to 255
	// (default: 0).
	// MaxBrightness is the maximum output of the LED. Goes from 0 to 255
	// (default: 64).
	//
	// The color will be scaled to these values. For example, color.RGBA{255,
	// 255, 255, 255} will output led(R: 128, G: 128, B: 128) if MaxBrightness
	// is set to 128, and color.RGBA(0, 0, 0, 0) will output led(R: 10, G: 10,
	// B: 10) if MinBrightness is set to 10.
	MinBrightness, MaxBrightness int
	// GpioPin is the GPIO pin on the Raspberry Pi with PWM output (default:
	// GPIO 18). *Do not confuse with the physical pin number*
	GpioPin int
}

// New creates a new LED ring with given options.
func New(options *Options) (*Ring, error) {
	if os.Getuid() != 0 {
		return nil, fmt.Errorf("ring: rpi-ws281x needs root permissions (try running as sudo)")
	}
```

```go

	opt := ws2811.DefaultOptions
	if options.LedCount != 0 {
		opt.Channels[0].LedCount = options.LedCount
	}
	if options.MaxBrightness != 0 {
		opt.Channels[0].Brightness = options.MaxBrightness
	}
	if options.GpioPin != 0 {
		opt.Channels[0].GpioPin = options.GpioPin
	}

	dev, err := ws2811.MakeWS2811(&opt)
	if err != nil {
		return nil, fmt.Errorf("ring: could not create ws2811 device: %w", err)
	}

	r := &Ring{
		device: dev,
		ledArc: 2 * math.Pi / float64(options.LedCount),
		opt:    options,
	}

	if err := r.device.Init(); err != nil {
		return nil, fmt.Errorf("ring: could not start ws2811 device: %w", err)
	}

	return r, nil
}

// Render updates the LED ring.
func (r *Ring) Render() error {
	pixels := make([]color.Color, r.Size())
	pixel := make([]color.Color, len(r.layers))

	for i := range r.device.Leds(0) {
		for j, l := range r.layers {
			switch l.Options().ContentMode {
			case ContentTile:
				pixel[j] = l.Pixel(i)
			case ContentCrop:
				if i < l.Options().Resolution {
					pixel[j] = l.Pixel(i)
				} else {
					pixel[j] = color.Transparent
				}
			case ContentScale:
				pixel[j] = l.Pixel(scale(i, r.Size(), l.Options().Resolution))
			}
		}
		pixels[i] = blendOver(pixel...)
	}
	rotInt := math.Floor(r.offset)
	rotFloat := r.offset - rotInt
	for i := range r.device.Leds(0) {
		r.device.Leds(0)[i] = serialize(lerp(int(rotInt)+i, pixels, rotFloat))
	}

	if err := r.device.Render(); err != nil {
		return err
	}

	return nil
}

func lerp(i int, pixels []color.Color, alpha float64) color.Color {
	return blendLerp(pixels[mod(i, len(pixels))], pixels[mod(i+1, len(pixels))], alpha)
}

// AddLayer adds a drawable layer to the ring.
func (r *Ring) AddLayer(l Pixeler) {
	r.layers = append(r.layers, l)
}

// Close turns off the LED ring and closes the device.
func (r *Ring) Close() {
	r.TurnOff()
	r.device.Fini()
}

// TurnOff tuns off the LED ring without closing the device.
func (r *Ring) TurnOff() {
	for i := range r.device.Leds(0) {
		r.device.Leds(0)[i] = 0
	}
	r.device.Render()
}

// Size returns the total number of LEDs of the ring.
func (r *Ring) Size() int {
	return r.opt.LedCount
}

```

```go
// Offset sets an angular offset (in radians) to render the layers.
// A positive angle rotates counter-clockwise.
func (r *Ring) Offset(rotation float64) {
	if rotation < 0 {
		r.ledOffset = int(math.Ceil(rotation / r.ledArc))
	} else {
		r.ledOffset = int(math.Floor(rotation / r.ledArc))
	}
	r.offset = rotation / r.ledArc
}

func scale(v, fmax, tmax int) int {
	return v * tmax / fmax
}
```

D.3.3 *ring/layer.go*

```go
package ring

import (
	"errors"
	"image/color"
	"math"
	"sync"
)

// Layer represents a drawable layer of the LED ring.
type Layer struct {
	pixels []color.Color

	pixArc   float64 // pixel arc in radians
	rotFloat float64 // float part of rotation in radians
	rotInt   int     // integer part of rotation in radians

	opt    *LayerOptions
	buffer []color.Color
	lock   *sync.RWMutex
}

// LayerOptions is the list of options of a layer.
type LayerOptions struct {
	// Resolution sets the number of pixels a layer has. Usually, this is set
	// to the same number of LEDs the ring has.
	Resolution int
	// ContentMode sets how the layer will be rendered (default: Tile).
	ContentMode ContentMode
}

// ContentMode defines how the layer will be rendered.
type ContentMode uint8

const (
	// ContentTile sets the layer to crop its content if it is larger that the ring
	// and to repeat the content.
	ContentTile ContentMode = iota
	// ContentCrop sets the layer to crop its content if it is larger than the ring
	// and does not repeat the content.
	ContentCrop
	// ContentScale sets the layer to scale up or down its content to fit the ring.
	ContentScale
)

var (
	// ErrZeroResolution defines the error when a layer is initialized with
	// resolution 0.
	ErrZeroResolution = errors.New("ring: resolution of new layer is 0")
)

// NewLayer creates a new drawable layer.
func NewLayer(options *LayerOptions) (*Layer, error) {
	if options.Resolution == 0 {
		return nil, ErrZeroResolution
	}

	l := &Layer{
		pixels: make([]color.Color, options.Resolution),
		buffer: make([]color.Color, options.Resolution),
		pixArc: 2 * math.Pi / float64(options.Resolution),
		opt:    options,
		lock:   new(sync.RWMutex),
	}
	l.SetAll(color.Transparent)
	l.update()

	return l, nil
}

```

```go
// SetAll sets all the pixels of a layer to an uniform color.
func (l *Layer) SetAll(c color.Color) {
	for i := range l.pixels {
		l.pixels[i] = c
	}
	l.update()
}

// SetPixel sets the color of a single pixel in the layer.
func (l *Layer) SetPixel(i int, c color.Color) {
	l.pixels[i] = c
	l.update()
}

// Rotate sets the rotation of the layer. A positive angle makes a counter-clockwise rotation.
func (l *Layer) Rotate(angle float64) {
	rotArc := angle / l.pixArc
	rotInt := math.Floor(rotArc)
	l.rotFloat = rotArc - rotInt
	l.rotInt = int(rotInt)

	l.update()
}

// pixelRotated returns the color of the pixel at position i adjusted for the
// rotation of the layer.
func (l *Layer) pixelRotated(i int) (c color.Color) {
	i += l.rotInt
	c = blendLerp(l.pixelRaw(i), l.pixelRaw(i+1), l.rotFloat)

	return c
}

// Pixel returns the color of the pixel at position i, with layer
// transformations.
func (l *Layer) Pixel(i int) (c color.Color) {
	l.lock.RLock()
	defer l.lock.RUnlock()
	return l.buffer[mod(i, l.opt.Resolution)]
}

// Options returns the options of the layer.
func (l *Layer) Options() *LayerOptions {
	return l.opt
}

func (l *Layer) update() {
	l.lock.Lock()
	defer l.lock.Unlock()
	for i := range l.pixels {
		l.buffer[i] = l.pixelRotated(i)
	}
}

// pixelRaw returns the color of the pixelRaw at position i.
func (l *Layer) pixelRaw(i int) (c color.Color) {
	return l.pixels[mod(i, l.opt.Resolution)]
}

func mod(p, n int) (r int) {
	r = p % n
	if r < 0 {
		r += n
	}

	return r
}
```

D.3.4 *ring/color.go*

```go
package ring

import (
	"image/color"
)

// serialize transforms color information to uint32 with the shape 0x00RRGGBB
func serialize(c color.Color) uint32 {
	r, g, b, _ := c.RGBA()

	return ((r >> 8) << 16) |
		((g >> 8) << 8) |
		(b >> 8)
}

// blendOver blends multiple colors using the over operator and returns an
// alpha pre-multiplied color. The first color is considered to be at the
```

```go
// bottom and the last color is considered to be at the top.
func blendOver(cs ...color.Color) (blend *color.RGBA) {
	over := func(a, b, delta uint32) uint8 {
		return uint8((a + b*delta/0xFFFF) >> 8)
	}
	blend = &color.RGBA{0, 0, 0, 0}
	for _, c := range cs {
		r, g, b, a := c.RGBA()
		bR, bG, bB, bA := blend.RGBA()
		delta := (0xFFFF - a)

		blend.R = over(r, bR, delta)
		blend.G = over(g, bG, delta)
		blend.B = over(b, bB, delta)
		blend.A = over(a, bA, delta)
	}

	return blend
}

// blendLerp blends two colors by linearly interpolating between them given the
// amount l: (0.0 to 1.0) -> (a to b).
func blendLerp(a, b color.Color, l float64) (blend *color.RGBA) {
	lerp := func(a, b, l uint32) uint8 {
		return uint8((a - (a-b)*l/0xFFFF) >> 8)
	}

	aR, aG, aB, aA := a.RGBA()
	bR, bG, bB, bA := b.RGBA()

	l16 := uint32(l * 0xFFFF)

	blend = &color.RGBA{
		R: lerp(aR, bR, l16),
		G: lerp(aG, bG, l16),
		B: lerp(aB, bB, l16),
		A: lerp(aA, bA, l16),
	}

	return blend
}
```

D.3.5 *ring/color_test.go*

```go
package ring

import (
	"fmt"
	"image/color"
	"testing"
)

func TestMod(t *testing.T) {
	const n = 12
	tests := []struct {
		idx  int
		want int
	}{
		{
			1,
			1,
		},
		{
			12,
			0,
		},
		{
			13,
			1,
		},
		{
			-1,
			11,
		},
	}

	for _, ts := range tests {
		t.Run(fmt.Sprintf("%d", ts.idx), func(t *testing.T) {
			got := mod(ts.idx, n)
			if got != ts.want {
				t.Errorf("got: %#v, want: %#v", got, ts.want)
			}
		})
	}
}

func TestSerialize(t *testing.T) {
```

```go
	tests := []struct {
		name  string
		color color.Color
		want  uint32
	}{
		{
			"rgb",
			color.NRGBA{0x16, 0x16, 0x16, 0xFF},
			0x161616,
		},
		{
			"alpha",
			color.NRGBA{0xFF, 0xFF, 0xFF, 0x32},
			0x323232,
		},
		{
			"16bit",
			color.NRGBA64{0x3214, 0x1234, 0x00FF, 0xFFFF},
			0x321200,
		},
		{
			"gray",
			color.Gray{0x10},
			0x101010,
		},
	}

	for _, ts := range tests {
		t.Run(ts.name, func(t *testing.T) {
			got := serialize(ts.color)
			if got != ts.want {
				t.Errorf("got: %#v, want: %#v", got, ts.want)
			}
		})
	}
}

func TestBlendOver(t *testing.T) {
	tests := []struct {
		name   string
		colors []color.Color
		want   color.RGBA
	}{
		{
			"single",
			[]color.Color{
				color.RGBA{0x15, 0x16, 0x17, 0x18},
			},
			color.RGBA{0x15, 0x16, 0x17, 0x18},
		},
		{
			"white over black",
			[]color.Color{
				color.RGBA{0x00, 0x00, 0x00, 0xFF},
				color.RGBA{0xFF, 0xFF, 0xFF, 0xFF},
			},
			color.RGBA{0xFF, 0xFF, 0xFF, 0xFF},
		},
		{
			"black over white",
			[]color.Color{
				color.RGBA{0xFF, 0xFF, 0xFF, 0xFF},
				color.RGBA{0x00, 0x00, 0x00, 0xFF},
			},
			color.RGBA{0x00, 0x00, 0x00, 0xFF},
		},
		{
			"red over green",
			[]color.Color{
				color.NRGBA{0x00, 0x80, 0x00, 0xFF},
				color.NRGBA{0x80, 0x00, 0x00, 0xA1},
			},
			color.RGBA{0x51, 0x2F, 0x00, 0xFF},
		},
	}

	for _, ts := range tests {
		t.Run(ts.name, func(t *testing.T) {
			got := *blendOver(ts.colors...)
			if got != ts.want {
				t.Errorf("got: %#v, want: %#v", got, ts.want)
			}
		})
	}
}

func TestBlendLerp(t *testing.T) {
	tests := []struct {
		name   string
		colorA color.Color
		colorB color.Color
		l      float64
		want   color.RGBA
```

```
	}{
		{
			"fullA",
			color.RGBA{128, 128, 0, 128},
			color.RGBA{0, 255, 255, 255},
			0.0,
			color.RGBA{128, 128, 0, 128},
		},
		{
			"fullB",
			color.RGBA{128, 128, 0, 128},
			color.RGBA{0, 255, 255, 255},
			1.0,
			color.RGBA{0, 255, 255, 255},
		},
		{
			"half",
			color.RGBA{128, 128, 0, 128},
			color.RGBA{0, 255, 255, 255},
			0.5,
			color.RGBA{64, 192, 127, 192},
		},
		{
			"quater",
			color.RGBA{128, 128, 0, 128},
			color.RGBA{0, 255, 255, 255},
			0.75,
			color.RGBA{32, 224, 191, 224},
		},
	}

	for _, ts := range tests {
		t.Run(ts.name, func(t *testing.T) {
			got := *blendLerp(ts.colorA, ts.colorB, ts.l)
			if got != ts.want {
				t.Errorf("got: %v, want: %v", got, ts.want)
			}
		})
	}
}
```

D.3.6 *ring/example_test.go*

```
package ring_test

import (
	"bufio"
	"fmt"
	"image/color"
	"log"
	"math"
	"os"
	"sync"
	"time"

	"github.com/cgxeiji/ring"
)

func Example() {
	// Initialize the ring.
	r, err := ring.New(&ring.Options{
		LedCount:      12,  // adjust this to the number of LEDs you have
		MaxBrightness: 180, // value from 0 to 255
	})
	r.Offset(-math.Pi / 3) // you can set a rotation offset for the ring
	if err != nil {
		log.Fatal(err)
	}
	// Make sure to properly close the ring.
	defer r.Close()

	// Create a new layer.  This will be a static white background.
	bg, err := ring.NewLayer(&ring.LayerOptions{
		Resolution:  1,                 // set to 1 pixel because it is a uniform color background
		ContentMode: ring.ContentScale, // this scales the pixel to the whole ring
	})
	if err != nil {
		log.Fatal(err)
	}
	// Set all pixels of the layer to white.
	bg.SetAll(color.White)
	// Add the layer to the ring.
	r.AddLayer(bg)

	// Create a mask layer.  This will fade the background.
	bgMask, err := ring.NewLayer(&ring.LayerOptions{
		Resolution: 1, // set to 1 pixel because it is a uniform mask
```

```
	})
	if err != nil {
		log.Fatal(err)
	}
	r.AddLayer(bgMask)

	// Render the ring.
	if err := r.Render(); err != nil {
		log.Fatal(err)
	}

	// Wait for 1 second to see the beauty of the freshly rendered layer.
	time.Sleep(1 * time.Second)

	// Create another layer.  This will set 3 pixels to red, green and blue,
	// and a hidden purple pixel with transparency of 200, that rotate
	// counter-clockwise.
	triRotate, err := ring.NewLayer(&ring.LayerOptions{
		Resolution: 48,
	})
	if err != nil {
		log.Fatal(err)
	}
	// We can immediately add the layer to the ring.  By default, new layers
	// are initialized with transparent pixels.  The new layer is added on top
	// of the previous layers.
	r.AddLayer(triRotate)

	// Set the colors.
	triRotate.SetPixel(0, color.NRGBA{128, 0, 0, 200})    // dark red
	triRotate.SetPixel(3, color.NRGBA{0, 128, 0, 200})    // dark green
	triRotate.SetPixel(6, color.NRGBA{0, 0, 128, 200})    // dark blue
	triRotate.SetPixel(24, color.NRGBA{128, 0, 255, 200}) // purple
	// Render the ring.
	if err := r.Render(); err != nil {
		log.Fatal(err)
	}

	// Wait for 1 second to see the beauty of both layers.
	time.Sleep(1 * time.Second)

	// Create another layer. This will set a pixel that will blink every 500ms.
	blink, err := ring.NewLayer(&ring.LayerOptions{
		Resolution:  3,               // we are going to set the 3rd pixel (index: 2) to blink
		ContentMode: ring.ContentCrop, // this crops the layer to avoid repetition
	})
	if err != nil {
		log.Fatal(err)
	}
	// Add the layer to the ring. This will be on top of the previous two
	// layers.
	r.AddLayer(blink)

	// Set the color. We can use any variable that implements the color.Color
	// interface.
	blink.SetPixel(2, color.CMYK{255, 0, 0, 0})
	// Render the ring.
	if err := r.Render(); err != nil {
		log.Fatal(err)
	}

	// Wait for 1 second and enjoy the view.
	time.Sleep(1 * time.Second)

	/* ANIMATION SETUP */
	done := make(chan struct{})   // this will cancel all animations
	render := make(chan struct{}) // this will request a concurrent-safe render
	var ws sync.WaitGroup         // this makes sure we close all goroutines

	/* render goroutine */
	ws.Add(1)
	go func() {
		defer ws.Done()
		for {
			select {
			case <-done:
				return
			case <-render:
				if err := r.Render(); err != nil {
					log.Fatal(err)
				}
			}
		}
	}()

	/* fading goroutine */
	ws.Add(1)
	go func() {
		defer ws.Done()
		c := color.NRGBA{0, 0, 0, 0}
		step := uint8(5)
		for {
			for a := uint8(0); a < 255; a += step {
```

```
				c.A = a
				bgMask.SetAll(c)
				select {
				case <-done:
					return
				case render <- struct{}{}:
				}
				time.Sleep(20 * time.Millisecond)
			}
			for a := uint8(255); a > 0; a -= step {
				c.A = a
				bgMask.SetAll(c)
				select {
				case <-done:
					return
				case render <- struct{}{}:
				}
				time.Sleep(20 * time.Millisecond)
			}
		}
	}()

	/* rotation goroutine */
	ws.Add(1)
	go func() {
		defer ws.Done()
		for {
			for a := 0.0; a < math.Pi*2; a += 0.01 {
				triRotate.Rotate(a)
				select {
				case <-done:
					return
				case render <- struct{}{}:
				}
				time.Sleep(20 * time.Millisecond)
			}
		}
	}()

	/* blinking goroutine */
	ws.Add(1)
	go func() {
		defer ws.Done()
		c := color.CMYK{255, 0, 0, 0}
		timer := time.NewTicker(500 * time.Millisecond)
		on := true
		for {
			select {
			case <-done:
				return
			case <-timer.C:
				if on {
					blink.SetPixel(2, color.Transparent)
					on = false
				} else {
					blink.SetPixel(2, c)
					on = true
				}
				select {
				case <-done:
					return
				case render <- struct{}{}:
				}
			}
		}
	}()

	fmt.Println("Press [ENTER] to exit")
	stdin := bufio.NewReader(os.Stdin)
	stdin.ReadString('\n')

	// Stop all animations
	close(done)
	// Wait for goroutines to exit
	ws.Wait()

	// Remember that we called a defer `r.Close()` at the beginning of the
	// code. This will turn off the LEDs and clean up the resources used by the
	// ring before exiting. Otherwise, the ring will stay on with the latest
	// render.
}
```

D.4 Module: PiCam

D.4.1 License

```
MIT License

Copyright (c) 2020 Eiji Onchi

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

D.4.2 *picam/bench_test.go*

```go
package picam

import (
	"fmt"
	"testing"
)

func BenchmarkPicam(b *testing.B) {
	benchmarks := []struct {
		format Format
	}{
		{YUV},
		{RGB},
		{Gray},
	}

	for _, bm := range benchmarks {
		b.Run(fmt.Sprintf("%s", bm.format), func(b *testing.B) {
			cam, err := New(640, 480, bm.format)
			if err != nil {
				b.Fatal(err)
			}
			defer cam.Close()
			_ = cam.ReadUint8()

			b.ResetTimer()
			for i := 0; i < b.N; i++ {
				_ = cam.ReadUint8()
			}
		})
	}
}
```

D.4.3 *picam/example_save_test.go*

```go
package picam_test

import (
	"image/png"
	"log"
	"os"

	"github.com/cgxeiji/picam"
)

func Example_save() {
```

```go
	cam, err := picam.New(640, 480, picam.YUV)
	if err != nil {
		log.Fatal(err)
	}
	defer cam.Close()

	img := cam.Read()

	f, err := os.Create("./image.png")
	if err != nil {
		log.Fatal(err)
	}
	defer f.Close()

	err = png.Encode(f, img)
	if err != nil {
		log.Fatal(err)
	}
}
```

D.4.4 picam/example_test.go

```go
package picam_test

import (
	"fmt"
	"log"

	"github.com/cgxeiji/picam"
)

func Example() {
	cam, err := picam.New(640, 480, picam.YUV)
	if err != nil {
		log.Fatal(err)
	}
	defer cam.Close()

	nFrames := 5
	fmt.Println("Reading", nFrames, "frames:")

	for i := 0; i < nFrames; i++ {
		// Get an image.Image
		img := cam.Read()

		/* do something with img */
		fmt.Println("got", img.Bounds().Size())

		// Or get a raw []uint8 slice
		raw := cam.ReadUint8()

		/* do something with img */
		fmt.Println("read", len(raw), "bytes")
	}

	// Output:
	// Reading 5 frames:
	// got (640,480)
	// read 460800 bytes
	// got (640,480)
	// read 460800 bytes
	// got (640,480)
	// read 460800 bytes
	// got (640,480)
	// read 460800 bytes
	// got (640,480)
	// read 460800 bytes
}
```

D.4.5 picam/format_string.go

```go
// Code generated by "stringer -type=Format"; DO NOT EDIT.

package picam

import "strconv"

func _() {
	// An "invalid array index" compiler error signifies that the constant values have changed.
	// Re-run the stringer command to generate them again.
	var x [1]struct{}
	_ = x[YUV-0]
```

```go
		_ = x[RGB-1]
		_ = x[Gray-2]
}

const _Format_name = "YUVRGBGray"

var _Format_index = [...]uint8{0, 3, 6, 10}

func (i Format) String() string {
		if i >= Format(len(_Format_index)-1) {
				return "Format(" + strconv.FormatInt(int64(i), 10) + ")"
		}
		return _Format_name[_Format_index[i]:_Format_index[i+1]]
}
```

D.4.6 *picam/info.go*

```go
// Package picam is a Go wrapper to `raspiyuv` to get `[]uint8` and `image.Image` data of
// the latests frame captured by the Raspberry Pi camera.
//
// Under the hood, it executes:
//		$ raspiyuv --timeout 0 --timelapse 0
// to get raw frames.
//
// Currently, three image formats are available:
//		* picam.YUV
//		* picam.RGB
//		* picam.Gray
//
// The time between frames, measured on a Raspberry Pi Zero W, is between `180ms` to
// `210ms` for a `640x480` pixels image.
//
// If you want to test the speed in your system, run:
//		$ cd $(go env GOPATH)/src/github.com/cgxeiji/picam
//		$ go test -bench . -benchtime=10x
//
// This will take 10 frames and output the average time between each frame. Change
// `-benchtime=10x` to `100x` or `Nx` to change the number of frames to test.
package picam
```

D.4.7 *picam/picam.go*

```go
package picam

import (
		"bufio"
		"fmt"
		"image"
		"io"
		"math"
		"os/exec"
		"strconv"
		"sync"
)

// Camera is a struct that stores camera information.
type Camera struct {
		cmd *exec.Cmd
		// Width sets the width of the image
		// Height sets the height of the image
		Width, Height int
		frame		<-chan []uint8
		format		Format
		done		chan struct{}
		ws		*sync.WaitGroup
}

// Format is the type of image that picam will output.
type Format uint8

//go:generate stringer -type=Format
const (
		// YUV 420 color format.
		YUV Format = iota
		// RGB color format.
		RGB
		// Gray color format.
		Gray
)

// New initializes and starts a raspiyuv process to capture RGB frames.
func New(width, height int, format Format) (*Camera, error) {
```

```go
	args := []string{
		"--burst",
		"--width", strconv.Itoa(width),
		"--height", strconv.Itoa(height),
		"--timeout", "0",
		"--timelapse", "0",
		"--nopreview",
	}

	var img []uint8
	switch format {
	case RGB:
		args = append(args, "--rgb")
		img = make([]uint8, width*height*3)
	case Gray:
		args = append(args, "--luma")
		img = make([]uint8, width*height)
	default:
		w, h := roundUp(width, 32), roundUp(height, 16)
		img = make([]uint8, w*h+w*h/2)
	}

	args = append(args, []string{"--output", "-"}...)

	cmd := exec.Command("raspiyuv", args...)

	out, err := cmd.StdoutPipe()
	if err != nil {
		return nil, fmt.Errorf("picam: cannot create out pipe: %w", err)
	}

	err = cmd.Start()
	if err != nil {
		return nil, fmt.Errorf("picam: unable to start picam: %w", err)
	}

	frame := make(chan []uint8)
	done := make(chan struct{})
	var ws sync.WaitGroup
	ws.Add(1)
	go func() {
		defer ws.Done()
		defer close(frame)
		defer cmd.Process.Kill()
		r := bufio.NewReader(out)
		for {
			_, _ = io.ReadFull(r, img)
			select {
			case <-done:
				return
			case frame <- img:
			default:
			}
		}
	}()

	return &Camera{
		Width:  width,
		Height: height,
		cmd:    cmd,
		frame:  frame,
		format: format,
		done:   done,
		ws:     &ws,
	}, nil
}

// Close closes picam.
func (c *Camera) Close() {
	close(c.done)
	c.ws.Wait()
}

// Read returns an image.Image interface of the last frame.
//
//	cam, _ := picam.New(width, height, format)
//	img := cam.Read()
//
// The type returned depends on the format passed at picam.New():
//
//	format         type(img)
//	----------     ---------------
//	picam.YUV  -> image.YCbCr 420
//	picam.RGB  -> image.NRGBA
//	picam.Gray -> image.Gray
func (c *Camera) Read() (img image.Image) {
	size := image.Rect(0, 0, c.Width, c.Height)
	switch c.format {
	case RGB:
		rgba := image.NewNRGBA(size)
		pixels := make([]uint8, c.Width*c.Height*4)
		rgb := <-c.frame
		for i, idx := 0, 0; i < len(rgb); i++ {
```

```go
						pixels[idx] = rgb[i]
						idx++
						if i%3 == 2 {
							pixels[idx] = 255
							idx++
						}
				}
				rgba.Pix = pixels
				img = rgba
		case Gray:
				gray := image.NewGray(size)
				gray.Pix = <-c.frame
				img = gray
		default:
				yuv := image.NewYCbCr(size, image.YCbCrSubsampleRatio420)

				yRange := roundUp(c.Width, 32) * roundUp(c.Height, 16)
				uvRange := yRange / 4

				frame := <-c.frame
				yuv.Y = frame[0:yRange]
				yuv.Cb = frame[yRange : uvRange+yRange]
				yuv.Cr = frame[yRange+uvRange : uvRange*2+yRange]
				img = yuv
		}

		return img
}

func roundUp(value, multiple int) int {
		return int(math.Ceil(float64(value)/float64(multiple))) * multiple
}

// ReadUint8 returns the raw uint8 values of the last frame.
//
//		cam, _ := picam.New(width, height, format)
//		raw := cam.ReadUint8()
//
// The size of the slice returned depends on the format and dimensions passed at picam.New():
//
//		format		len(raw)
//		----------	---------------------------------------------------------
//		picam.YUV  -> roundUpMultiple32(width) * roundUpMultiple16(height) * 1.5
//		picam.RGB  -> (width * height) * 3
//		picam.Gray -> width * height
func (c *Camera) ReadUint8() (img []uint8) {
		return <-c.frame
}
```

D.4.8 *picam/picam_test.go*

```go
package picam

import (
		"fmt"
		"image"
		"image/color"
		"testing"
)

func TestNew(t *testing.T) {
		cam, err := New(640, 480, YUV)
		if err != nil {
				t.Fatal(err)
		}
		defer cam.Close()

}

func TestRead(t *testing.T) {
		c := color.RGBA{}
		tests := []struct {
				format Format
				want   color.Color
		}{
				{YUV, color.YCbCrModel.Convert(c)},
				{RGB, color.NRGBAModel.Convert(c)},
				{Gray, color.GrayModel.Convert(c)},
		}

		for _, ts := range tests {
				t.Run(fmt.Sprintf("%s", ts.format), func(t *testing.T) {
						cam, err := New(640, 480, ts.format)
						if err != nil {
								t.Fatal(err)
						}
						defer cam.Close()
```

```go

			img := cam.Read()

			got := img.ColorModel().Convert(c)
			if got != ts.want {
				t.Errorf("got: %T, want: %T", got, ts.want)
			}
		})
	}
}

func TestReadUint8(t *testing.T) {
	w, h := 640, 480
	tests := []struct {
		format Format
		want   int // byte size
	}{
		{YUV, w*h + w*h/2},
		{RGB, w * h * 3},
		{Gray, w * h},
	}

	for _, ts := range tests {
		t.Run(fmt.Sprintf("%s", ts.format), func(t *testing.T) {
			cam, err := New(640, 480, ts.format)
			if err != nil {
				t.Fatal(err)
			}
			defer cam.Close()

			img := cam.ReadUint8()

			got := len(img)
			if got != ts.want {
				t.Errorf("got: %d, want: %d", got, ts.want)
			}
		})
	}
}

func TestReadUint8_Sizes(t *testing.T) {
	tests := []struct {
		format        Format
		width, height int
		want          int // byte size
	}{
		{
			YUV,
			320, 240,
			320*240 + 320*240/2,
		},
		{
			YUV,
			100, 100,
			128*112 + 128*112/2,
		},
		{
			RGB,
			320, 240,
			320 * 240 * 3,
		},
		{
			Gray,
			320, 240,
			320 * 240,
		},
	}

	for _, ts := range tests {
		t.Run(fmt.Sprintf("%s (%d,%d)", ts.format, ts.width, ts.height), func(t *testing.T) {
			cam, err := New(ts.width, ts.height, ts.format)
			if err != nil {
				t.Fatal(err)
			}
			defer cam.Close()

			raw := cam.ReadUint8()

			got := len(raw)
			if got != ts.want {
				t.Errorf("got: %d, want: %d", got, ts.want)
			}

			img := cam.Read()
			gotP := img.Bounds().Size()
			wantP := image.Point{ts.width, ts.height}
			if gotP != wantP {
				t.Errorf("got: %+v, want: %+v", gotP, wantP)
			}
		})
	}
}
```

D.5 Module: anim

D.5.1 License

D.5.2 *anim/anim.go*

```go
package anim

import (
	"fmt"
	"io/ioutil"
	"sync"

	"module/body"
	"module/eye"

	"gopkg.in/yaml.v2"
)

type data struct {
	loaded   bool
	Playlist []string               `yaml:"playlist"`
	Eye      map[string][]eyeFrame  `yaml:"eye.data,flow"`
	Body     map[string][]bodyFrame `yaml:"body.data,flow"`
	ws       *sync.WaitGroup
}

// Internal variable to load the animations.
var animation data

// Eye represents a global Eye object.
var Eye *eye.Eye

// Body represents a global Eye object.
var Body *body.Body

// ReadFile loads the animation file to be played and returns a string set of
// animations.
func ReadFile(filename string) ([]string, error) {
	d, err := ioutil.ReadFile(filename)
	if err != nil {
		return nil, fmt.Errorf("anim: could not read file %s: %w", filename, err)
	}
	if err := yaml.Unmarshal(d, &animation); err != nil {
		return nil, fmt.Errorf("anim: could not unmarshal yaml file %s: %w", filename, err)
	}

	var ws sync.WaitGroup
	animation.ws = &ws

	animation.loaded = true

	return animation.Playlist, nil
}

// Play plays a named animation. If the name of the animation is not found in
// the animation playlist, it returns an ErrAnimationNotFound error.
func Play(name string) (wait Waiter, err error) {
```

```
        return animation.play(name)
}

// Waiter implements the Wait function.
type Waiter interface {
        // Wait waits for the current animation to complete.
        Wait()
}

// Wait implements the Waiter interface.
func (a *data) Wait() {
        a.ws.Wait()
}

type syncer func()

func (a *data) play(name string) (wait Waiter, err error) {
        if !a.loaded {
                return nil, fmt.Errorf("anim: no animation file was loaded")
        }

        ef, ok := a.Eye[name]
        if !ok {
                return nil, fmt.Errorf("anim: eye animation \"%s\": %w", name, ErrAnimationNotFound)
        }

        bf, ok := a.Body[name]
        if !ok {
                return nil, fmt.Errorf("anim: body animation \"%s\": %w", name, ErrAnimationNotFound)
        }

        sCh := make(chan struct{}, 1)
        resume := make(chan struct{})
        var s syncer = func() {
                select {
                case sCh <- struct{}{}:
                        <-resume
                default:
                        // fmt.Println("syncing")
                        for i := 0; i < cap(sCh); i++ {
                                <-sCh
                                resume <- struct{}{}
                        }
                        // fmt.Println("done")
                }
        }

        a.ws.Add(1)
        go func(f []eyeFrame) {
                defer a.ws.Done()
                for i := range f {
                        f[i].play(s)
                }
        }(ef)

        a.ws.Add(1)
        go func(f []bodyFrame) {
                defer a.ws.Done()
                for i := range f {
                        f[i].play(s)
                }
        }(bf)

        // fmt.Printf("ef = %+v\n", ef)
        // fmt.Printf("bf = %+v\n", bf)

        return a, nil
}
```

D.5.3 *anim/body.go*

```
package anim

import (
        "fmt"
        "strconv"
        "strings"
        "sync"
        "time"

        "module/body"
)

type bodyFrame struct {
        Body string
}

```

```go
const (
	idxX int = iota
	idxY
	idxBT
	idxSpeed
)

func (b *bodyFrame) play(s syncer) {
	if Body == nil {
		return
	}

	// fmt.Printf("b.Body = %+v\n", b.Body)

	if strings.HasPrefix(b.Body, "sync") {
		s()
	}

	f := strings.TrimPrefix(b.Body, "sync")

	args := strings.Split(f, ",")
	for i := range args {
		args[i] = strings.TrimSpace(args[i])
	}

	x, err := strconv.ParseFloat(args[idxX], 64)
	if err != nil {
		fmt.Printf("err = %+v\n", err)
		return
	}
	y, err := strconv.ParseFloat(args[idxY], 64)
	if err != nil {
		fmt.Printf("err = %+v\n", err)
		return
	}
	t, err := strconv.ParseInt(args[idxBT], 10, 0)
	if err != nil {
		fmt.Printf("err = %+v\n", err)
		return
	}
	speed, err := strconv.ParseFloat(args[idxSpeed], 64)
	if err != nil {
		fmt.Printf("err = %+v\n", err)
		return
	}

	var ws sync.WaitGroup

	ws.Add(1)
	go func() {
		defer ws.Done()
		Body.SetSpeed(speed)
		Body.Move(x, y, body.MoveAbsolute).Wait()
		time.Sleep(time.Duration(t) * time.Millisecond)
	}()

	ws.Wait()

	return
}
```

D.5.4 *anim/errors.go*

```go
package anim

import "fmt"

// Errors
var (
	ErrAnimationNotFound = fmt.Errorf("animation not found")
)
```

D.5.5 *anim/eye.go*

```go
package anim

import (
	"fmt"
	"image/color"
	"math"
	"strconv"
	"strings"
```

```go
		"sync"
		"time"
)

type eyeFrame struct {
	Eye      string
	Rotation string
	Shine    string
}

const (
	idxTop int = iota
	idxBottom
	idxDelay
	idxAlpha
)

const (
	idxRot int = iota
	idxRotDelay
)

func (e *eyeFrame) play(s syncer) {
	if Eye == nil {
		return
	}

	// fmt.Printf("e.Eye = %+v\n", e.Eye)

	if strings.HasPrefix(e.Eye, "sync") {
		s()
	}

	f := strings.TrimPrefix(e.Eye, "sync")

	args := strings.Split(f, ",")
	for i := range args {
		args[i] = strings.TrimSpace(args[i])
	}

	top, err := strconv.ParseFloat(args[idxTop], 64)
	if err != nil {
		fmt.Printf("err = %+v\n", err)
		return
	}
	bottom, err := strconv.ParseFloat(args[idxBottom], 64)
	if err != nil {
		fmt.Printf("err = %+v\n", err)
		return
	}
	t, err := strconv.ParseInt(args[idxDelay], 10, 0)
	if err != nil {
		fmt.Printf("err = %+v\n", err)
		return
	}
	alpha, err := strconv.ParseFloat(args[idxAlpha], 64)
	if err != nil {
		fmt.Printf("err = %+v\n", err)
		return
	}

	var ws sync.WaitGroup

	Eye.Color(color.CMYK{255, 0, 0, uint8((1-alpha)*245 + 10)})

	ws.Add(1)
	go func() {
		defer ws.Done()
		Eye.Eyelids(top, bottom, time.Duration(t)*time.Millisecond)
	}()

	if e.Rotation != "" {
		fmt.Printf("e.Rotation = %+v\n", e.Rotation)
		args = strings.Split(e.Rotation, ",")
		for i := range args {
			args[i] = strings.TrimSpace(args[i])
		}
		rot, err := strconv.ParseFloat(args[idxRot], 64)
		if err != nil {
			fmt.Printf("err = %+v\n", err)
			return
		}
		rot = rot/180*math.Pi + Body.Angle() - 0.72
		var rotT int64
		if len(args) > 1 {
			rotT, err = strconv.ParseInt(args[idxRotDelay], 10, 0)
			if err != nil {
				fmt.Printf("err = %+v\n", err)
				return
			}
		}
		ws.Add(1)
		go func() {
```

```
                        defer ws.Done()
                        Eye.LerpOffset(rot, time.Duration(rotT)*time.Millisecond)
                }()
        }

        ws.Wait()

        return
}
```

D.5.6 `anim/anim.yaml`

```yaml
# For now, the animation being played is 'test'.
# To add a frame, add a line with the following information:
#   anim:
#     - eye: top, bot, time(ms), alpha
#       shine: angle, alpha <- this is optional
#       rotation: angle(degrees) <- this is optional
#
#     - body: x, y, hold_time(ms), speed
#
# top, bot, alpha, and speed have values from 0.0 to 1.0
# x and y have values from -1.0 to 1.0
#
# Put the keyword 'sync' after eye and body to synchronize those two frames.
#   eye.data:
#     anim:
#       - eye: sync top, bot, time(ms), alpha
#   body.data:
#     anim:
#       - body: sync x, y, hold_time(ms), speed
#
#
# You can choose which animations will be played using the playlist:
#   playlist:
#     - anim1
#     - anim2
#     - anim3
#
# The animations can be repeated. The robot will return to the default
# animation after playing all the animations. There is a 2 seconds interval
# between each animation.

playlist:
  - think
  - anim1
  - anim2
  - anim3
  - anim4

eye.data:
  think:
    - eye: sync 1.0, 0.0, 500, 1.0
    - eye: sync 1.0, 0.0, 500, 1.0
      # shake
    - eye: sync 0.7, 0.2, 200, 1.0
    - eye: 0.8, 0.1, 200, 1.0
    - eye: 0.7, 0.2, 200, 1.0
    - eye: sync 0.8, 0.1, 200, 1.0
      # return
    - eye: sync 1.0, 0.0, 500, 1.0
  anim1:
    - eye: sync 0.5, 0.0, 500, 1.0
    - eye: sync 1.0, 0.0, 500, 1.0
  anim1:
    - eye: sync 0.9, 0.1, 500, 1.0
    - eye: sync 1.0, 0.0, 500, 1.0
  anim1:
    - eye: sync 1.0, 0.3, 500, 1.0
    - eye: sync 1.0, 0.0, 500, 1.0
  anim1:
    - eye: sync 1.0, 0.0, 500, 1.0
    - eye: 0.3, 0.3, 100, 1.0
    - eye: 1.0, 0.0, 200, 1.0
    - eye: sync 1.0, 0.0, 500, 1.0

body.data:
```

```
  think:
    - body: sync 0, 0.5, 0, 1.0
    - body: sync 0, 0.5, 0, 1.0
      # shake
    - body: sync 0, 0.6, 0, 1.0
    - body: 0, 0.5, 0, 1.0
    - body: 0, 0.6, 0, 1.0
    - body: sync 0, 0.5, 0, 1.0
      # return
    - body: sync 0, 0, 0, 1.0
  anim1:
    - body: sync 0.1, 0.1, 0, 1.0
    - body: -0.1, -0.1, 0, 1.0
    - body: 0.1, 0.1, 0, 1.0
    - body: -0.1, -0.1, 0, 1.0
    - body: sync 0, 0, 0, 1.0
  anim2:
    - body: sync 0.2, 0.0, 200, 1.0
    - body: -0.2, 0.0, 200, 1.0
    - body: 0.2, 0.0, 200, 1.0
    - body: sync 0, 0, 0, 1.0
  anim3:
    - body: sync 0, 0.5, 0, 1.0
    - body: 0.1, 0.6, 0, 1.0
    - body: 0.0, 0.5, 0, 1.0
    - body: 0.1, 0.6, 0, 1.0
    - body: 0.0, 0.5, 0, 1.0
    - body: sync 0, 0, 0, 1.0
  anim4:
    - body: sync 0, 0.2, 0, 1.0
    - body: 0.1, 0.2, 0, 1.0
    - body: 0.0, 0.2, 0, 1.0
    - body: sync 0, 0, 0, 1.0
```

D.6 Module: tracker

D.6.1 License

D.6.2 *tracker/tracker.go*

```go
package tracker

import (
	"fmt"
	"image"
	"io/ioutil"
	"math"
	"sync"

	"github.com/cgxeiji/picam"
	pigo "github.com/esimov/pigo/core"
)

// Face defines a face tracker.
type Face struct {
	classifier *pigo.Pigo
	camera     *picam.Camera
	cParams    pigo.CascadeParams
	track      chan float64
	face       chan pigo.Detection
	img        chan []uint8

	scale float64

	done chan struct{}
	ws   *sync.WaitGroup
}

// New creates a new face tracker.
func New(width, height int) (*Face, error) {
	file, err := ioutil.ReadFile("/home/pi/sandbox/facefinder")
	if err != nil {
		return nil, fmt.Errorf("tracker: could not load classifier: %w", err)
	}

	classifier, err := pigo.NewPigo().Unpack(file)
	if err != nil {
		return nil, fmt.Errorf("tracker: could not unpack classifier: %w", err)
	}

	camera, err := picam.New(width, height, picam.Gray)
	if err != nil {
		return nil, fmt.Errorf("tracker: could not load camera: %w", err)
	}

	cParams := pigo.CascadeParams{
		MinSize:     90,
		MaxSize:     200,
		ShiftFactor: 0.1,
		ScaleFactor: 1.1,
		ImageParams: pigo.ImageParams{
			Rows: camera.Height,
```

```
			Cols: camera.Width,
			Dim:  camera.Width,
		},
	}

	done := make(chan struct{})
	var ws sync.WaitGroup

	t := &Face{
		classifier: classifier,
		camera:     camera,
		cParams:    cParams,
		track:      make(chan float64),
		face:       make(chan pigo.Detection),
		img:        make(chan []uint8, 1),

		scale: math.Min(float64(camera.Height), float64(camera.Width)),

		done: done,
		ws:   &ws,
	}

	ws.Add(1)
	go func() {
		defer ws.Done()
		for {
			select {
			case <-done:
				close(t.face)
				return
			case angle := <-t.track:
				t.cParams.Pixels = t.camera.ReadUint8()
				select {
				case t.img <- t.cParams.Pixels:
				default:
				}
				faces := t.classifier.ClusterDetections(
					t.classifier.RunCascade(t.cParams, angle), 0.1)

				idx := 0
				for i := range faces {
					if faces[i].Q > faces[idx].Q {
						idx = i
					}
				}

				if idx < len(faces) {
					t.face <- faces[idx]
				} else {
					t.face <- pigo.Detection{}
				}
			}
		}
	}()

	return t, nil
}

// Close stops the tracker and cleans after itself.
func (f *Face) Close() {
	close(f.done)
	f.ws.Wait()
	f.camera.Close()
}

// Detect returns the normalized position of the face from the center if found,
// given a detection angle in radians.
func (f *Face) Detect(angle float64) (x, y float64, found bool) {
	f.track <- angle / (math.Pi * 2)
	face := <-f.face

	if face.Scale == 0 {
		return 0, 0, false
	}

	fx := float64(face.Col)/float64(f.cParams.Cols)*2 - 1
	fy := 1 - float64(face.Row)/float64(f.cParams.Rows)*2

	angle *= -1
	s, c := math.Sin(angle), math.Cos(angle)

	x = s*fy - c*fx
	y = s*fx + c*fy

	return x, y, true
}

// Image returns a uint8 vector with the latest processed image.
func (f *Face) Image() *image.Gray {
	img := <-f.img
	i := make([]uint8, len(img))
	copy(i, img)
	gray := image.NewGray(image.Rect(0, 0, f.cParams.Cols, f.cParams.Rows))
```

```
        gray.Pix = i

        return gray
}
```

D.7 Module: bracelet

D.7.1 License

D.7.2 *bracelet/console.go*

```go
package main

import (
	"image"
	"strings"

	"golang.org/x/image/font"
	"golang.org/x/image/font/inconsolata"
	"golang.org/x/image/math/fixed"
	"periph.io/x/periph/conn/i2c"
	"periph.io/x/periph/conn/i2c/i2creg"
	"periph.io/x/periph/devices/ssd1306"
	"periph.io/x/periph/devices/ssd1306/image1bit"
	"periph.io/x/periph/host"
)

type console struct {
	dev *ssd1306.Dev
	bus i2c.BusCloser

	buffer []string
	drawer font.Drawer
	cursor int
	bounds image.Rectangle
}

func newConsole() (*console, error) {
	f := inconsolata.Regular8x16

	if _, err := host.Init(); err != nil {
		return nil, err
	}

	bus, err := i2creg.Open("")
	if err != nil {
		return nil, err
	}

	dev, err := ssd1306.NewI2C(bus, &ssd1306.DefaultOpts)
	if err != nil {
		return nil, err
	}

	drawer := font.Drawer{
		Src:  &image.Uniform{image1bit.On},
		Face: f,
		Dot:  fixed.P(0, 0),
	}

	m := drawer.Face.Metrics()
	h := m.Height.Round()
	lines := dev.Bounds().Dy() / h
```

```go

	return &console{
		dev: dev,
		bus: bus,

		buffer: make([]string, lines),
		drawer: drawer,
		bounds: dev.Bounds(),
	}, nil
}

func (c *console) close() {
	c.bus.Close()
}

func (c *console) println(s string) {
	for i := 0; i < len(c.buffer)-1; i++ {
		c.buffer[i] = c.buffer[i+1]
	}

	c.buffer[len(c.buffer)-1] = s
}

func (c *console) clear() {
	for i := 0; i < len(c.buffer); i++ {
		c.buffer[i] = ""
	}
}

func (c *console) render() error {
	c.drawer.Dst = image1bit.NewVerticalLSB(c.bounds)

	m := c.drawer.Face.Metrics()

	for i, s := range c.buffer {
		c.drawer.Dot = fixed.P(0, m.Height.Round()*i+m.Ascent.Round()-4)
		c.drawer.DrawString(s)
	}

	return c.dev.Draw(c.bounds, c.drawer.Dst, image.Point{})
}

func (c *console) Write(p []byte) (n int, err error) {
	var b strings.Builder

	b.Write(p)
	strs := strings.Split(b.String(), "\n")

	for _, s := range strs {
		c.println(s)
	}

	return len(p), nil
}
```

D.7.3 bracelet/main.go

```go
package main

import (
	"fmt"
	"log"
	"os"
	"sandbox/ads1x15"
	"sync"
	"time"

	"github.com/cgxeiji/lsm6"
	"github.com/cgxeiji/max3010x"
)

func main() {
	fmt.Println("loading")
	fn := time.Now().Format("20060102150405")
	_, err := os.Stat(fmt.Sprintf("./logs/%s.log", fn))

	for err == nil {
		fn += "a"
		_, err = os.Stat(fmt.Sprintf("./logs/%s.log", fn))
	}

	f, err := os.OpenFile(fmt.Sprintf("./logs/%s.log", fn), os.O_RDWR|os.O_CREATE|os.O_APPEND, 0666)
	if err != nil {
		log.Fatalf("error opening log file: %v", err)
	}
	defer f.Close()
	log.SetFlags(log.LstdFlags | log.Lmicroseconds)
```

```go
	log.SetOutput(f)

	log.Println("Starting device")

	done := make(chan struct{})
	var wg sync.WaitGroup

	console, err := newConsole()
	if err != nil {
		log.Fatal(err)
	}

	renderCh := make(chan struct{}, 1)

	type dateS struct {
		date string
		time string
	}

	timeCh := make(chan dateS)

	type ag struct {
		ax float64
		ay float64
		az float64

		gx float64
		gy float64
		gz float64
	}

	wg.Add(1)
	go func() {
		defer wg.Done()
		t := time.NewTicker(1 * time.Second)
		for {
			select {
			case <-done:
				return
			case <-t.C:
			}

			date := time.Now().Format("Jan-02")
			hour := time.Now().Format("15:04:05")

			select {
			case <-done:
				return
			case timeCh <- dateS{
				date: date,
				time: hour,
			}:
			}
		}
	}()

	wg.Add(1)
	go func() {
		defer wg.Done()
		t := time.NewTicker(4 * time.Millisecond)
		for {
			select {
			case <-done:
				return
			case <-t.C:
			}

			select {
			case <-done:
				return
			case renderCh <- struct{}{}:
			}
		}
	}()

	hrCh := hrInit()
	imuCh := imuInit()
	adsCh := adsInit()

	wg.Add(2)
	go func() {
		defer wg.Done()
		date := dateS{}
		hr := hrch{}
		imu := imuch{}
		ads := adsch{}

		render := make(chan struct{})
		go func() {
			defer wg.Done()
			for {
				select {
				case <-done:
```

```
					return
				case <-render:
				}
				//fmt.Fprintf(console, "     %s", date.date)
				fmt.Fprintf(console, "    %s", date.time)
				fmt.Fprintf(console, "  %.2f s: %d", imu.ax, ads.ads)
				//fmt.Fprintf(console, "     LEELAB")
				fmt.Fprintf(console, "  h: %.2f", hr.red)
				//fmt.Fprintf(console, "      OSAD")
				fmt.Fprintf(console, "")

				if err := console.render(); err != nil {
					log.Fatal(err)
				}
			}
		}()

		for {
			select {
			case <-done:
				return
			case <-renderCh:
			}

			select {
			case date = <-timeCh:
			default:
			}
			select {
			case hr = <-hrCh:
			default:
			}
			select {
			case date = <-timeCh:
			default:
			}
			select {
			case imu = <-imuCh:
			default:
			}
			select {
			case ads = <-adsCh:
			default:
			}

			if hr.valid {
				log.Printf("[HR]: %.8f %.8f\n", hr.red, hr.ir)
			}

			if imu.valid {
				log.Printf("[IMU]: %f %f %f %f %f %f\n", imu.ax, imu.ay, imu.az, imu.gx, imu.gy, imu.gz)
			}

			if ads.valid {
				log.Printf("[ADS]: %v\n", ads.ads)
			}

			select {
			case render <- struct{}{}:
			default:
			}

			//time.Sleep(100 * time.Millisecond)
		}
	}()

	select {
	case <-done:
	}
	//fmt.Println("Press [ENTER] to exit.")
	//bufio.NewReader(os.Stdin).ReadString('\n')
	//close(done)
	//wg.Wait()
}

type hrch struct {
	red   float64
	ir    float64
	valid bool
}

func hrInit() <-chan hrch {
	//done := make(chan struct{})
	hrCh := make(chan hrch)

	device, err := max3010x.New()
	fmt.Println("hr loaded")

	go func() {
		defer device.Close()
		toSend := hrch{}
		for {
			hrCh <- toSend
```

```
				if device != nil && err == nil {
					sensor, err := device.ToMax30102()
					if err != nil {
						toSend = hrch{}
						continue
					}
					ir, red, err := sensor.IRRed()
					if err != nil {
						toSend = hrch{}
						continue
					}
					toSend = hrch{
						red:   red,
						ir:    ir,
						valid: true,
					}
				} else {
					time.Sleep(1 * time.Second)
					device, err = max3010x.New()
					toSend = hrch{}
				}
			}
	}()

	return hrCh
}

type imuch struct {
	ax    float64
	ay    float64
	az    float64
	gx    float64
	gy    float64
	gz    float64
	valid bool
}

func imuInit() <-chan imuch {
	imuCh := make(chan imuch)

	sensor, err := lsm6.New("", 0)
	fmt.Println("imu loaded")

	go func() {
		defer sensor.Close()
		imuCh <- imuch{}
		for {
			if sensor != nil && err == nil {
				ax, ay, az, err := sensor.Acc()
				if err != nil {
					imuCh <- imuch{}
					log.Println(err)
					sensor = nil
					continue
				}
				gx, gy, gz, err := sensor.Gyro()
				if err != nil {
					imuCh <- imuch{}
					log.Println(err)
					sensor = nil
					continue
				}
				imuCh <- imuch{
					ax:    ax,
					ay:    ay,
					az:    az,
					gx:    gx,
					gy:    gy,
					gz:    gz,
					valid: true,
				}
			} else {
				sensor, err = lsm6.New("", 0)
			}
		}
	}()

	return imuCh
}

type adsch struct {
	ads   int
	valid bool
}

func adsInit() <-chan adsch {
	adsCh := make(chan adsch)

	sensor, err := ads1x15.New("", ads1x15.AddrGND)
	fmt.Println("skin loaded")

	go func() {
		defer sensor.Close()
```

```
                adsCh <- adsch{}
                for {
                        if sensor != nil && err == nil {
                                v, err := sensor.ReadSingle(0)
                                if err != nil {
                                        adsCh <- adsch{}
                                        log.Println(err)
                                        sensor = nil
                                        continue
                                }
                                adsCh <- adsch{
                                        ads:   v,
                                        valid: true,
                                }
                        } else {
                                sensor, err = ads1x15.New("", ads1x15.AddrGND)
                        }
                }
        }()

        return adsCh
}
```

D.8 Module: ads1x15

D.8.1 `License`

D.8.2 *`ads1x15/ads1x15.go`*

```go
package ads1x15

import (
	"fmt"

	"github.com/cgxeiji/serial"
)

// Device defines a ADS1x15 device.
type Device struct {
	i2c *serial.I2C
}

// New returns a new ADS1x15 device.
func New(busName string, addr uint16) (*Device, error) {
	if addr == 0 {
		addr = Addr
	}

	i2c, err := serial.NewI2C(busName, addr)
	if err != nil {
		return nil, fmt.Errorf("ads1x15: could not initialize I2C: %w", err)
	}

	d := &Device{
		i2c: i2c,
	}

	// default options
	d.Options(
		DisableComparator(),
		NonLatch(),
		LowPolarity(),
		TraditionalMode(),
		SingleShot(),
		Rate(SPS1600),
		Gain(FS6v144),
	)

	return d, nil
}

// Close closes the device and cleans after itself.
func (d *Device) Close() {
	d.i2c.Close()
}

// ReadSingle returns the single ended result of the ADC on the requested
// channel.
func (d *Device) ReadSingle(channel int) (int, error) {
	if channel > 3 || channel < 0 {
		return 0, fmt.Errorf("ads1x15: channel outside the allowed range: (want: [0-3], got: %d)", channel)
```

```go
	}

	var err error
	switch channel {
	case 0:
		_, err = d.Options(Mux(Mux0))
	case 1:
		_, err = d.Options(Mux(Mux1))
	case 2:
		_, err = d.Options(Mux(Mux2))
	case 3:
		_, err = d.Options(Mux(Mux3))
	}
	if err != nil {
		return 0, fmt.Errorf("ads1x15: could not select channel: %w", err)
	}

	if err := d.singleConv(); err != nil {
		return 0, fmt.Errorf("ads1x15: could not read channel %d: %w", channel, err)
	}

	if err := d.waitUntil(CfgAddr, OSMask, 1); err != nil {
		return 0, fmt.Errorf("ads1x15: could not read channel %d: %w", channel, err)
	}

	value, err := d.i2c.ReadBytes(ConvAddr, 2)
	if err != nil {
		return 0, fmt.Errorf("ads1x15: could not read channel %d: %w", channel, err)
	}

	v := (uint16(value[0])<<8 | uint16(value[1])) >> 4
	if v > 0x07FF {
		v |= 0xF000
	}

	return int(v), nil
}

// GetChannel returns a Channel struct attached to a specific multiplexer setting.
func (d *Device) GetChannel(channel MuxSetting) *Channel {
	return &Channel{
		d:   d,
		mux: channel,
	}
}

func (d *Device) singleConv() error {
	_, err := d.config(CfgAddr, OSMask, SingleConv)
	return err
}

func (d *Device) waitUntil(reg byte, flag uint16, bit byte) error {
	switch bit {
	case 1:
		for {
			state, err := d.i2c.ReadBytes(reg, 2)
			s := uint16(state[0])<<8 | uint16(state[1])
			if err != nil {
				return fmt.Errorf("could not wait for %#b in %#b to be %v", flag, reg, bit)
			} else if s&flag != 0 {
				return nil
			}
		}
	case 0:
		for {
			state, err := d.i2c.ReadBytes(reg, 2)
			s := uint16(state[0])<<8 | uint16(state[1])
			if err != nil {
				return fmt.Errorf("could not wait for %#b in %#b to be %v", flag, reg, bit)
			} else if s&flag == 0 {
				return nil
			}
		}
	}

	return fmt.Errorf("invalid bit %v, it should be 1 or 0", bit)
}

// Read reads a single byte from a register.
func (d *Device) Read(reg byte) (byte, error) {
	return d.i2c.Read(reg)
}

// ReadBytes reads n bytes from a register.
func (d *Device) ReadBytes(reg byte, n int) ([]byte, error) {
	return d.i2c.ReadBytes(reg, n)
}

// Write writes a byte or bytes to a register.
func (d *Device) Write(reg byte, data ...byte) error {
	return d.i2c.Write(reg, data...)
}
```

D.8.3 *ads1x15/channel.go*

```go
package ads1x15

import "fmt"

// Channel returns a Channel struct attached to a specific multiplexer setting.
type Channel struct {
	d   *Device
	mux MuxSetting
}

func (c *Channel) Read() (int, error) {
	old, err := c.d.Options(Mux(c.mux))
	if err != nil {
		return 0, fmt.Errorf("channel %v: could not select channel: %w", c.mux, err)
	}

	if err := c.d.singleConv(); err != nil {
		return 0, fmt.Errorf("channel %v: could not read channel: %w", c.mux, err)
	}
	if err := c.d.waitUntil(CfgAddr, OSMask, 1); err != nil {
		return 0, fmt.Errorf("channel %v: could not wait for channel: %w", c.mux, err)
	}

	value, err := c.d.i2c.ReadBytes(ConvAddr, 2)
	if err != nil {
		return 0, fmt.Errorf("channel %v: could not get value: %w", c.mux, err)
	}

	v := (uint16(value[0])<<8 | uint16(value[1])) >> 4
	if v > 0x07FF {
		v |= 0xF000
	}

	_, err = c.d.Options(old)
	if err != nil {
		return 0, fmt.Errorf("channel %v: could not return to previous configuration: %w", c.mux, err)
	}

	return int(v), nil
}
```

D.8.4 *ads1x15/const.go*

```go
package ads1x15

// Register addresses
const (
	ConvAddr = 0b00
	CfgAddr  = 0b01
	LoThresh = 0b10
	HiThresh = 0b11
)

// Operational Status
const (
	SingleConv uint16 = (1 << 15)
	OSMask     uint16 = (1 << 15)
)

// MuxSetting defines a multiplexer settings type.
type MuxSetting uint16

// Input Multiplexer
const (
	Mux0_1 MuxSetting = (iota << 12)
	Mux0_3
	Mux1_3
	Mux2_3
	Mux0
	Mux1
	Mux2
	Mux3

	MuxMask uint16 = (0b111 << 12)
)

// Programmable Gain Amplifier
const (
	FS6v144 uint16 = (iota << 9)
	FS4v096
	FS2v048
	FS1v024
	FS0v512
```

```
        FS0v256

        PGAMask uint16 = (0b111 << 9)
)

// Mode
const (
        ModeContinuous uint16 = (iota << 8)
        ModeSingleShot

        ModeMask uint16 = (1 << 8)
)

// Data Rate
const (
        SPS128 uint16 = (iota << 5)
        SPS250
        SPS490
        SPS920
        SPS1600
        SPS2400
        SPS3300

        DRMask uint16 = (0b111 << 5)
)

// Comparator Mode
const (
        CompTrad uint16 = (iota << 4)
        CompWindow

        CompModeMask uint16 = (1 << 4)
)

// Comparator Polarity
const (
        ActiveLow uint16 = (iota << 3)
        ActiveHigh

        CompPolMask uint16 = (1 << 3)
)

// Latch Comparator
const (
        CompNonLatch uint16 = (iota << 2)
        CompLatch

        CompLatchMask uint16 = (1 << 2)
)

// Comparator Queue
const (
        OneConv uint16 = (iota << 0)
        TwoConv
        FourConv
        DisableComp

        CompQueMask uint16 = (0b11 << 0)
)

// Device constants
const (
        Addr    = 0x48
        AddrGND = 0x48
        AddrVDD = 0x49
        AddrSDA = 0x4A
        AddrSCL = 0x4B
)
```

D.8.5 *ads1x15/muxsetting_string.go*

```
// Code generated by "stringer -type=MuxSetting"; DO NOT EDIT.

package ads1x15

import "strconv"

func _() {
        // An "invalid array index" compiler error signifies that the constant values have changed.
        // Re-run the stringer command to generate them again.
        var x [1]struct{}
        _ = x[Mux0_1-0]
        _ = x[Mux0_3-4096]
        _ = x[Mux1_3-8192]
        _ = x[Mux2_3-12288]
        _ = x[Mux0-16384]
        _ = x[Mux1-20480]
```

```
        _ = x[Mux2-24576]
        _ = x[Mux3-28672]
}

const (
        _MuxSetting_name_0 = "channel 0-1"
        _MuxSetting_name_1 = "channel 0-3"
        _MuxSetting_name_2 = "channel 1-3"
        _MuxSetting_name_3 = "channel 2-3"
        _MuxSetting_name_4 = "channel 0"
        _MuxSetting_name_5 = "channel 1"
        _MuxSetting_name_6 = "channel 2"
        _MuxSetting_name_7 = "channel 3"
)

func (i MuxSetting) String() string {
        switch {
        case i == 0:
                return _MuxSetting_name_0
        case i == 4096:
                return _MuxSetting_name_1
        case i == 8192:
                return _MuxSetting_name_2
        case i == 12288:
                return _MuxSetting_name_3
        case i == 16384:
                return _MuxSetting_name_4
        case i == 20480:
                return _MuxSetting_name_5
        case i == 24576:
                return _MuxSetting_name_6
        case i == 28672:
                return _MuxSetting_name_7
        default:
                return "MuxSetting(" + strconv.FormatInt(int64(i), 10) + ")"
        }
}
```

D.8.6 *ads1x15/options.go*

```
package ads1x15

import "fmt"

// Option defines a functional option for the device.
type Option func(d *Device) (Option, error)

// Options sets different configuration options and returns thte previous value
// of the last option passed.
func (d *Device) Options(options ...Option) (last Option, err error) {
        for _, opt := range options {
                last, err = opt(d)
                if err != nil {
                        return nil, err
                }
        }

        return last, nil
}

func (d *Device) config(reg byte, mask, flag uint16) (uint16, error) {
        cfg, err := d.ReadBytes(reg, 2)
        if err != nil {
                return 0, fmt.Errorf("could not get %#b from %#x: %w", mask, reg, err)
        }
        c := uint16(cfg[0])<<8 | uint16(cfg[1])

        old := c & mask
        c &= ^mask
        flag = flag & mask
        c |= flag
        if err := d.Write(reg, byte(c>>8), byte(c&0xFF)); err != nil {
                return 0, fmt.Errorf("could not set %#b from %#x: %w", flag, reg, err)
        }

        return old, nil
}

// Mux configures the input multiplexer.
func Mux(mode MuxSetting) Option {
        return func(d *Device) (Option, error) {
                old, err := d.config(CfgAddr, MuxMask, uint16(mode))
                if err != nil {
                        return nil, fmt.Errorf("ads1x15: could not configure multiplexer %#x: %w", mode, err)
                }
                return Mux(MuxSetting(old)), nil
        }
```

```go
}

// Gain configures the programmable gain amplifier.
func Gain(mode uint16) Option {
	return func(d *Device) (Option, error) {
		old, err := d.config(CfgAddr, PGAMask, mode)
		if err != nil {
			return nil, fmt.Errorf("ads1x15: could not configure gain amplifier %#x: %w", mode, err)
		}
		return Gain(old), nil
	}
}

// Rate configures the data rate.
func Rate(mode uint16) Option {
	return func(d *Device) (Option, error) {
		old, err := d.config(CfgAddr, DRMask, mode)
		if err != nil {
			return nil, fmt.Errorf("ads1x15: could not configure data rate %#x: %w", mode, err)
		}
		return Rate(old), nil
	}
}

// TraditionalMode sets the comparator to traditional mode.
func TraditionalMode() Option {
	return func(d *Device) (Option, error) {
		old, err := d.config(CfgAddr, CompModeMask, CompTrad)
		if err != nil {
			return nil, fmt.Errorf("ads1x15: could not set comparator mode to 'traditional': %w", err)
		}
		switch old {
		case CompTrad:
			return TraditionalMode(), nil
		case CompWindow:
			return WindowMode(), nil
		}
		return nil, fmt.Errorf("ads1x15: invalid previous comparator mode: got %#x", old)
	}
}

// WindowMode sets the comparator to traditional mode.
func WindowMode() Option {
	return func(d *Device) (Option, error) {
		old, err := d.config(CfgAddr, CompModeMask, CompWindow)
		if err != nil {
			return nil, fmt.Errorf("ads1x15: could not set comparator mode to 'window': %w", err)
		}
		switch old {
		case CompTrad:
			return TraditionalMode(), nil
		case CompWindow:
			return WindowMode(), nil
		}
		return nil, fmt.Errorf("ads1x15: invalid previous comparator mode: got %#x", old)
	}
}

// LowPolarity sets the comparator to traditional mode.
func LowPolarity() Option {
	return func(d *Device) (Option, error) {
		old, err := d.config(CfgAddr, CompPolMask, ActiveLow)
		if err != nil {
			return nil, fmt.Errorf("ads1x15: could not set comparator polarity to 'active low': %w", err)
		}
		switch old {
		case ActiveLow:
			return LowPolarity(), nil
		case ActiveHigh:
			return HighPolarity(), nil
		}
		return nil, fmt.Errorf("ads1x15: invalid previous comparator polarity: got %#x", old)
	}
}

// HighPolarity sets the comparator to traditional mode.
func HighPolarity() Option {
	return func(d *Device) (Option, error) {
		old, err := d.config(CfgAddr, CompPolMask, ActiveHigh)
		if err != nil {
			return nil, fmt.Errorf("ads1x15: could not set comparator polarity to 'active low': %w", err)
		}
		switch old {
		case ActiveLow:
			return LowPolarity(), nil
		case ActiveHigh:
			return HighPolarity(), nil
		}
		return nil, fmt.Errorf("ads1x15: invalid previous comparator polarity: got %#x", old)
	}
}

// NonLatch sets the comparator to traditional mode.
```

```
func NonLatch() Option {
	return func(d *Device) (Option, error) {
		old, err := d.config(CfgAddr, CompLatchMask, CompNonLatch)
		if err != nil {
			return nil, fmt.Errorf("ads1x15: could not set comparator latching mode to 'non-latching': %w", err)
		}
		switch old {
		case CompNonLatch:
			return NonLatch(), nil
		case CompLatch:
			return Latch(), nil
		}
		return nil, fmt.Errorf("ads1x15: invalid previous comparator latching mode: got %#x", old)
	}
}

// Latch sets the comparator to traditional mode.
func Latch() Option {
	return func(d *Device) (Option, error) {
		old, err := d.config(CfgAddr, CompLatchMask, CompLatch)
		if err != nil {
			return nil, fmt.Errorf("ads1x15: could not set comparator latching mode to 'latching': %w", err)
		}
		switch old {
		case CompNonLatch:
			return NonLatch(), nil
		case CompLatch:
			return Latch(), nil
		}
		return nil, fmt.Errorf("ads1x15: invalid previous comparator latching mode: got %#x", old)
	}
}

// Queue configures the comparator queue.
func Queue(mode uint16) Option {
	return func(d *Device) (Option, error) {
		old, err := d.config(CfgAddr, CompQueMask, mode)
		if err != nil {
			return nil, fmt.Errorf("ads1x15: could not configure comparator queue %#x: %w", mode, err)
		}
		return Queue(old), nil
	}
}

// DisableComparator disables the comparator.
func DisableComparator() Option {
	return func(d *Device) (Option, error) {
		old, err := Queue(DisableComp)(d)
		if err != nil {
			return nil, fmt.Errorf("ads1x15: could not configure disable comparator: %w", err)
		}
		return old, nil
	}
}

// SingleShot sets the measurement mode to single shot.
func SingleShot() Option {
	return func(d *Device) (Option, error) {
		old, err := d.config(CfgAddr, ModeMask, ModeSingleShot)
		if err != nil {
			return nil, fmt.Errorf("ads1x15: could not set mode to 'single shot': %w", err)
		}
		switch old {
		case ModeSingleShot:
			return SingleShot(), nil
		case ModeContinuous:
			return Continuous(), nil
		}
		return nil, fmt.Errorf("ads1x15: invalid previous measurement mode: got %#x", old)
	}
}

// Continuous sets the measurement mode to single shot.
func Continuous() Option {
	return func(d *Device) (Option, error) {
		old, err := d.config(CfgAddr, ModeMask, ModeContinuous)
		if err != nil {
			return nil, fmt.Errorf("ads1x15: could not set mode to 'continuous': %w", err)
		}
		switch old {
		case ModeSingleShot:
			return SingleShot(), nil
		case ModeContinuous:
			return Continuous(), nil
		}
		return nil, fmt.Errorf("ads1x15: invalid previous measurement mode: got %#x", old)
	}
}
```

D.9 Module: lsm6

D.9.1 License

D.9.2 *lsm6/const.go*

```go
package lsm6

// Register addresses
const (
	FuncCfgAccess = 0x01

	FIFOCtrl1  = 0x06
	FIFOCtrl2  = 0x07
	FIFOCtrl3  = 0x08
	FIFOCtrl4  = 0x09
	FIFOCtrl5  = 0x0A
	OrientCfgG = 0x0B

	Int1Ctrl = 0x0D
	Int2Ctrl = 0x0E
	WhoAmI   = 0x0F
	Ctrl1XL  = 0x10
	Ctrl2G   = 0x11
	Ctrl3C   = 0x12
	Ctrl4C   = 0x13
	Ctrl5C   = 0x14
	Ctrl6C   = 0x15
	Ctrl7G   = 0x16
	Ctrl8XL  = 0x17
	Ctrl9XL  = 0x18
	Ctrl10C  = 0x19

	WakeUpSrc = 0x1B
	TapSrc    = 0x1C
	D6DSrc    = 0x1D
	StatusReg = 0x1E

	OutTempL = 0x20
	OutTempH = 0x21
	OutXLG   = 0x22
	OutXHG   = 0x23
	OutYLG   = 0x24
	OutYHG   = 0x25
	OutZLG   = 0x26
	OutZHG   = 0x27
	OutXLXL  = 0x28
	OutXHXL  = 0x29
	OutYLXL  = 0x2A
	OutYHXL  = 0x2B
	OutZLXL  = 0x2C
	OutZHXL  = 0x2D

	FIFOStatus1   = 0x3A
	FIFOStatus2   = 0x3B
	FIFOStatus3   = 0x3C
	FIFOStatus4   = 0x3D
	FIFODataOutL  = 0x3E
```

```
        FIFODataOutH  = 0x3F
        Timestamp0Reg = 0x40
        Timestamp1Reg = 0x41
        Timestamp2Reg = 0x42

        StepTimestampL = 0x49
        StepTimestampH = 0x4A
        StepCounterL   = 0x4B
        StepCounterH   = 0x4C

        FuncSrc = 0x53

        TapCfg    = 0x58
        TapThs6D  = 0x59
        IntDur2   = 0x5A
        WakeUpThs = 0x5B
        WakeUpDur = 0x5C
        FreeFall  = 0x5D
        MD1Cfg    = 0x5E
        MD2Cfg    = 0x5F
)

// Device constants
const (
        Addr = 0x6B
)

const (
        maxADC  = (1 << 16) - 1
        halfADC = (1 << 15) - 1
)
```

D.9.3 lsm6/device.go

```
package lsm6

import (
        "fmt"

        "github.com/cgxeiji/serial"
)

// Device defines a LSM6 device.
type Device struct {
        i2c *serial.I2C

        accScale  float64
        gyroScale float64
}

// New returns a new LSM6 device.
func New(bus string, addr uint16) (*Device, error) {
        if addr == 0 {
                addr = Addr
        }

        i2c, err := serial.NewI2C(bus, addr)
        if err != nil {
                return nil, fmt.Errorf("lsm6: could not initialize I2C: %w", err)
        }

        d := &Device{
                i2c: i2c,
        }

        if err := d.i2c.Write(Ctrl1XL, 0x80); err != nil {
                return nil, fmt.Errorf("lsm6: could not configure device: %w", err)
        }
        d.accScale = 2

        if err := d.i2c.Write(Ctrl2G, 0x80); err != nil {
                return nil, fmt.Errorf("lsm6: could not configure device: %w", err)
        }
        d.gyroScale = 245

        if err := d.i2c.Write(Ctrl3C, 0x04); err != nil {
                return nil, fmt.Errorf("lsm6: could not configure device: %w", err)
        }

        return d, nil
}

// Close closes the device and cleans after itself.
func (d *Device) Close() {
        d.i2c.Close()
}

```

```go
// Acc returns the values of the accelerometer.
func (d *Device) Acc() (x, y, z float64, err error) {
	x, y, z, err = d.readPkg6(OutXLXL)
	if err != nil {
		return 0, 0, 0, fmt.Errorf("lsm6: could not read accelerometer: %w", err)
	}

	x *= d.accScale
	y *= d.accScale
	z *= d.accScale

	return x, y, z, nil
}

// Gyro returs the values of the gyroscope.
func (d *Device) Gyro() (x, y, z float64, err error) {
	x, y, z, err = d.readPkg6(OutXLG)
	if err != nil {
		return 0, 0, 0, fmt.Errorf("lsm6: could not read gyroscope: %w", err)
	}

	x *= d.gyroScale
	y *= d.gyroScale
	z *= d.gyroScale

	return x, y, z, nil
}

func (d *Device) readPkg6(reg byte) (x, y, z float64, err error) {
	b, err := d.i2c.ReadBytes(reg, 6)
	if err != nil {
		return 0, 0, 0, fmt.Errorf("lsm6: could not read %#x: %w", reg, err)
	}

	x = float64(
		int16(b[1])<<8|
			int16(b[0])) / halfADC
	y = float64(
		int16(b[3])<<8|
			int16(b[2])) / halfADC
	z = float64(
		int16(b[5])<<8|
			int16(b[4])) / halfADC

	return x, y, z, nil
}
```

D.10 Module: max3010x

D.10.1 License

D.10.2 *max3010x/beat.go*

```go
package max3010x

type beat struct {
	filterFIR *fir
	signal    struct {
		dc movingAverage
		ac struct {
			max  float64
			min  float64
			prev float64
		}
		rising bool
	}
}

func newBeat() *beat {
	return &beat{
		filterFIR: newFIR(),
	}
}

// check receives a normalized (0.0 - 1.0) signal input and checks for
// beats. It returns true on rising edges (positive zero crossings) or false
// otherwise.
func (b *beat) check(signal float64) bool {
	beat := false

	b.signal.dc.add(signal)
	ac := b.filterFIR.lowPass(signal - b.signal.dc.mean)

	// Rising edge
	if b.signal.ac.prev < 0 && ac >= 0 {
		delta := b.signal.ac.max - b.signal.ac.min
		if delta > 0.5 && delta < 50 {
			beat = true
		}

		b.signal.rising = true
		b.signal.ac.max = 0
	}

	// Falling edge
	if b.signal.ac.prev > 0 && ac <= 0 {
		b.signal.rising = false
		b.signal.ac.min = 0
	}

	if b.signal.rising {
		if ac > b.signal.ac.prev {
			b.signal.ac.max = ac
		}
	} else {
```

```go
		if ac < b.signal.ac.prev {
			b.signal.ac.min = ac
		}
	}

	b.signal.ac.prev = ac

	return beat
}
```

D.10.3 *max3010x/const.go*

```go
package max3010x

const (
	maxIntStatus  = 0x00
	maxIntEnable  = 0x01
	maxFifoWrPtr  = 0x02
	maxOvfCounter = 0x03
	maxFifoRdPtr  = 0x04
	maxFifoData   = 0x05
	maxModeCfg    = 0x06
	maxSpO2Cfg    = 0x07
	maxLedCfg     = 0x09
	maxTempInt    = 0x16
	maxTempFrac   = 0x17
	maxRevID      = 0xFE
	maxPartID     = 0xFF

	maxID00 = 0x11
	maxID02 = 0x15

	maxAddr = 0x57
)

const (
	sr50 = iota
	sr100
	sr167
	sr200
	sr400
	sr600
	sr800
	sr1000
)

const (
	modeHR   = 0b0000010
	modeSPO2 = 0b0000011
	modeTemp = 0b0001000
	modeRST  = 0b0100000
	modeSHDN = 0b1000000
)

const (
	pw200 = iota
	pw400
	pw800
	pw1600
)

const (
	mA0 = iota
	mA44
	mA76
	mA110
	mA142
	mA174
	mA208
	mA24
	mA271
	mA306
	mA338
	mA370
	mA402
	mA436
	mA468
	mA500
)
```

D.10.4 max3010x/fir.go

```go
package max3010x

var firC = []float64{21.5, 40.125, 72.375, 115.875, 170.0, 232.25, 298.75, 364.5, 423.875, 471.0, 501.5, 512.0}

const firSize = 32

type fir struct {
	buffer []float64
	idx    int
}

func newFIR() *fir {
	return &fir{
		buffer: make([]float64, firSize),
	}
}

// lowPass applies a low pass FIR filter to a delta.
func (f *fir) lowPass(delta float64) float64 {
	f.buffer[f.idx] = delta

	z := firC[11] * f.buffer[(f.idx-11)&0x1F]

	for i := 0; i < 11; i++ {
		z += firC[i] * (f.buffer[(f.idx-i)&0x1F] + f.buffer[(f.idx-(firSize-10)+i)&0x1F])
	}

	f.idx++
	f.idx %= firSize

	return z
}
```

D.10.5 max3010x/heartrate.go

```go
package max3010x

import (
	"context"
	"errors"
	"fmt"
	"time"
)

// HeartRate returns the current heart rate. Heart rate is expected to be
// between 10 to 250 beats per minute. Values outside that range are considered
// invalid and the function will continue to sample until a valid bpm is found.
// If no contact is detect on the sensor, this function returns 0 with an
// ErrNotDetected error. If the sensor cannot detect a beat after 1s, it
// returns 0 with an ErrTooNoisy error.
func (d *Device) HeartRate() (float64, error) {
	type beatPkg struct {
		span float64
		err  error
	}
	beatCh := make(chan beatPkg)

	ctx, cancel := context.WithTimeout(context.Background(), 7*time.Second)
	defer cancel()

	go func(ctx context.Context) {
		if err := d.detectBeat(ctx); err != nil {
			beatCh <- beatPkg{
				err: err,
			}
			return
		}
		timer := time.Now()

		for {
			select {
			case <-ctx.Done():
				beatCh <- beatPkg{
					err: ctx.Err(),
				}
			default:
			}

			if err := d.detectBeat(ctx); err != nil {
				beatCh <- beatPkg{
					err: err,
				}
				return
```

```go
			}
			t := time.Since(timer)

			if t > 6*time.Second { // less than 10 bpm
				continue // invalid
			}
			if t < 238*time.Millisecond { // more than 250 bpm
				continue // invalid
			}

			beatCh <- beatPkg{
				span: float64(t.Milliseconds()),
			}
			break
		}
	}(ctx)

	select {
	case <-ctx.Done():
		return 0, fmt.Errorf("max3010x: could not get heart rate: %w", ErrTooNoisy)

	case b := <-beatCh:
		if errors.Is(b.err, errLowValue) {
			d.hr.reset()
			return 0, fmt.Errorf("max3010x: could not get heart rate: %w", ErrNotDetected)
		} else if b.err != nil {
			d.hr.reset()
			return 0, fmt.Errorf("max3010x: could not get heart rate: %w", b.err)
		}

		// if first measurement, pre-fill values.
		if d.hr.mean == 0 {
			d.hr.mean = b.span
		}

		d.hr.add(b.span)
	}

	return 60000 / d.hr.mean, nil
}

func (d *Device) detectBeat(ctx context.Context) error {
	for {
		select {
		case <-ctx.Done():
			return ctx.Err()
		default:
		}

		err := d.ledsSingle()
		if err != nil {
			return fmt.Errorf("detectBeat: %w", err)
		}
		r := d.redLED.last()
		if r < threshold {
			return errLowValue
		}
		if d.beat.check(r) {
			break
		}
	}

	return nil
}
```

D.10.6 *max3010x/max3010x.go*

```go
package max3010x

import (
	"errors"
	"fmt"

	"github.com/cgxeiji/max3010x/max30102"
)

var (
	// ErrWrongDevice is thrown when trying to convert a max3010x.Device
	// interface to the underlying *Device struct and the device does not match
	// the PartID.
	ErrWrongDevice = errors.New("wrong device")
	// ErrNotDetected is thrown when trying to read a heart rate or SpO2 level
	// and nothing is detected on the sensor (e.g. no finger is placed on the
	// sensor when the function is called).
	ErrNotDetected = errors.New("nothing detected on the sensor")
	// ErrTooNoisy is thrown when trying to read data and has too much
	// variation, therefore consistent measurements cannot be done (e.g.
```

```go
	// ambient light, moving finger, etc.).
	ErrTooNoisy = errors.New("data has too much noise")

	errLowValue = errors.New("low value")
)

// Device defines a MAX3010x device.
type Device struct {
	sensor sensor
	redLED *tSeries
	irLED  *tSeries
	readCh chan struct{}

	hr   movingAverage
	spo2 movingAverage

	bus  string
	addr uint16

	beat *beat

	// PartID is the byte part ID as set by the manufacturer.
	// MAX30100: 0x11 or max30100.PartID
	// MAX30102: 0x15 or max30102.PartID
	PartID byte
	RevID  byte
}

type sensor interface {
	Temperature() (float64, error)
	RevID() (byte, error)
	Reset() error
	Calibrate() error

	IRRed() (float64, float64, error)
	IRRedBatch() ([]float64, []float64, error)

	Shutdown() error
	Startup() error

	Close()
}

const threshold = 0.10

// New returns a new MAX3010x device.
func New(options ...Option) (*Device, error) {
	d := &Device{
		readCh: make(chan struct{}, 1),
		beat:   newBeat(),
		irLED:  newTSeries(64),
		redLED: newTSeries(64),
	}

	for _, option := range options {
		option(d)
	}

	sensor, err := max30102.New(d.bus, d.addr)
	if err != nil {
		return nil, err
	}
	d.sensor = sensor

	d.PartID = max30102.PartID

	if d.RevID, err = d.sensor.RevID(); err != nil {
		return nil, fmt.Errorf("max3010x: could not get revision ID: %w", err)
	}

	d.readCh <- struct{}{}

	return d, nil
}

// Close closes the devices and cleans after itself.
func (d *Device) Close() {
	d.sensor.Close()
}

// Calibrate calibrates the power of each LED.
func (d *Device) Calibrate() error {
	return d.sensor.Calibrate()
}

// Temperature returns the current temperature of the device.
func (d *Device) Temperature() (float64, error) {
	return d.sensor.Temperature()
}

// ToMax30102 converts a max3010x device to a max30102 device to access low
// level functions. Check the package max3010x/max30102 for detailed behavior.
func (d *Device) ToMax30102() (*max30102.Device, error) {
```

```go
	device, ok := d.sensor.(*max30102.Device)
	if !ok {
		return nil, ErrWrongDevice
	}

	return device, nil
}

// Shutdown sets the device into power-save mode.
func (d *Device) Shutdown() error {
	return d.sensor.Shutdown()
}

// Startup wakes the device from power-save mode.
func (d *Device) Startup() error {
	return d.sensor.Startup()
}

func (d *Device) leds() error {
	select {
	case <-d.readCh:
		r, ir, err := d.sensor.IRRedBatch()
		if err != nil {
			return fmt.Errorf("could not get LEDs: %w", err)
		}
		d.redLED.add(r...)
		d.irLED.add(ir...)
		d.readCh <- struct{}{}

	default:
		select {
		case <-d.readCh:
			d.readCh <- struct{}{}
		}
	}
	return nil
}

func (d *Device) ledsSingle() error {
	select {
	case <-d.readCh:
		r, ir, err := d.sensor.IRRed()
		if err != nil {
			return fmt.Errorf("could not get LEDs: %w", err)
		}
		d.redLED.add(r)
		d.irLED.add(ir)
		d.readCh <- struct{}{}
	}
	return nil
}
```

D.10.7 *max3010x/moving_average.go*

```go
package max3010x

// movingAverage stores an estimated moving average of the last 4 values.
type movingAverage struct {
	mean float64
}

func (m *movingAverage) add(n float64) {
	m.mean += (n - m.mean) / 4
}

func (m *movingAverage) reset() {
	m.mean = 0
}
```

D.10.8 *max3010x/options.go*

```go
package max3010x

// An Option configures a device.
type Option func(d *Device) Option

// OnBus can be used to specify I²C bus name
// ("/dev/i2c-2", "I2C2", "2"). By default, the bus name is "", which selects
// the first available bus.
func OnBus(name string) Option {
	return func(d *Device) Option {
		old := d.bus
```

```
                d.bus = name
                return OnBus(old)
        }
}

// OnAddr can be used to specify alternative I²C name.
// By default, the address is 0x57.
func OnAddr(addr uint16) Option {
        return func(d *Device) Option {
                old := d.addr
                d.addr = addr
                return OnAddr(old)
        }
}
```

D.10.9 *max3010x/spo2.go*

```
package max3010x

import (
        "errors"
        "fmt"
)

// SpO2 returns the SpO2 value in 100%.
func (d *Device) SpO2() (float64, error) {
        r, err := d.rValue()
        if errors.Is(err, errLowValue) {
                d.spo2.reset()
                return 0, fmt.Errorf("max3010x: could not get SpO2: %w", ErrNotDetected)
        } else if err != nil {
                d.spo2.reset()
                return 0, fmt.Errorf("max3010x: could not get R value: %w", err)
        }

        spo2 := 104 - 17*r
        if spo2 <= 0 {
                return 0, nil
        }

        // if first measurement, pre-fill values.
        if d.spo2.mean == 0 {
                d.spo2.mean = spo2
        }
        d.spo2.add(spo2)

        return d.spo2.mean, nil
}

func (d *Device) rValue() (float64, error) {
        err := d.leds()
        if err != nil {
                return 0, err
        }

        if d.redLED.last() < threshold || d.irLED.last() < threshold {
                return 0, errLowValue
        }

        irACDC := d.irLED.acdc()
        if irACDC == 0 {
                return 0, nil
        }

        return d.redLED.acdc() / irACDC, nil
}
```

D.10.10 *max3010x/time_series.go*

```
package max3010x

type tSeries struct {
        buffer []float64
        idx    int

        max float64
        min float64
}

func newTSeries(size int) *tSeries {
        return &tSeries{
                buffer: make([]float64, size),
```

```
	}
}

func (t *tSeries) add(entries ...float64) {
	for _, e := range entries {
		t.idx++
		t.idx %= len(t.buffer)

		old := t.buffer[t.idx]
		t.buffer[t.idx] = e

		if old == t.max || old == t.min {
			t.max = e
			t.min = e
			for _, b := range t.buffer {
				t.minmax(b)
			}
		} else {
			t.minmax(e)
		}
	}

}

func (t *tSeries) minmax(v float64) {
	if v > t.max {
		t.max = v
	}
	if v < t.min {
		t.min = v
	}
}

func (t *tSeries) last() float64 {
	return t.buffer[t.idx]
}

func (t *tSeries) acdc() float64 {
	if t.min == 0 {
		return 0
	}

	return (t.max - t.min) / t.min
}
```

D.10.11 *max3010x/max3010x/main.go*

```
package main

import (
	"bufio"
	"errors"
	"fmt"
	"log"
	"os"
	"sync"
	"time"

	"github.com/cgxeiji/max3010x"
	"github.com/cgxeiji/max3010x/max30102"
)

func main() {
	sensor, err := max3010x.New()
	if err != nil {
		log.Fatal(err)
	}
	defer sensor.Close()

	// Check which sensor is connected using the PartID.
	switch sensor.PartID {
	case 0x11: // TODO: test with MAX30100
		fmt.Printf("MAX30100 rev.%d detected\n", sensor.RevID)
	case max30102.PartID:
		fmt.Printf("MAX30102 rev.%d detected\n", sensor.RevID)
	}
	fmt.Println("Press [ENTER] to exit")
	fmt.Println("--------------------------")

	done := make(chan struct{})
	var wg sync.WaitGroup

	// Read the heart rate every 200ms.
	hrCh := make(chan float64)
	wg.Add(1)
	go func() {
		defer wg.Done()
```

```
		t := time.NewTicker(200 * time.Millisecond)
		for {
			select {
			case <-done:
				return
			case <-t.C:
				hr, err := sensor.HeartRate()
				if errors.Is(err, max3010x.ErrNotDetected) {
					hr = 0
				} else if errors.Is(err, max3010x.ErrTooNoisy) {
					hr = -1
				} else if err != nil {
					log.Fatal(err)
				}
				select {
				case hrCh <- hr:
				case <-done:
				}
			}
		}
	}()

	// Read the SpO2 every 200ms.
	spO2Ch := make(chan float64)
	wg.Add(1)
	go func() {
		defer wg.Done()
		t := time.NewTicker(200 * time.Millisecond)
		for {
			select {
			case <-done:
				return
			case <-t.C:
				spO2, err := sensor.SpO2()
				if errors.Is(err, max3010x.ErrNotDetected) {
					spO2 = 0
				} else if err != nil {
					log.Fatal(err)
				}
				select {
				case spO2Ch <- spO2:
				case <-done:
				}
			}
		}
	}()

	// Read the sensor's temperature every second.
	tempCh := make(chan float64)
	wg.Add(1)
	go func() {
		defer wg.Done()
		t := time.NewTicker(1 * time.Second)
		for {
			select {
			case <-done:
				return
			case <-t.C:
				temp, err := sensor.Temperature()
				if err != nil {
					log.Fatal(temp)
				}
				select {
				case tempCh <- temp:
				case <-done:
				}
			}
		}
	}()

	// Access the underlying device for low level functions.
	// Read raw LED values as fast as possible.
	rawCh := make(chan []float64)
	wg.Add(1)
	go func() {
		defer wg.Done()
		device, err := sensor.ToMax30102()
		if errors.Is(err, max3010x.ErrWrongDevice) {
			fmt.Println("device is not MAX30102")
			return
		} else if err != nil {
			log.Fatal(err)
		}
		for {
			select {
			case <-done:
				return
			default:
			}
			ir, red, err := device.IRRed()
			if err != nil {
				log.Fatal(err)
			}
```

```go

			// Adjusting raw value for visualization
			ir -= 0.37
			ir *= 300
			if ir < 0 {
				ir = 0
			}

			// Adjusting raw value for visualization
			red -= 0.37
			red *= 300
			if red < 0 {
				red = 0
			}
			select {
			case rawCh <- []float64{red, ir}:
			case <-done:
			}
		}
	}()

	wg.Add(1)
	go func() {
		defer wg.Done()
		t := time.NewTicker(50 * time.Millisecond)
		fmt.Printf("\n\n\n\n\n")

		temp := 0.0
		hr := 0.0
		sp02 := 0.0
		raw := make([]float64, 2)
		for {
			select {
			case temp = <-tempCh:
			case hr = <-hrCh:
			case sp02 = <-sp02Ch:
			case raw = <-rawCh:
			case <-done:
				return
			}
			fmt.Printf("\033[5F")
			fmt.Printf("sensor temp\t: %2.1fC        \n", temp)
			switch hr {
			case 0:
				fmt.Printf("heart rate\t: --             \n")
			case -1:
				fmt.Printf("heart rate\t: too noisy      \n")
			default:
				fmt.Printf("heart rate\t: %3.2fbpm       \n", hr)
			}
			if sp02 == 0 {
				fmt.Printf("Sp02\t\t: --                   \n")
			} else {
				fmt.Printf("Sp02\t\t: %3.2f%%              \n", sp02)
			}
			fmt.Printf("red LED\t\t: %s              \n", float2bar(raw[0]))
			fmt.Printf("IR LED\t\t: %s               \n", float2bar(raw[1]))
			<-t.C
		}
	}()

	bufio.NewReader(os.Stdin).ReadString('\n')
	close(done)
	wg.Wait()
}

func float2bar(n float64) string {
	block := []string{"", "", "", "", "", "", "", "", "", ""}
	t := int(n)
	s := ""
	f := int((n - float64(t)) * 10)

	for i := 0; i < t; i++ {
		s += ""
	}
	s += block[f]

	return s
}
```

D.10.12 *max3010x/max30102/const.go*

```go
package max30102

// Register addresses
const (
	IntStat1         = 0x00
```

```
	IntStat2         = 0x01
	IntEna1          = 0x02
	IntEna2          = 0x03
	FIFOWrPtr        = 0x04
	OvfCount         = 0x05
	FIFORdPtr        = 0x06
	FIFOData         = 0x07
	FIFOCfg          = 0x08
	ModeCfg          = 0x09
	SpO2Cfg          = 0x0A
	Led1PA           = 0x0C
	Led2PA           = 0x0D
	MultiLedModeS2S1 = 0x11
	MultiLedModeS4S3 = 0x12
	TempInt          = 0x1F
	TempFrac         = 0x20
	TempCfg          = 0x21
	RegRevID         = 0xFE
	RegPartID        = 0xFF
)

// Interrupt flags
const (
	// Status 1
	AlmostFull            byte = (1 << 7)
	NewFIFOData           byte = (1 << 6)
	AmbientLightCancelOvf byte = (1 << 5)
	PowerReady            byte = (1 << 0)

	// Status 2
	DieTempReady byte = (1 << 1)
)

// Device constants
const (
	Addr   = 0x57
	PartID = 0x15
)

// Settings
const (
	TempEna      byte = 0b0000_0001
	ModeHR       byte = 0b010
	ModeSpO2     byte = 0b011
	ModeMultiLed byte = 0b111
	modeMask     byte = 0b1111_1000
	modeSHDN     byte = (1 << 7)

	ResetControl = 0b0100_0000
)

// SpO2 Sample Rate Control
const (
	SR50 = (iota << 2)
	SR100
	SR200
	SR400
	SR800
	SR1000
	SR1600
	SR3200

	srMask byte = 0b1_11_000_11
)

// LED Pulse Width Control
const (
	PW69 = iota
	PW118
	PW215
	PW411

	pwMask byte = 0b1_11_111_00
)

// masks
const (
	fifoFullMask byte = 0b111_1_0000
)

const (
	maxADC  = (1 << 18) - 1
	halfADC = (1 << 17) - 1
)
```

D.10.13 max3010x/max30102/max30102.go

```go
package max30102

import (
	"errors"
	"fmt"
	"time"

	"github.com/cgxeiji/serial"
)

var (
	// ErrNotDevice throws an error when the device part ID does not match a
	// MAX30102 signature (0x15).
	ErrNotDevice error = errors.New("max30102: part ID does not match (0x15)")
)

// Device defines a MAX30102 device.
type Device struct {
	i2c *serial.I2C
}

// New returns a new MAX30102 device. By default, this sets the LED pulse
// amplitude to 2.4mA, with a pulse width of 411us and a sample rate of 100
// samples/s.
//
// Argument "busName" can be used to specify the exact bus to use ("/dev/i2c-2", "I2C2", "2").
// Argument "addr" can be used to specify alternative address if default (0x57) is unavailable and changed.
// If "busName" argument is specified as an empty string "" the first available bus will be used.
func New(busName string, addr uint16) (*Device, error) {
	if addr == 0 {
		addr = Addr
	}

	i2c, err := serial.NewI2C(busName, addr)
	if err != nil {
		return nil, fmt.Errorf("max30102: could not initialize I2C: %w", err)
	}

	d := &Device{
		i2c: i2c,
	}

	part, err := d.Read(RegPartID)
	if err != nil {
		return nil, fmt.Errorf("max30102: could not get part ID: %w", err)
	}
	if part != PartID {
		return nil, ErrNotDevice
	}

	err = d.Reset()
	if err != nil {
		return nil, fmt.Errorf("max30102: could not reset device: %w", err)
	}
	if _, err = d.Options(
		RedPulseAmp(2.8),
		IRPulseAmp(2.8),
		PulseWidth(PW411),
		SampleRate(SR100),
		InterruptEnable(NewFIFOData|AlmostFull),
		AlmostFullValue(0),
		Mode(ModeSpO2),
	); err != nil {
		return nil, fmt.Errorf("max30102: could not initialize device: %w", err)
	}
	d.drain()

	return d, nil
}

// Close closes the device and cleans after itself.
func (d *Device) Close() {
	d.Shutdown()
	d.i2c.Close()
}

// RevID returns the revision ID of the device.
func (d *Device) RevID() (byte, error) {
	rev, err := d.Read(RegRevID)
	if err != nil {
		return 0, fmt.Errorf("max30102: could not get revision ID: %w", err)
	}
	return rev, nil
}

func (d *Device) waitUntil(reg, flag byte, bit byte) error {
	switch bit {
	case 1:
		for {
```

```go
			state, err := d.Read(reg)
			if err != nil {
				return fmt.Errorf("could not wait for %v in %v to be %v", flag, reg, bit)
			} else if state&flag != 0 {
				//fmt.Printf("%#x = %#b\n", reg, state)
				return nil
			}
		}
	case 0:
		for {
			if state, err := d.Read(reg); err != nil {
				return fmt.Errorf("could not wait for %v in %v to be %v", flag, reg, bit)
			} else if state&flag == 0 {
				//fmt.Printf("%#x = %#b\n", reg, state)
				return nil
			}
		}
	}

	return fmt.Errorf("invalid bit %v, it should be 1 or 0", bit)
}

func (d *Device) tempEnable() error {
	if err := d.Write(TempCfg, TempEna); err != nil {
		return fmt.Errorf("max30102: could not enable temperature: %w", err)
	}
	return nil
}

func (d *Device) tempReady() (bool, error) {
	state, err := d.Read(TempCfg)
	if err != nil {
		return false, fmt.Errorf("max30102: could not read temperature state: %w", err)
	}
	return (state & TempEna) == 0, nil
}

// Temperature returns the current temperature of the device.
func (d *Device) Temperature() (float64, error) {
	if err := d.tempEnable(); err != nil {
		return 0, err
	}
	if err := d.waitUntil(TempCfg, TempEna, 0); err != nil {
		return 0, err
	}

	i, err := d.Read(TempInt)
	if err != nil {
		return 0, fmt.Errorf("max30102: could not read integer part of temperature: %w", err)
	}

	f, err := d.Read(TempFrac)
	if err != nil {
		return 0, fmt.Errorf("max30102: could not read fractional part of temperature: %w", err)
	}

	return float64(int8(i)) + (float64(f) * 0.0625), nil
}

// Read reads a single byte from a register.
func (d *Device) Read(reg byte) (byte, error) {
	return d.i2c.Read(reg)
}

// ReadBytes reads n bytes from a register.
func (d *Device) ReadBytes(reg byte, n int) ([]byte, error) {
	return d.i2c.ReadBytes(reg, n)
}

// Write writes a byte to a register.
func (d *Device) Write(reg, data byte) error {
	return d.i2c.Write(reg, data)
}

// Reset resets the device. All configurations, thresholds, and data registers
// are reset to their power-on state.
func (d *Device) Reset() error {
	if err := d.Write(ModeCfg, ResetControl); err != nil {
		return fmt.Errorf("max30102: could not reset: %w", err)
	}
	if err := d.waitUntil(ModeCfg, ResetControl, 0); err != nil {
		return fmt.Errorf("max30102: could not reset: %w", err)
	}

	return nil
}

// IRRed returns the value of the red LED and IR LED. The values are normalized
// from 0.0 to 1.0.
func (d *Device) IRRed() (ir, red float64, err error) {
	const msbMask byte = 0b0000_0011

	err = d.waitUntil(IntStat1, NewFIFOData, 1)
```

```go
	if err != nil {
		return 0, 0, err
	}

	bytes, err := d.ReadBytes(FIFOData, 6)
	if err != nil {
		return 0, 0, err
	}

	ir = float64(
		int(bytes[3]&msbMask)<<16|
			int(bytes[4])<<8|
			int(bytes[5])) / maxADC
	red = float64(
		int(bytes[0]&msbMask)<<16|
			int(bytes[1])<<8|
			int(bytes[2])) / maxADC

	return ir, red, nil
}

// IRRedBatch returns a batch of IR and red LED values based on the AlmostFull
// flag. The amount of data returned can be configured by setting the
// AlmostFullValue leftover value, which is set to 0 by default. Therefore,
// this function returns 32 samples by default.
func (d *Device) IRRedBatch() (ir, red []float64, err error) {
	const maxADC = 262143
	const msbMask byte = 0b0000_0011

	err = d.drain()
	if err != nil {
		return nil, nil, fmt.Errorf("max30102: could not empty FIFO: %w", err)
	}
	err = d.waitUntil(IntStat1, AlmostFull, 1)
	if err != nil {
		return nil, nil, fmt.Errorf("max30102: error waiting for almost full interrupt: %w", err)
	}

	n, err := d.available()
	if err != nil {
		return nil, nil, fmt.Errorf("max30102: error reading available data: %w", err)
	}

	ir = make([]float64, n)
	red = make([]float64, n)
	for i := 0; i < n; i++ {
		bytes, err := d.ReadBytes(FIFOData, 6)
		if err != nil {
			return nil, nil, err
		}

		irData := float64(
			int(bytes[3]&msbMask)<<16|
				int(bytes[4])<<8|
				int(bytes[5])) / maxADC
		redData := float64(
			int(bytes[0]&msbMask)<<16|
				int(bytes[1])<<8|
				int(bytes[2])) / maxADC

		ir[i] = irData
		red[i] = redData
	}

	return ir, red, nil
}

func (d *Device) drain() error {
	n, err := d.available()
	if err != nil {
		return err
	}
	for i := 0; i < n; i++ {
		_, err := d.ReadBytes(FIFOData, 6)
		if err != nil {
			return err
		}
	}
	return nil
}

func (d *Device) available() (int, error) {
	wr, err := d.Read(FIFOWrPtr)
	if err != nil {
		return 0, nil
	}
	rd, err := d.Read(FIFORdPtr)
	if err != nil {
		return 0, nil
	}

	if wr == rd {
		return 32, nil
```

```go
	}
	return (int(wr) + 32 - int(rd)) % 32, nil
}

// Calibrate auto-calibrates the current of each LED.
func (d *Device) Calibrate() error {
	var ir []float64
	var red []float64
	var err error

	irAmp := 0.0
	redAmp := 0.0

	if _, err = d.Options(
		IRPulseAmp(irAmp),
		RedPulseAmp(redAmp),
	); err != nil {
		return fmt.Errorf("max30102: could not calibrate sensor: %w", err)
	}

	for mean(ir) < 0.4 {
		if irAmp >= 5 {
			break
		}
		irAmp += 0.5

		if _, err = d.Options(
			IRPulseAmp(irAmp),
		); err != nil {
			return fmt.Errorf("max30102: could not calibrate sensor: %w", err)
		}
		time.Sleep(40 * time.Millisecond)

		ir, red, err = d.IRRedBatch()
		if err != nil {
			return fmt.Errorf("max30102: could not calibrate sensor: %w", err)
		}
	}

	for mean(red) < 0.4 {
		if redAmp >= 5 {
			break
		}
		redAmp += 0.5

		if _, err = d.Options(
			RedPulseAmp(redAmp),
		); err != nil {
			return fmt.Errorf("max30102: could not calibrate sensor: %w", err)
		}
		time.Sleep(40 * time.Millisecond)

		ir, red, err = d.IRRedBatch()
		if err != nil {
			return fmt.Errorf("max30102: could not calibrate sensor: %w", err)
		}
	}

	fmt.Println("calibration:")
	fmt.Printf("  irAmp = %.1fmA\n", irAmp)
	fmt.Printf("  redAmp = %.1fmA\n", redAmp)

	return nil
}

func mean(a []float64) float64 {
	if len(a) == 0 {
		return 0
	}

	r := 0.0
	for _, v := range a {
		r += v
	}

	return r / float64(len(a))
}

// Shutdown sets the device into power-save mode.
func (d *Device) Shutdown() error {
	_, err := d.config(ModeCfg, ^modeSHDN, modeSHDN)

	return err
}

// Startup wakes the device from power-save mode.
func (d *Device) Startup() error {
	_, err := d.config(ModeCfg, ^modeSHDN, ^modeSHDN)

	return err
}

func (d *Device) debugRegister(reg byte) {
```

```go
	b, _ := d.Read(reg)
	fmt.Printf("%#x = %#x (%#b)\n", reg, b, b)
}
```

D.10.14 *max3010x/max30102/options.go*

```go
package max30102

import "fmt"

// Option defines a functional option for the device.
type Option func(d *Device) (Option, error)

// Options sets different configuration options and returns the previous value
// of the last option passed.
func (d *Device) Options(options ...Option) (Option, error) {
	var old Option
	var err error
	for _, opt := range options {
		old, err = opt(d)
		if err != nil {
			return nil, err
		}
	}

	return old, nil
}

func (d *Device) config(reg, mask, flag byte) (byte, error) {
	cfg, err := d.Read(reg)
	if err != nil {
		return 0, fmt.Errorf("could not get %v from %v: %w", mask, reg, err)
	}
	old := cfg &^ mask
	cfg &= mask
	flag = flag &^ mask
	cfg |= flag
	if err := d.Write(reg, cfg); err != nil {
		return 0, fmt.Errorf("could not set %v in %v: %w", flag, reg, err)
	}

	return old, nil
}

// Mode sets the operation mode of the device.
func Mode(mode byte) Option {
	return func(d *Device) (Option, error) {
		old, err := d.config(ModeCfg, modeMask, mode)
		if err != nil {
			return nil, fmt.Errorf("max30102: could not configure mode %#x: %w", mode, err)
		}

		if err = d.Write(FIFOWrPtr, 0); err != nil {
			return nil, fmt.Errorf("max30102: could not configure mode %#x: %w", mode, err)
		}
		if err = d.Write(OvfCount, 0); err != nil {
			return nil, fmt.Errorf("max30102: could not configure mode %#x: %w", mode, err)
		}
		if err = d.Write(FIFORdPtr, 0); err != nil {
			return nil, fmt.Errorf("max30102: could not configure mode %#x: %w", mode, err)
		}

		return Mode(old), nil
	}
}

// RedPulseAmp sets the pulse amplitude of the red LED. It accepts values
// from 0.0 to 51.0 mA and the value is rounded down to the nearest multiple of 0.2.
func RedPulseAmp(current float64) Option {
	return func(d *Device) (Option, error) {
		if current > 51 {
			current = 51
		}
		if current < 0 {
			current = 0
		}
		b := byte(current * 5)

		old, err := d.config(Led1PA, 0, b)
		if err != nil {
			return nil, fmt.Errorf("max30102: could not configure red LED pulse amplitud: %w", err)
		}

		return RedPulseAmp(float64(old) / 5), nil
	}
}

```

```go
// IRPulseAmp sets the pulse amplitude of the red LED. It accepts values
// from 0.0 to 51.0 mA and the value is rounded down to the nearest multiple of 0.2.
func IRPulseAmp(current float64) Option {
	return func(d *Device) (Option, error) {
		if current > 51 {
			current = 51
		}
		if current < 0 {
			current = 0
		}
		b := byte(current * 5)

		old, err := d.config(Led2PA, 0, b)
		if err != nil {
			return nil, fmt.Errorf("max30102: could not configure IR LED pulse amplitud: %w", err)
		}

		return IRPulseAmp(float64(old) / 5), nil
	}
}

// PulseWidth sets the pulse width of the device.
func PulseWidth(pw byte) Option {
	return func(d *Device) (Option, error) {
		old, err := d.config(SpO2Cfg, pwMask, pw)
		if err != nil {
			return nil, fmt.Errorf("max30102: could not configure pulse width: %w", err)
		}

		return PulseWidth(old), nil
	}
}

// SampleRate sets the SpO2 sample rate control of the device.
func SampleRate(sr byte) Option {
	return func(d *Device) (Option, error) {
		old, err := d.config(SpO2Cfg, srMask, sr)
		if err != nil {
			return nil, fmt.Errorf("max30102: could not configure sample rate: %w", err)
		}

		return SampleRate(old), nil
	}
}

// InterruptEnable enables interrupts.
func InterruptEnable(i byte) Option {
	return func(d *Device) (Option, error) {
		old, err := d.config(IntEna1, ^i, i)
		if err != nil {
			return nil, fmt.Errorf("max30102: could not configure interrupt flags: %w", err)
		}

		return InterruptEnable(old), nil
	}
}

// AlmostFullValue sets when the AlmostFull interrupt should be triggered. It
// can take values from 0 to 15.
func AlmostFullValue(left byte) Option {
	return func(d *Device) (Option, error) {
		left &= ^fifoFullMask
		old, err := d.config(FIFOCfg, fifoFullMask, left)
		if err != nil {
			return nil, fmt.Errorf("max30102: could not configure almost full value to %d: %w", left, err)
		}

		return AlmostFullValue(old), nil
	}
}
```

D.11 Module: serial

D.11.1 License

D.11.2 *serial/serial.go*

```go
package serial

import (
	"fmt"

	"periph.io/x/periph/conn/i2c"
	"periph.io/x/periph/conn/i2c/i2creg"
	"periph.io/x/periph/host"
)

// I2C defines an I2C type to read and write from/to registers.
type I2C struct {
	dev  *i2c.Dev
	bus  i2c.BusCloser
	addr uint16
}

// NewI2C returns a new I2C interface at the specified bus and address.
// If `bus` is set to "", the first available bus is used. The address must
// always be specified.
func NewI2C(bus string, addr uint16) (*I2C, error) {
	if _, err := host.Init(); err != nil {
		return nil, fmt.Errorf("serial: could not initialize host: %w", err)
	}

	b, err := i2creg.Open(bus)
	if err != nil {
		return nil, fmt.Errorf("serial: could not open I2C bus: %w", err)
	}

	dev := &i2c.Dev{
		Addr: addr,
		Bus:  b,
	}

	i2c := &I2C{
		dev:  dev,
		bus:  b,
		addr: addr,
	}

	return i2c, nil
}

// Read reads a single byte from a register.
func (i *I2C) Read(reg byte) (byte, error) {
	b := make([]byte, 1)
	if err := i.dev.Tx([]byte{reg}, b); err != nil {
		return 0, fmt.Errorf("serial: could not read byte from register %x at address %x: %w", reg, i.addr, err)
	}

	return b[0], nil
```

```go
}

// ReadBytes reads n bytes from a register.
func (i *I2C) ReadBytes(reg byte, n int) ([]byte, error) {
	b := make([]byte, n)
	if err := i.dev.Tx([]byte{reg}, b); err != nil {
		return nil, fmt.Errorf("serial: could not read all %d bytes from register %x at address %x: %w", n, reg, i.addr, err)
	}

	return b, nil
}

// Write writes a byte or bytes to a register.
func (i *I2C) Write(reg byte, data ...byte) error {
	n, err := i.dev.Write(append([]byte{reg}, data...))
	if err != nil {
		return fmt.Errorf("serial: could not write %x to register %x at address %x: %w", data, reg, i.addr, err)
	}
	n-- // remove register write
	if n != len(data) {
		return fmt.Errorf("serial: wrong number of bytes written: want %d, got %d", len(data), n)
	}

	return nil
}

// Close closes the bus used by I2C.
func (i *I2C) Close() {
	i.bus.Close()
}
```

Bibliography

Adafruit Industries (2021a). *Adafruit Micro-Lipo Charger for LiPoly Batt with USB Type C Jack*. URL: https://www.adafruit.com/product/4410 (visited on 08/13/2021).

— (2021b). *ADS1015 12-Bit ADC - 4 Channel with Programmable Gain Amplifier : ID 1083 : $9.95 : Adafruit Industries, Unique & fun DIY electronics and kits*. URL: https://www.adafruit.com/product/1083 (visited on 09/14/2021).

— (2021c). *NeoPixel Ring - 12 x 5050 RGB LED with Integrated Drivers*. URL: https://www.adafruit.com/product/1643 (visited on 09/25/2021).

— (2021d). *PowerBoost 500 Basic - 5V USB Boost @ 500mA from 1.8V+*. URL: https://www.adafruit.com/product/1903 (visited on 08/13/2021).

— (2021e). *Zero Spy Camera for Raspberry Pi Zero ID: 3508*. URL: https://www.adafruit.com/product/3508 (visited on 09/25/2021).

Admoni, Henny and Brian Scassellati (Mar. 1, 2017). "Social Eye Gaze in Human-Robot Interaction: A Review". In: *Journal of Human-Robot Interaction* 6.1, pp. 25–63. DOI: 10.5898/jhri.6.1.admoni.

Akash, Kumar et al. (Nov. 22, 2018). "A Classification Model for Sensing Human Trust in Machines Using EEG and GSR". In: *ACM Transactions on Interactive Intelligent Systems* 8.4, pp. 1–20. ISSN: 2160-6455. DOI: 10.1145/3132743.

Amazon.com Inc (2021). *Amazon.com: Alexa*. URL: https://www.amazon.com/b?node=21576558011 (visited on 09/22/2021).

Andreasson, Rebecca et al. (Dec. 1, 2017). "Affective Touch in Human–Robot Interaction: Conveying Emotion to the Nao Robot". In: *International Journal of Social Robotics* 10.4, pp. 473–491. ISSN: 1875-4791. DOI: 10.1007/s12369-017-0446-3.

Andrist, Sean et al. (2014). "Conversational gaze aversion for humanlike robots". In: *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction - HRI '14*. DOI: 10.1145/2559636.2559666.

Appelhans, Bradley M. and Linda J. Luecken (Sept. 2006). "Heart Rate Variability as an Index of Regulated Emotional Responding". In: *Review of General Psychology* 10.3, pp. 229–240. ISSN: 1089-2680. DOI: 10.1037/1089-2680.10.3.229.

Azarbarzin, Ali et al. (Apr. 1, 2014). "Relationship between Arousal Intensity and Heart Rate Response to Arousal". In: *Sleep* 37.4, pp. 645–653. ISSN: 0161-8105. DOI: 10.5665/sleep.3560.

Baldwin, Dare A. and Jodie A. Baird (Apr. 2001). "Discerning intentions in dynamic human action". In: *Trends in Cognitive Sciences* 5.4, pp. 171–178. ISSN: 1364-6613. DOI: 10.1016/s1364-6613(00)01615-6.

Baraka, Kim, Ana Paiva, and Manuela Veloso (Dec. 2, 2015). "Expressive Lights for Revealing Mobile Service Robot State". In: *Advances in Intelligent Systems and Computing*, pp. 107–119. ISSN: 978-3-319-27146-0. DOI: 10.1007/978-3-319-27146-0_9.

Baron-Cohen, Simon, Sally Wheelwright, and Therese Jolliffe (Sept. 1997). "Is There a "Language of the Eyes"? Evidence from Normal Adults, and Adults with Autism or Asperger Syndrome". In: *Visual Cognition* 4.3, pp. 311–331. ISSN: 1350-6285. DOI: 10.1080/713756761.

Bartneck, Christoph et al. (Feb. 19, 2009a). "Does the Design of a Robot Influence Its Animacy and Perceived Intelligence?" In: *International Journal* *of Social Robotics* 1.2, pp. 195–204. ISSN: 1875-4791. DOI: 10.1007/s12369-009-0013-7.

Bartneck, Christoph et al. (Jan. 2009b). "Measurement Instruments for the Anthropomorphism, Animacy, Likeability, Perceived Intelligence, and Perceived Safety of Robots". In: *International Journal of Social Robotics* 1.1, pp. 71–81. ISSN: 1875-4791, 1875-4805. DOI: 10.1007/s12369-008-0001-3.

Bates, Douglas et al. (2015). "Fitting Linear Mixed-Effects Models Usinglme4". In: *Journal of Statistical Software* 67.1. ISSN: 1548-7660. DOI: 10.18637/jss.v067.i01.

Bateson, Mary Catherine (Sept. 1975). "Mother-Infant Exchanges: The Epigenesis of Conversational Interaction". In: *Annals of the New York Academy of Sciences* 263 (1 Developmental), pp. 101–113. ISSN: 0077-8923, 1749-6632. DOI: 10.1111/j.1749-6632.1975.tb41575.x.

Benoit, Kenneth, David Muhr, and Kohei Watanabe (Feb. 10, 2021). *Package 'stopwords'*. CRAN. 9 pp. URL: https://cran.r-project.org/web/packages/stopwords/stopwords.pdf.

Bentivoglio, Anna Rita et al. (Nov. 1997). "Analysis of blink rate patterns in normal subjects". In: *Movement Disorders* 12.6, pp. 1028–1034. DOI: 10.1002/mds.870120629.

Betella, Alberto and Paul F. M. J. Verschure (Feb. 5, 2016). "The Affective Slider: A Digital Self-Assessment Scale for the Measurement of Human Emotions". In: *PLOS ONE* 11.2. DOI: 10.1371/journal.pone.0148037.

Betella, Alberto and Paul F. M. J. Vershure (2016). *The Sefl-Assessment Manikin (SAM), adapted with permission from Bradley and Lang 1994*. Figure. License: CC BY 4.0. URL: https://www.researchgate.net/figure/The-Self-Assessment-Manikin-SAM-adapted-with-permission-from-Bradley-and-Lang-1994_fig5_293120723 (visited on 09/22/2021).

Bickmore, Timothy W. and Rosalind W. Picard (Apr. 24, 2004). "Towards Caring Machines". In: *Extended Abstracts on Human Factors in Computing Systems*. ACM Press, pp. 1489–1492. ISBN: 978-1-58113-703-3. DOI: 10.1145/985921.986097.

Bordin, Edward S. (1979). "The generalizability of the psychoanalytic concept of the working alliance." In: *Psychotherapy: Theory, Research & Practice* 16.3, pp. 252–260. DOI: 10.1037/h0085885.

Boston Dynamics (2021). *Spot*. URL: https://shop.bostondynamics.com/spot?cclcl=en_US&pid=aDl6g000000XdpZCAS (visited on 09/14/2021).

Boucsein, Wolfram (2012). *Electrodermal Activity*. 2nd ed. US: Springer US. ISBN: 978-1-4614-1125-3. DOI: 10.1007/978-1-4614-1126-0.

Bradley, Margaret M. and Peter J. Lang (Mar. 1994). "Measuring Emotion: The Self-Assessment Manikin and the Semantic Differential". In: *Journal of Behavior Therapy and Experimental Psychiatry* 25.1, pp. 49–59. ISSN: 00057916. DOI: 10.1016/0005-7916(94)90063-9.

Breazeal, Cynthia L. (2002). *Designing Sociable Robots*. Intelligent robots and autonomous agents. Cambridge, Mass: MIT Press. 263 pp. ISBN: 978-0-262-02510-2.

Bynion, Teah-Marie and Matthew T. Feldner (2017). *Self-Assessment Manikin*. Ed. by Virgil Zeigler-Hill and Todd K. Shackelford. Cham. DOI: 10.1007/978-3-319-28099-8_77-1.

Carpinella, Colleen M. et al. (2017). "The Robotic Social Attributes Scale (RoSAS): Development and Validation". In: ACM Press, pp. 254–262. ISBN: 978-1-4503-4336-7. DOI: 10.1145/2909824.3020208.

Cañigueral, Roser and Antonia F. de C. Hamilton (Mar. 15, 2019). "The Role of Eye Gaze During Natural Social Interactions in Typical and Autistic People". In: *Frontiers in Psychology* 10. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2019.00560.

Chernova, Sonia and Andrea L. Thomaz (Apr. 30, 2014). "Robot Learning from Human Teachers". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8.3, pp. 1–121. ISSN: 1939-4608, 1939-4616. DOI: 10.2200/S00568ED1V01Y201402AIM028.

Christopoulos, George I., Marilyn A. Uy, and Wei Jie Yap (Dec. 8, 2016). "The Body and the Brain: Measuring Skin Conductance Responses to Understand the Emotional Experience". In: *Organizational Research Methods* 22.1, pp. 394–420. ISSN: 1094-4281. DOI: 10.1177/1094428116681073.

Cohen, Jacob (1988). *Statistical Power Analysis for the Behavioral Sciences*. 2nd ed. Routledge. ISBN: 0805802835.

Collins English Dictionary - Complete and Unabridged (2014). *animacy*. Dictionary.

Combine Overwiki (May 1, 2011). *In-game screenshot*. Fair use. URL: https://en.wikipedia.org/w/index.php?curid=31623959 (visited on 10/02/2021).

Creighton, Susan and Andrea Szymkowiak (Aug. 2014). "The Effects of Co-operative and Competitive Games on Classroom Interaction Frequencies". In: *Procedia - Social and Behavioral Sciences* 140, pp. 155–163. ISSN: 1877-0428. DOI: 10.1016/j.sbspro.2014.04.402.

Denshinbashira (Nov. 14, 2016). *「ぼくのなつやすみ」よりボクくん (Boku-kun from 'My Summer Vacation')*. Online. URL: http://denshinbashira.blog76.fc2.com/blog-entry-28.html (visited on 10/02/2021).

Ekman, Paul and Wallace V. Friesen (2003). *UNMASKING THE FACE: A Guide to Recognizing Emotions from Facial Expressions*. Malor Books. 212 pp. ISBN: 1-883536-36-7.

Ekman, Paul, Wallace V. Friesen, and PHOEBE Ellsworth (1972). "What Emotion Categories Can Observers Judge from Facial Behavior?" In: *Emotion in the Human Face*, pp. 57–65. DOI: 10.1016/b978-0-08-016643-8.50024-0.

Eyssel, Friederike and Dieta Kuchenbrandt (Nov. 21, 2011). "Social categorization of social robots: Anthropomorphism as a function of robot group membership". In: *British Journal of Social Psychology* 51.4, pp. 724–731. ISSN: 0144-6665. DOI: 10.1111/j.2044-8309.2011.02082.x.

Eyssel, Friederike et al. (Mar. 2012). "'If you sound like me, you must be more human': On the Interplay of Robot and User Features on Human-Robot Acceptance and Anthropomorphism". In: *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction - HRI '12*. IEEE. DOI: 10.1145/2157689.2157717.

Feinerer, Ingo, Kurt Hornik, and Artifex Software, Inc. (Nov. 18, 2020). *Package 'tm'*. CRAN. 64 pp. URL: https://cran.r-project.org/web/packages/tm/tm.pdf.

Figner, Bernd and Ryan O. Murphy (Jan. 2011). "Using skin conductance in judgment and decision making research". In: *A handbook of process tracing methods for decision research: A critical review and user's guide*. Ed. by M. Schulte-Mecklenbeck, A. Kühberger, and R. Ranyard. Psychology Press, pp. 163–184.

Fritzing (Sept. 24, 2021). *fritzing: electronics made easy*. URL: https://fritzing.org/ (visited on 10/02/2021).

Funakoshi, Kotaro et al. (2008). "Smoothing human-robot speech interactions by using a blinking-light as subtle expression". In: *Proceedings of the 10th international conference on Multimodal interfaces - IMCI '08*. ACM Press, pp. 293–296. ISBN: 9781605581989. DOI: 10.1145/1452392.1452452.

Furnham, Adrian (Jan. 1986). "Response bias, social desirability and dissimulation". In: *Personality and Individual Differences* 7.3, pp. 385–400. DOI: 10.1016/0191-8869(86)90014-0.

Geethanjali, B. et al. (2017). "Emotion analysis using SAM (Self-Assessment Manikin) scale". In: *Biomed Research* (Special Issue), S18–S24. ISSN: 0970-938X.

GeGeGe no Kitaro Wiki (2021). *Tumblr p2s1tyuuwm1tnahllo5 500.png*. Online. URL: `https://gegegenokitaro.fandom.com/wiki/Medama-Oyaji` (visited on 10/02/2021).

Gent, Paul van et al. (Oct. 2019). "HeartPy: A novel heart rate algorithm for the analysis of noisy signals". In: *Transportation Research Part F: Traffic Psychology and Behaviour* 66, pp. 368–378. ISSN: 1369-8478. DOI: `10.1016/j.trf.2019.09.015`.

Gibson, Carolyn E., Joy Losee, and Christine Vitiello (May 2014). "A Replication Attempt of Stereotype Susceptibility (Shih, Pittinsky, & Ambady, 1999) - Identity Salience and Shifts in Quantitative Performance". In: *Social Psychology* 45.3. DOI: `10.1027/1864-9335/a000184`.

Go (2021). *The Go Programming Language*. URL: `https://golang.org/` (visited on 08/10/2021).

Godot (2021). *Godot Engine - Free and open source 2D and 3D game engine*. URL: `https://godotengine.org/` (visited on 09/14/2021).

Goldman, Susan R., Arthur C. Graesser, and Paul van den Broek, eds. (Aug. 1, 1999). *Narrative Comprehension, Causality, and Coherence*. 1st ed. Routledge. ISBN: 9781410603135.

Google Nest (2021). *Nest Audio*. URL: `https://store.google.com/us/product/nest_audio?hl=en-US` (visited on 09/22/2021).

Gouaillier, David et al. (May 2009). "Mechatronic design of NAO humanoid". In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. DOI: `10.1109/robot.2009.5152516`.

Graybiel, Ashton et al. (Apr. 1944). "Analysis of the electrocardiograms obtained from 1000 young healthy aviators". In: *American Heart Journal* 27.4, pp. 524–549. ISSN: 0002-8703. DOI: 10.1016/s0002-8703(44)90546-6.

Greco, Alberto et al. (2016). "cvxEDA: a Convex Optimization Approach to Electrodermal Activity Processing". In: *IEEE Transactions on Biomedical Engineering* 63.4, pp. 1–1. ISSN: 0018-9294. DOI: 10.1109/tbme.2015.2474131.

Han, JingGuang et al. (Dec. 2012). "Investigating the use of Non-verbal Cues in Human-Robot Interaction with a Nao robot". In: *2012 IEEE 3rd International Conference on Cognitive Infocommunications (CogInfoCom)*. IEEE. DOI: 10.1109/coginfocom.2012.6421937.

Hatcher, Robert L. and J. Arthur Gillaspy (Jan. 2006). "Development and validation of a revised short version of the working alliance inventory". In: *Psychotherapy Research* 16.1, pp. 12–25. DOI: 10.1080/10503300500352500.

HiLetgo (May 2, 2019). *HiLetgo MAX30102 低電力心拍数クリックセンサーブレークアウトボードモジュール Arduino パルスオキシメトリーソリューション SpO2 MAX30100 の交換*. online. URL: https://www.amazon.co.jp/dp/B07QC67KMQ/ (visited on 09/02/2021).

Hirst, Richard et al. (2013). *pi-blaster*. Computer software. URL: https://github.com/sarfata/pi-blaster.

HobbyKing (2021). *Towerpro MG996R Servo 10kg / 0.20sec / 55g*. Photo. URL: https://hobbyking.com/en_us/towerpro-mg996r-10kg-servo-10kg-0-20sec-55g.html (visited on 09/20/2021).

Hoffman, Guy (Apr. 1, 2019). "Evaluating Fluency in Human-Robot Collaboration". In: *IEEE Transactions on Human-Machine Systems* 49.3, pp. 209–218. DOI: 10.1109/THMS.2019.2904558.

Hogan, James L., Roger H. Fisher, and Bruce John Morrison (June 1, 1974). "Social Feedback and Cooperative Game Behavior". In: *Psychological Reports* 34.3, pp. 1075–1082. ISSN: 0033-2941. DOI: 10.2466/pr0.1974.34.3c.1075.

Holroyd, Aaron et al. (July 2011). "Generating connection events for human-robot collaboration". In: *2011 RO-MAN*. DOI: 10.1109/roman.2011.6005245.

Horvath, Adam O. and Leslie S. Greenberg (1989). "Development and validation of the Working Alliance Inventory". In: *Journal of Counseling Psychology* 36.2, pp. 223–233. DOI: 10.1037/0022-0167.36.2.223.

Hu, Y. et al. (Apr. 25, 2018). "Neural control of sweat secretion: a review". In: *British Journal of Dermatology* 178.6, pp. 1246–1256. ISSN: 0007-0963. DOI: 10.1111/bjd.15808.

Hvitfeldt, Emil and Julia Silge (May 4, 2020). *Package 'textdata'*. CRAN. 25 pp. URL: https://github.com/EmilHvitfeldt/textdata.

Hyeon, Yuna, Young-Hwan Pan, and Hoon-Sik Yoo (Sept. 30, 2019). "Analysis of Users' Emotions on Lighting Effect of Artificial Intelligence Devices". In: *Korean Society for Emotion and Sensibility* 22.3, pp. 35–46. ISSN: 1226-8593. DOI: 10.14695/kjsos.2018.22.3.35.

Hömke, Paul, Judith Holler, and Stephen C. Levinson (Dec. 12, 2018). "Eye blinks are perceived as communicative signals in human face-to-face interaction". In: *PLOS ONE* 13.12. Ed. by Nicholas D. Duran, e0208030. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0208030.

ITU Pictures (May 15, 2018). *Sophia, First Robot Citizen at the AI for Good Global Summit 2018*. Digital. URL: https://www.flickr.com/photos/itupictures/27254369347/.

Jack, RachaelăE., OliverăG.B. Garrod, and PhilippeăG. Schyns (Jan. 2014). "Dynamic Facial Expressions of Emotion Transmit an Evolving Hierarchy of Signals over Time". In: *Current Biology* 24.2, pp. 187–192. ISSN: 0960-9822. DOI: 10.1016/j.cub.2013.11.064.

Jaeger, Sara R. et al. (May 2019). "Valence, arousal and sentiment meanings of 33 facial emoji: Insights for the use of emoji in consumer research". In: *Food Research International* 119, pp. 895–907. DOI: 10.1016/j.foodres.2018.10.074.

Jeri, Petar et al. (Oct. 25, 2017). "The Effect of Emotions and Social Behavior on Performance in a Collaborative Serious Game Between Humans and Autonomous Robots". In: *International Journal of Social Robotics* 10.1, pp. 115–129. ISSN: 1875-4791. DOI: 10.1007/s12369-017-0437-4.

Johnson, Susan C. and Erica Ma (2005). "The Role of Agent Behavior in Mentalistic Attributions by Observers". In: *ROMAN 2005. IEEE International Workshop on Robot and Human Interactive Communication*. ROMAN 2005. IEEE International Workshop on Robot and Human Interactive Communication. Nashville, TN, USA: IEEE, pp. 723–728. ISBN: 978-0-7803-9274-8. DOI: 10.1109/ROMAN.2005.1513865.

Johnson, Susan C., Su-Jeong Ok, and Yuyan Luo (Sept. 2007). "The Attribution of Attention: 9-Month-Olds' Interpretation of Gaze as Goal-Directed Action". In: *Developmental Science* 10.5, pp. 530–537. ISSN: 1363-755X, 1467-7687. DOI: 10.1111/j.1467-7687.2007.00606.x.

Jokinen, Kristiina and Graham Wilcock (Aug. 28, 2013). "Multimodal Open-Domain Conversations with the Nao Robot". In: *Natural Interaction with Robots, Knowbots and Smartphones*, pp. 213–224. DOI: 10.1007/978-1-4614-8280-2_19.

Kaya, Naz and Helen H. Epps (2004). "Relationship between color and emotion: a study of college students". In: *College Student Journal*. Vol. 38. 3, pp. 369–406.

Kendon, Adam (1967). "Some functions of gaze-direction in social interaction". In: *Acta Psychologica* 26, pp. 22–63. DOI: 10.1016/0001-6918(67)90005-4.

Kidd, C.D. and C. Breazeal (Sept. 2008). "Robots at Home: Understanding Long-Term Human-Robot Interaction". In: IEEE, pp. 3230–3235. ISBN: 978-1-4244-2057-5 978-1-4244-2058-2. DOI: 10.1109/IROS.2008.4651113.

Kim, Jejoong, Yur Kim, and Eunu Jo (Sept. 30, 2020). "Effect of Color and Emotional Context on Processing Emotional Information of Biological Motion". In: *Korean Society for Emotion and Sensibility* 23.3, pp. 63–78. ISSN: 1226-8593. DOI: 10.14695/kjsos.2020.23.3.63.

Kishi, T. et al. (Oct. 2012). "Development of Expressive Robotic Head for Bipedal Humanoid Robot". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2012). Vilamoura-Algarve, Portugal: IEEE, pp. 4584–4589. ISBN: 978-1-4673-1736-8 978-1-4673-1737-5 978-1-4673-1735-1. DOI: 10.1109/IROS.2012.6386050.

Knuth, Donald Ervin (1998). *The Art of Computer Programming. Seminumerical algorithms*. Vol. 2. 3rd ed. Addison-Wesley. 767 pp. ISBN: 978-0-201-89684-8.

KODANSHA (2021). ブルーピリオド. URL: https://kc.kodansha.co.jp/product?item=0000052420 (visited on 10/02/2021).

Kossmann, Charles E. (Dec. 1953). "The Normal Electrocardiogram". In: *Circulation* 8.6, pp. 920–936. ISSN: 0009-7322. DOI: 10.1161/01.cir.8.6.920.

Krauss, Robert M., Yihsiu Chen, and Purnima Chawla (1996). *Nonverbal Behavior and Nonverbal Communication: What Do Conversational Hand Gestures Tell Us?* DOI: 10.1016/S0065-2601(08)60241-5.

Krosnick, Jon A. and Duane F. Alwin (1987). “An Evaluation of a Cognitive Theory of Response-Order Effects in Survey Measurement”. In: *Public Opinion Quarterly* 51.2, pp. 201–219. DOI: 10.1086/269029.

Kuhn, Gustav, Benjamin W. Tatler, and Geoff G. Cole (Aug. 2009). “You look where I look! Effect of gaze cues on overt and covert attention in misdirection”. In: *Visual Cognition* 17.6-7, pp. 925–944. ISSN: 1350-6285. DOI: 10.1080/13506280902826775.

Langton, Stephen R. H., Roger J. Watt, and Vicki Bruce (2000). “Cues to the Direction of Social Attention”. In: *Trends in Cognitive Sciences* 4.2, pp. 50–59.

Lee, Jeongsoo, Hyeonbo Yang, and Donghoon Lee (June 30, 2019). “Context Modulation Effect by Affective Words Influencing on the Judgment of Facial Emotio”. In: *Korean Society for Emotion and Sensibility* 22.2, pp. 37–48. ISSN: 1226-8593. DOI: 10.14695/kjsos.2018.22.2.37.

Lee, Kang Woo and Jeong-Hoon Hwang (2008). “Human–Robot Interaction as a Cooperative Game”. In: *Lecture Notes in Electrical Engineering*, pp. 91–103. DOI: 10.1007/978-0-387-74935-8_6.

Lee, Seunghee, Akira Harada, and Pieter Jan Stappers (2002). “Design Based on Kansei”. In: *Pleasure with Products: Beyond Usability*. Ed. by William S. Green and Patrick W. Jordan. Taylor & Francis, pp. 212–222.

Lee, Seunghee, Toshikazu Kato, and Akira Harada (1997). “Kansei Evaluation of Subjective Image by Iconic Abstraction”. In: *2nd Asian Design Conference*, pp. 127–135.

Lee, Seunghee and Pieter Jan Stappers (1999). “Extending of Design approach based on Kansei by Dynamic Manipulation of 3D Objects”. In: *Bulletin of 4th Asian Design Conference*, pp. 686–693.

Lee, Seunghee, Pieter Jan Stappers, and Akira Harada (2000). "Kansei Appreciation of Observing 3D Objects". In: *Proceedings of XVI Congress of the International Association of Empirical Aesthetics 2000*, pp. 83–84.

Lenth, Russell V. (2016). "Least-Squares Means: TheRPackagelsmeans". In: *Journal of Statistical Software* 69.1, pp. 1–33. ISSN: 1548-7660. DOI: 10.18637/jss.v069.i01.

Levenson, Robert W. (Jan. 24, 2006). "Blood, Sweat, and Fears". In: *Annals of the New York Academy of Sciences* 1000.1, pp. 348–366. ISSN: 0077-8923. DOI: 10.1196/annals.1280.016.

Levinson, Stephen C. (Jan. 2016). "Turn-taking in Human Communication – Origins and Implications for Language Processing". In: *Trends in Cognitive Sciences* 20.1, pp. 6–14. ISSN: 1364-6613. DOI: 10.1016/j.tics.2015.10.010.

Löffler, Diana, Nina Schmidt, and Robert Tscharn (2018). "Multimodal Expression of Artificial Emotion in Social Robots Using Color, Motion and Sound". In: *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction - HRI '18*. ACM, pp. 334–343. DOI: 10.1145/3171221.3171261.

Magezi, David A. (Jan. 22, 2015). "Linear mixed-effects models for within-participant psychology experiments: an introductory tutorial and free, graphical user interface (LMMgui)". In: *Frontiers in Psychology* 6. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2015.00002.

Makowski, Dominique et al. (Feb. 2, 2021). "NeuroKit2: A Python toolbox for neurophysiological signal processing". In: *Behavior Research Methods* 53.4, pp. 1689–1696. ISSN: 1554-3528. DOI: 10.3758/s13428-020-01516-y.

Marku, Nenad et al. (Aug. 19, 2014). *Object Detection with Pixel Intensity Comparisons Organized in Decision Trees*. arXiv: 1305.4537 [cs.CV].

Mathias, Charles W and Matthew S Stanford (July 2003). "Impulsiveness and arousal: heart rate under conditions of rest and challenge in healthy males". In: *Personality and Individual Differences* 35.2, pp. 355–371. ISSN: 0191-8869. DOI: 10.1016/s0191-8869(02)00195-2.

Maxim Integrated (Feb. 13, 2020). *MAX30102 High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health*. URL: https://www.maximintegrated.com/en/products/interface/sensor-interface/MAX30102.html (visited on 05/23/2021).

Mohammad, Saif (2018). "Obtaining Reliable Human Ratings of Valence, Arousal, and Dominance for 20,000 English Words". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pp. 174–184. DOI: 10.18653/v1/p18-1017.

Mohammad, Saif M. (Dec. 9, 2020). *Practical and Ethical Considerations in the Effective use of Emotion and Sentiment Lexicons*. arXiv: 2011.03492 [cs.CL].

Mori, Masahiro, Karl MacDorman, and Norri Kageki (June 2012). "The Uncanny Valley [From the Field]". In: *IEEE Robotics & Automation Magazine* 19.2, pp. 98–100. ISSN: 1070-9932. DOI: 10.1109/MRA.2012.2192811.

Morris, Robert R., Mira Dontcheva, and Elizabeth M. Gerber (Sept. 2012). "Priming for Better Performance in Microtask Crowdsourcing Environments". In: *IEEE Internet Computing* 16.5, pp. 13–19. ISSN: 1089-7801. DOI: 10.1109/mic.2012.68.

Nass, Clifford and Youngme Moon (Jan. 2000). "Machines and Mindlessness: Social Responses to Computers". In: *Journal of Social Issues* 56.1, pp. 81–103. DOI: 10.1111/0022-4537.00153.

Noguchi, Yohei, Hiroko Kamide, and Fumihide Tanaka (Feb. 25, 2018). "Effects on Self-disclosure of Elderly Speakers by a Robot which Supports

Remote Communication". ja. In: *The Transactions of Human Interface Society* 20.1. ISSN: 1344-7262. DOI: 10.11184/his.20.1_67.

OmniVision Technologies Inc. (Jan. 8, 2009). *OV5642 datasheet*. datasheet. URL: http://www.uctronics.com/download/cam_module/OV5642DS.pdf (visited on 10/02/2021).

Onchi, Eiji (Aug. 13, 2020). *servo*. Computer software. URL: https://github.com/cgxeiji/servo.

Onchi, Eiji, Natanya Cornet, and SeungHee Lee (2021). "Effects of LED on Emotion-Like Feedback of a Single-Eyed Spherical Robot". In: *Science of Emotion and Sensibility* 24.3. DOI: 10.14695/KJSOS.2021.24.3.109.

Onchi, Eiji and Seung Hee Lee (2019). "Design and Evaluation of a Spherical Robot with Emotion-Like Feedback during Human-Robot Training". In: *Transactions of Japan Society of Kansei Engineering* 19.1, pp. 105–116. ISSN: 1884-0833. DOI: 10.5057/jjske.tjske-d-19-00036.

Onchi, Eiji, Daniel Saakes, and Seung Hee Lee (2020). "Emotional Meaning of Eyelid Positions on a One-Eyed 2D Avatar". In: *International Symposium on Affective Science and Engineering* ISASE2020.0, pp. 1–4. ISSN: 2433-5428. DOI: 10.5057//isase.2020-c000016.

Orozco, Lourdes and Jennifer Parker-Starbuck, eds. (2015). *Performing Animality*. 1. London: Palgrave Macmillan, p. 238. ISBN: 978-1-349-47646-6. DOI: 10.1057/9781137373137.

Paetzel, Maike, Giulia Perugia, and Ginevra Castellano (Mar. 6, 2020). "The Persistence of First Impressions". In: *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*. ACM, pp. 73–82. DOI: 10.1145/3319502.3374786.

Pakarinen, Tomppa, Julia Pietila, and Hannu Nieminen (July 2019). "Prediction of Self-Perceived Stress and Arousal Based on Electrodermal Activity". In: *2019 41st Annual International Conference of the IEEE Engineering* *in Medicine and Biology Society (EMBC)*. IEEE. DOI: 10.1109/embc.2019.8857621.

Pandey, Amit Kumar and Rodolphe Gelin (Sept. 2018). "A Mass-Produced Sociable Humanoid Robot: Pepper: The First Machine of Its Kind". In: *IEEE Robotics and Automation Magazine* 25.3, pp. 40–48. ISSN: 1070-9932. DOI: 10.1109/mra.2018.2833157.

Payrato, Lluis (2009). "Non-verbal communication". In: *Key Notions for Pragmatics*. Ed. by Jef Verschueren and Jan-Ola Ostman. Amsterdam: John Benjamins, pp. 163–194. ISBN: 978 90 172 0778 4.

Pixar Wiki (2021). *EVE*. URL: https://pixar.fandom.com/wiki/EVE (visited on 10/02/2021).

Pololu (2021). *Pololu - LSM6DS33 3D Accelerometer and Gyro Carrier with Voltage Regulator*. URL: https://www.pololu.com/product/2736 (visited on 09/14/2021).

Posner, Jonathan, James A. Russell, and Bradley S. Peterson (Sept. 2005). "The circumplex model of affect: An integrative approach to affective neuroscience, cognitive development, and psychopathology". In: *Development and Psychopathology* 17.03. DOI: 10.1017/s0954579405050340.

Raspberry Pi Foundation (May 2017). *Raspberry Pi Zero W - Pi-Zero-w-Tilt-1-1620x1080*. Photo. URL: https://www.raspberrypi.org/app/uploads/2017/05/Pi-Zero-W-Tilt-1-1620x1080.jpg (visited on 09/25/2021).

— (2018). *Raspberry Pi Zero W - Raspberry Pi*. URL: https://www.raspberrypi.org/products/raspberry-pi-zero-w/ (visited on 11/19/2020).

Rethink Robotics (2021). *Sawyer, the high performance collaborative robot | Rethink Robotics*. URL: https://www.rethinkrobotics.com/sawyer (visited on 09/25/2021).

Russell, James A. (1980). "A circumplex model of affect". In: *Journal of Personality and Social Psychology* 39.6, pp. 1161–1178. DOI: 10.1037/h0077714.

Russell, James A. and Merry Bullock (Sept. 1986). "Fuzzy Concepts and the Perception of Emotion in Facial Expressions". In: *Social Cognition* 4.3, pp. 309–341. ISSN: 0278-016X. DOI: 10.1521/soco.1986.4.3.309.

Scholl, Brian J and Patrice D Tremoulet (Aug. 2000). "Perceptual causality and animacy". In: *Trends in Cognitive Sciences* 4.8, pp. 299–309. ISSN: 1364-6613. DOI: 10.1016/s1364-6613(00)01506-0.

Seeed Technology Inc. (July 31, 2014). *Grove - GSR Sensor - Seeed Wiki*. URL: https://wiki.seeedstudio.com/Grove-GSR_Sensor/ (visited on 08/30/2021).

Seif El-Nasr, Magy et al. (Apr. 10, 2010). "Understanding and evaluating cooperative games". In: *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*. ACM Press, pp. 253–262. DOI: 10.1145/1753326.1753363.

Shannon, C.E. (Jan. 1949). "Communication in the Presence of Noise". In: *Proceedings of the IRE* 37.1, pp. 10–21. ISSN: 0096-8390. DOI: 10.1109/jrproc.1949.232969.

Shih, Margaret, Todd L. Pittinsky, and Nalini Ambady (Jan. 1999). "Stereotype Susceptibility: Identity Salience and Shifts in Quantitative Performance". In: *Psychological Science* 10.1. DOI: 10.1111/1467-9280.00111.

Simo, Endre (May 9, 2020). *pigo*. v1.4.2. Computer software. URL: https://github.com/esimov/pigo.

Society for Psychotherapy Research (2016). *Downloads | Working Alliance Inventory*. URL: https://wai.profhorvath.com/downloads (visited on 04/25/2021).

SoftBank Robotics (2021). *Pepper the humanoid and programmable robot | SoftBank Robotics*. URL: https://www.softbankrobotics.com/emea/en/pepper (visited on 09/14/2021).

Song, Sichao and Seiji Yamada (2017). "Expressing Emotions through Color, Sound, and Vibration with an Appearance-Constrained Social Robot". In:

Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction - HRI '17. ACM, pp. 2–11. DOI: 10.1145/2909824.3020239.

Spodick, David H. et al. (May 1992). "Operational definition of normal sinus heart rate". In: *The American Journal of Cardiology* 69.14, pp. 1245–1246. ISSN: 0002-9149. DOI: 10.1016/0002-9149(92)90947-w.

Stimson, Mike (Nov. 16, 2016). *Raspberry Pi Zero (Reduced Schematics)*. Schematic. URL: https://www.electronicsdatasheets.com/datasheet/RPI-ZERO-V1_3_reduced.pdf (visited on 09/25/2021).

Stroessner, Steven J. and Jonathan Benitez (Nov. 8, 2018). "The Social Perception of Humanoid and Non-Humanoid Robots: Effects of Gendered and Machinelike Features". In: *International Journal of Social Robotics* 11.2, pp. 305–315. ISSN: 1875-4791. DOI: 10.1007/s12369-018-0502-7.

Sumioka, Hidenobu et al. (Mar. 2013). "Design of Human Likeness in HRI from Uncanny Valley to Minimal Design". In: *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. 2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI). Tokyo, Japan: IEEE, pp. 433–434. ISBN: 978-1-4673-3101-2 978-1-4673-3099-2 978-1-4673-3100-5. DOI: 10.1109/HRI.2013.6483633.

Supcik, Jacques et al. (Dec. 24, 2020). *rpi-ws281x-go*. Computer software. URL: https://github.com/rpi-ws281x/rpi-ws281x-go.

Tapus, Adriana et al. (Dec. 19, 2012). "Children with autism social engagement in interaction with Nao, an imitative robot". In: *Interaction Studies* 13.3, pp. 315–347. ISSN: 1572-0373. DOI: 10.1075/is.13.3.01tap.

Tavakol, Mohsen and Reg Dennick (June 27, 2011). "Making sense of Cronbach's alpha". In: *International Journal of Medical Education* 2, pp. 53–55. ISSN: 2042-6372. DOI: 10.5116/ijme.4dfb.8dfd.

Terada, Kazunori, Chikara Takeuchi, and Akita Ito (Aug. 2013). "Effect of Emotional Expression in Simple Line Drawings of a Face on Human Economic Behavior". In: *2013 IEEE RO-MAN*. 2013 IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN). Gyeongju: IEEE, pp. 51–56. ISBN: 978-1-4799-0509-6. DOI: 10.1109/ROMAN.2013.6628530.

Terada, Kazunori, Atsushi Yamauchi, and Akira Ito (Sept. 2012). "Artificial emotion expression for a robot by dynamic color change". In: *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*. IEEE. ISBN: 9781467346061. DOI: 10.1109/roman.2012.6343772.

The Periph Authors (2021). *periph: Peripherals I/O in Go*. URL: https://periph.io/ (visited on 09/25/2021).

Thomas, Frank and Ollie Johnston (1981). *The Illusion of Life: Disney Animation*. 1st Hyperion ed. United States: Abbeville Press. 576 pp. ISBN: 0-7868-6070-7.

Thomaz, Andrea L. and Cynthia Breazeal (Apr. 2008). "Teachable Robots: Understanding Human Teaching Behavior to Build More Effective Robot Learners". In: *Artificial Intelligence* 172.6-7, pp. 716–737. ISSN: 00043702. DOI: 10.1016/j.artint.2007.09.009.

Torq Pro and Tower Pro (2014). *MG996R | Tower Pro*. URL: http://www.towerpro.com.tw/product/mg996r/ (visited on 10/02/2021).

Tracey, Terence J. and Anna M. Kokotovic (1989). "Factor structure of the Working Alliance Inventory." In: *Psychological Assessment* 1.3, pp. 207–210. DOI: 10.1037/1040-3590.1.3.207.

ubahnverleih (July 3, 2016). *Nao Robot (Robocup 2016)*. Digital. URL: https://upload.wikimedia.org/wikipedia/commons/thumb/4/47/Nao_Robot_%28Robocup_2016%29.jpg/800px-Nao_Robot_%28Robocup_2016%29.jpg.

University of Tsukuba (2021). 新型コロナウイルス感染症への対応（まとめ）- 筑波大学. Japanese. (Response to the Novel Coronavirus Infections (Summary) - University of Tsukuba). URL: https://www.tsukuba.ac.jp/about/antidisaster-crisismanagement/covid-19/ (visited on 08/12/2021).

Vatikiotis-Bateson, Eric et al. (Sept. 1998). "Eye Movement of Perceivers during Audiovisualspeech Perception". In: *Perception & Psychophysics* 60.6, pp. 926–940. ISSN: 0031-5117, 1532-5962. DOI: 10.3758/BF03211929.

Wainer, Joshua et al. (Sept. 11, 2013). "A Pilot Study with a Novel Setup for Collaborative Play of the Humanoid Robot KASPAR with Children with Autism". In: *International Journal of Social Robotics* 6.1, pp. 45–65. ISSN: 1875-4791. DOI: 10.1007/s12369-013-0195-x.

Wilms, Lisa and Daniel Oberfeld (2018). "Color and emotion: effects of hue, saturation, and brightness". In: *Psychological Research* 82.5, pp. 896–914. ISSN: 0340-0727. DOI: 10.1007/s00426-017-0880-8.

Wilson, Jason R. et al. (Sept. 14, 2019). *Developing Computational Models of Social Assistance to Guide Socially Assistive Robots*. arXiv: 1909.06510 [cs.HC].

Ypma, Jelmer et al. (Feb. 29, 2020). *The NLopt nonlinear-optimization package*. R Package. URL: https://cran.r-project.org/web/packages/nloptr/index.html.