Is human-in-the-loop reinforcement learning enhanced if the robot emotes its learning progress? An experimental study

F.C.J. Lycklama à Nijeholt

Delft University of Technology

TU Delft

INTERACTIVE ROBOTICS

Is human-in-the-loop reinforcement learning enhanced if the robot emotes its learning progress? An experimental study

by

F.C.J. Lycklama à Nijeholt

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday June 8th, 2023 at 9:00.

Student number:	4378105	
Thesis committee:	Dr. ir. D.J. Broekens,	LIACS, daily supervisor
	Dr. ir. J.C.F. de Winter,	TU Delft, supervisor
	Dr. D. Dodou	Committee member

Cover:	`www.aldebaran.com/sites/default/files/inline-images/nao-photo5-full.jpg` (Modified)
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

INTERACTIVE ROBOTICS

Summary

As technology continues to evolve at a rapid pace, robots are becoming an increasingly common sight in our daily lives. Robots that work with humans need to adapt to a variety of users and tasks, and learn to optimise their behaviour. For non-specialist users to interact with such robots, the robot's learning process needs to be transparent through its behaviour. Reinforcement Learning (RL) is a promising learning method to achieve this adaptability. However, the behaviour generated by RL is not inherently transparent because of the exploration/exploitation trade-off that is needed to optimise a policy for a specific task.

A RL algorithm is Temporal Difference (TD) learning. In TD learning, the algorithm updates a Q-table to keep track of Q-values. Q-values represent the expected future rewards that the agent (the actor that decides what action to take) can receive by taking a specific action in a certain state. Calculating the Q-values involves a value called the Temporal Difference, which is the difference between the current Q-value with the received reward added and the Q-value for the future state and chosen action.

Emotions are a natural way of communicating intent and situational appraisal for humans. In this study, emotional expressions based on Temporal Differences were implemented as a means to increase the transparency of a robot's learning progress. The effects on the robot's learning progress, learning result, and user experience were analysed.

A between-subject experiment with 61 participants on the following three robot modes was performed: no emotions, simulated emotions, and simulated emotions with matching attribution (see Table 1.1). The simulated emotions are hope, fear, joy, and distress, which were expressed by a humanoid robot. The robot mode with simulated emotions and matching attributions would explain for what task it was feeling hope or fear. The task was a simple task where a human teacher had to help a humanoid robot to learn to express three different colours based on human commands.

The results demonstrate minimal differences between these three conditions. This means that for simple tasks, emotional expressions grounded in RL do not have a significant effect, and thus do not help nor hurt. The findings are discussed, and it is proposed that emotion simulation is beneficial for tasks that are more complex, afford some robot autonomy, and for which the emotion is informative about how the user should influence the robot's actions to the benefit of the robot's policy.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
ANOVA	Analysis of Variance
HRI	Human Robot Interaction
MANOVA	Multivariate Analysis of Variance
MDP	Markov Decision Process
NAO	Name of the humanoid robot used in this research
RL	Reinforcement Learning
SARSA	State-Action-Reward-State-Action
SD	Standard Deviation
TD	Temporal Difference
TDRL	Temporal Difference Reinforcement Learning
UEQ	User Experience Questionnaire

Definitions

Concept	Definition
Agent	The actor that decides what action to take
Deterministic	A process or system that is completely predictable and has a fixed outcome
Exploitation actions	Performing the currently most optimal action
Exploration actions	Performing an action to search for an optimal action
Policy	The decision making of an agent in an environment
Q-value	The expected future rewards that the agent can receive by taking a specific action in a certain state
Reinforcement Learning	Machine learning algorithm where the agent learn to optimise its actions to maximize a reward signal
Stochastic	A process or system that involves randomness or probability
Temporal Difference	The difference between the received reward combined with the current Q-value and the estimated future Q-value
Transparency	The ability of users to understand what the intelligent systems are doing and why

Robot modes

Mode	Characteristic	Example
Mode 1	Neutral	[-]
Mode 2	Simulated expressions	Fear
Mode 3	Simulated expressions with attributions	Fear for the colour red

1 Introduction

1.1. Motivation and problem statement

Transparency of behaviour is important for intelligent systems that work with humans [1], especially when these systems become increasingly more complex. The term transparency refers to the ability of users to understand what the intelligent systems are doing and why. This is no different for robots that work with humans. Transparency can help users better understand the reasoning behind the robot's behaviour, enabling them to better assess the robot's capabilities [2]. Transparency also reduces conflict and improves the robustness of an interaction, particularly in team performance between robot and human [3].

If robots need to adapt to a variety of users and tasks, they need to learn to optimise their behaviour. When a robot is able to learn to optimise its behaviour the robot will be more efficient, flexible, and robust. It is more flexible as it can find optimal behaviour for different and changing environments. Due to this flexibility, the robot will also be more robust, as it can perform in the presence of (unexpected) variations.

Reinforcement Learning (RL) is a promising learning method for this purpose [4]. By repeatedly interacting with the environment, an RL agent (the actor that decides what action to take) learns to adapt the values of actions to achieve the optimal state transition policy, which decides what actions to perform to go to the next state, thus maximizing the rewards over time.

However, the behaviour generated by RL is not inherently transparent due to the exploration/exploitation tradeoff that is needed to optimise a policy for a specific task [5]. During exploration, the robot will perform actions that are not the best known actions at that moment in the learning process, which might be confusing for the users. During premature exploitation, before the best action has been found, the agent will select actions that are suboptimal, again potentially confusing the user.

Researchers [6, 7] state that it is important for intelligent systems to remain transparent to the users, especially when the systems become increasingly more complex. With this self-explanatory capability, users understand what these intelligent systems are doing during the Human Robot Interaction (HRI) and why.

Studies [8, 9, 10] have shown the focus should be on creating transparency in order to generate trust between humans and robots, instead of making the (automation) capabilities as high as possible. This can be done, for example, by mutual communication and enabling vulnerable communication in casual and non-work-related interactions. Vulnerable communication is communication where thoughts, feelings, and emotions in an open manner are communicated, and non work-related interactions are shared experiences outside of the task at hand that create a shared understanding.

It has also been suggested that simulation and expression of emotions in robots can be used to create transparency [5]. Most emotion theories propose that emotions appear when a change in the situation has an impact on the agent. The expression of emotions is a language-independent way to communicate information about the current state of an individual to someone else. This communication method is something that comes naturally to humans. The Temporal Difference Reinforcement Learning (TDRL) theory of emotion [11] proposes that emotions are manifestations of reward processing in RL, in particular manifestations of neural Temporal Difference (TD) assessment, which is the mentally computed TD. This TD represents the agent's perception of gain or loss of utility (well-being), resulting from new state.

This suggests that agents and robots that use RL to learn can also simulate and express emotions grounded in their learning process. Indeed, research suggests that simulated emotions are plausible [12, 11, 13, 14]. However, experimental evidence that these emotions are understandable by and plausible for human teachers in a robot-human interaction setting is lacking.

1.2. Main objective

The goal of this research is to determine whether emotional expressions based on TD learning towards a human teacher can be used as a means to increase the transparency of a robot's learning process, and thereby impact said learning process. Specifically, the study examines the effect of robot emotional expressions and the explanation of the source (attribution) of the emotions on the teacher's behaviour and experience, and on the robot's learning result and learning process. An example for the emotional expressions and attributes is a robot expressing fear (the emotional expression) for the colour red (the source/attribution). Three robot modes were used: a robot mode that showed no emotions, a robot mode that showed simulated emotions, and a robot mode that showed simulated emotions with matching attribution. The simulated emotions were hope, fear, joy, and distress, which were expressed by a humanoid robot.

1.2.1. Research question

The main research question derived from the main objective is:

- What is the influence of introducing Temporal Difference-based emotions in a human-robot interaction with a human teacher?

With the following sub-questions:

a) What is the influence of introducing Temporal Difference-based emotions on the learning result?

This sub-question focuses on the final learning result of the robot. The learning behaviour of the robot is the same for any of robot modes. However, the human teacher might have a different approach to teaching the tasks if the robot shows emotions. For example, if the robot communicates to the teacher that it is afraid for a specific task, the teacher might decide to avoid that task, which will result in the robot not learning said task.

b) What is the influence of introducing Temporal Difference-based emotions on the learning process?

This sub-question focusses on the learning process of the robot and how the user influences this. During the learning process, the robot has the choice to perform exploration or exploitation actions to find the correct actions for each task. However, the robot has no influence on what task the human teacher will give. This sub-question focusses on the exploration and exploitation actions of the human teacher, as well as the progress of the q-table, with the main focus on for how many tasks a positive route has been found.

c) What is the influence of introducing Temporal Difference-based emotions on the user experience?

This sub-question focusses on how the participant experienced the interaction with the robot. This user experience will be assessed using a questionnaire given to the user after the experiment. The user experience will be evaluated based on the perceived animacy, anthropomorphism, attractiveness, efficiency, intelligence, likeability, novelty, and stimulation.

1.3. Research methods

The study primarily involves conducting an experiment with reinforcement learning using a human teacher. In this experiment, the human will receive a simple learning task that they will have to perform, which is teaching the robot the three primary colours: red, green, and blue.

The participant will perform this task with a robot with one of the robot modes that can be seen in Table 1.1.

Apart from this difference in emotion display, all participants will receive the same task and the same values will be measured during this experiment. After the interaction with the robot has been completed, the participant will be asked to answer the User Experience Questionnaire [15] and the Godspeed questionnaire [16].

A more detailed explanation of the research method can be found in chapter 3.

Mode	Characteristic	Example
Mode 1	Neutral	[-]
Mode 2	Simulated expressions	Fear
Mode 3	Simulated expressions with attributions	Fear for the colour red

Table 1.1: The three different robot modes, with examples

1.4. Report outline

In Chapter 2, the background knowledge that is relevant to reinforcement learning and emotions is reported. In Chapter 3 the experiment is explained. Here it can be found how the learning algorithm has been programmed to learn colours as well as how the emotions have been implemented, and what measurements have been performed during the experiment. In Chapter 4, the results of the tests are reported. In Chapter 5, the conclusions on the research questions provided by the knowledge reported in the previous chapters as well as a brief discussion can be found.

2
Background information

In this chapter the subject of Reinforcement Learning (RL) is introduced (2.1) and its applications. The specific RL method used in this study, Temporal Difference Reinforcement Learning (TDRL), is explained in Section 2.2 and the policies in are introduced in 2.3. In Section 2.4 the TDRL theory of Emotion, which is used to simulate the emotions is introduced and in Section 2.5 related literature on simulating emotions based on RL is listed.

2.1. Introduction

RL is an area of machine learning, in which an RL problem consists of an agent and an environment. The RL agent acts on the environment and receives a reward or a punishment which tells the agent what actions are beneficial and which ones are detrimental. The task for the RL agent is to maximise the total reward and thus find an optimal policy. In an RL problem, it is not necessary to program how a task needs to be achieved. The RL agent will need to perform either exploration actions to examine the environment or exploitation actions in which the RL agent tries to exploit the current knowledge to obtain a high reward [17]. The other components of an RL problem such as the two most used TD methods and policy will be discussed in this chapter.

2.2. Temporal Difference Method

With TDRL, the algorithm updates a Q-table to keep track of Q-values. Q-values represent the expected future rewards that the agent can receive by taking a specific action in a certain state. Calculating the Q-values involves the TD value, which is the difference between the current Q-value with the received reward added and the Q-value for the future state and chosen action. This TD is used to adjust predictions during the learning process.

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \overbrace{(r_t + \gamma \underbrace{Q(s_{t+1}, a)}_{\text{estimate of future value}} - \underbrace{Q(s_t, a_t))}_{\text{current value}}}^{\text{Temporal Difference}} \tag{2.1}$$

Here, Q refers to the function that is computed and is used to determine how good action a in state s is. s_t is the current state, a_t is the selected action, s_{t+1} is the new state that the system will enter after performing a_t, α is the learning rate which is a value between 0 and 1, r_t is the received reward at the current time. And γ is the discount factor, which is also a value between 0 and 1, and determines the importance of the estimate of the future value [18]. $Q(s_{t+1}, a)$ is the Q-value belonging to the future state and an action. How this future Q-value is calculated is the main difference between SARSA (State-Action-Reward-State-Action) and Q-learning [19], which are two algorithms that have been widely used to solve RL problems

For Q-learning, the estimate is based on the maximum of the available actions, whilst SARSA learns the Q values associated with the actions that it chooses. The Q-Learning algorithm is summarised in algorithm 1 and the SARSA algorithm is summarised in algorithm 2 [20].

The RL agent updates the policy based on actions taken so it is known as an on-policy learning algorithm.

Algorithm 1 Q-Learning

Initialize Q(s,a)
repeat
  Observe initial state s_1
  for $t = 1 : T$ **do**
    Select an action a_t using policy (e.g. ϵ-greedy) derived from Q
    Carry out action a_t
    Receive reward r_t and find new state s_{t+1}
    Update Q using

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma * max(Q(s_{t+1}, a)) - Q(s_t, a_t))$$

    $[s_t \leftarrow s_{t+1}$
  end for
until s_t is terminal state

Algorithm 2 SARSA

Initialize Q(s,a)
repeat
  Observe initial state s_1
  Select an action a_1 using policy (e.g. ϵ-greedy) derived from Q
  for $t = 1 : T$ **do**
    Carry out action a_t
    Receive reward r_t and find new state s_{t+1}
    Choose next action a_{t+1} using policy (e.g. ϵ-greedy, see Section 2.3.2) derived from Q
    Update Q using

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

    $[s_t \leftarrow s_{t+1}$
  end for
until s_t is terminal state

2.3. Policies

As mentioned in section 2.2, an RL agent can be in states and perform actions. When the RL agent is in a state, it needs to make a choice for which action it will take. This choice tends to be based on the Q-values and what policy is used. This policy needs to find a good balance between exploration and exploitation. Some actions are deterministic and depend only on the state and chosen action, and have a fixed outcome, and other are stochastic and depend on the state and chosen action but also have a randomness or a probability distribution.

Two commonly used policies are softmax and ϵ-greedy [21].

2.3.1. Softmax

Softmax uses a vector with K numbers. This vector is then normalised into a probability distribution

$$\sigma(\mathbf{z})i = \frac{e^{z_i}}{\sum j = 1^K e^{z_j}} \tag{2.2}$$

Here **z** is the vector with K numbers, z_i are the values of the elements of the input vector, which can only have real values.

The sum of this normalised distribution can be used as probabilities of the actions to happen. When a value is very small or negative, the likelihood of choosing the action corresponding to that value will be very small as well.

2.3.2. ϵ- Greedy

The ϵ-greedy algorithm shows a clear difference between the exploration and exploitation actions. There is a probability of ϵ that the algorithm will choose an exploration action and a probability of 1-ϵ that the algorithm chooses an exploitation action.

$$\text{Action} = \begin{cases} \text{max } Q_t(a) & \text{With probability 1-}\epsilon \\ \text{random action (a)} & \text{With probability } \epsilon \text{ .} \end{cases} \tag{2.3}$$

However, with the ϵ-greedy the algorithm will remain the same over time. The likelihood of selecting an exploration action will always be ϵ. To solve this, the ϵ-decay strategy exists, where the ϵ values decreases over time [19]. Due to this decrease, the algorithm will first mainly choose exploration actions and over time increasingly opt for the exploitation action.

2.4. Joy, Distress, Hope and Fear in the TDRL Theory of Emotion

The TDRL Theory of Emotion proposes that all emotions are manifestations of TD errors [11, 5]. Emotion is defined as *valenced appraisals in reaction to (mental) events providing feedback to modify action tendencies, grounded in primary reinforcers* [11].

In Q-learning, a method to learn action values $Q(s, a)$ based on repeated observations of states, actions and rewards, the TD is defined as:

$$TD = r + \gamma \max_{a'} Q(s', a') - Q(s, a)_{old} \tag{2.4}$$

In the TDRL Theory of Emotion, joy is proposed to be the manifestation of a positive TD, whilst distress is a negative TD. As such, Joy and Distress are defined as follows [11]:

$$if(TD > 0) \Rightarrow Joy = TD \tag{2.5}$$

$$if(TD < 0) \Rightarrow Distress = TD \tag{2.6}$$

Hope and fear are proposed in a similar manner to joy and distress. While joy and distress are proposed to be manifestations of the actual received TD, hope and fear are proposed to be a manifestation of the expected TD. Hope represents the anticipation (forward simulated) of a positive TD, whilst fear is the anticipation of a negative TD [11].

2.5. Related work on emotion simulation based on RL

As mentioned in chapter 1, RL-generated behaviour is not always inherently transparent due to the exploration/exploitation trade-off required to optimise policies [5]. During the exploration phase, or during an exploitation phase before the optimal solution has been found, the robot may take sub-optimal actions, which can be confusing for users.

Simulating and expressing emotions in a robot has been suggested to create transparency [5]. Expressing emotions is a natural, language-independent way to communicate information about an individual's current state to others.

Simulating emotions for RL agents and robots is not new. For example, Broekens et al.[12], propose a computational model of joy, distress, hope, and fear as mappings between values used in RL (reward, value, update signal, etc...). In this model, joy/distress is derived from the positive/negative TD signal for the current state, and hope/fear is derived from the expected TD for future states. This model is explained more in section 2.4. Later work using the same model showed, with a theoretical analysis, plausible simulations of fear and hope [22], and regret [14].

Moussa and Magnenat-Thalmann[23] included emotions, attachment (which represents whether an agent loves or hates a person) and learning in a decision-making Q-learning architecture for a virtual agent. Their framework was evaluated by interacting with users in different scenarios. Preliminary results showed that the virtual agent showed appropriate emotional responses to different user behaviours.

A recent study simulated emotions based on TD signals and presented participants with videos of the robot expressing these emotions [24]. The study was inconclusive with respect to the transparency gained from these emotional expressions. A later study [25] observed a slight increase in transparency but also proposed a larger scale study.

3 Method

During the experiment, participants received the objective to teach a robot a task. The focus of this research is on the implementation of the emotion simulation and not on implementing a complex learning algorithm. That is why the learning task was a simple task; teach the robot 3 different colours; red, green, and blue.

In this chapter the learning task is described in Section 3.1. In Section 3.2 the robot is introduced. In Section 3.3 the programmed behaviour is explained. In Section 3.4 the measurements are listed, and in the final Section, section 3.5 the performed experiment is described.

3.1. The learning task

For the modelling of the learning process, a simple environment had been set up. A visual overview of this environment can be seen in the Markov Decision Process (MDP) in Figure 3.1. This MDP consists of a set of states, a set of actions, and transitions with associated rewards. In the start state, the robot can only ask for a task. Then, depending on the task the robot received, it will either go to ”Task eyes red”, ”Task eyes green”, or ”Task eyes blue”, the colour referring to the colour the teacher said the eyes should be changed to.

In the states ”Task eyes red”, ”Task eyes green”, and ”Task eyes blue”, the algorithm will have three actions that it can perform, namely ”Make eyes red”, ”Make eyes green”, and ”Make eyes blue”. So, in total, the environment consisted of seven states and four possible actions.
States:

- Start
- Task eyes red
- Task eyes green
- Task eyes blue
- Eyes are red
- Eyes are green
- Eyes are blue

Actions:

- Ask for task
- Make eyes red
- Make eyes green
- Make eyes blue

The decision about which action to perform will be made using the ϵ-greedy variant ϵ-decay. This is because of the high levels of exploration at the beginning and high exploitation later in the experiment. The robot will maintain a Q-table to track the known routes and rewards. At the start of each experiment, the Q-table will be populated with zeroes only. During the learning process, the robot will continue to repeat the steps outlined in Algorithm 2 in Section 2.2.

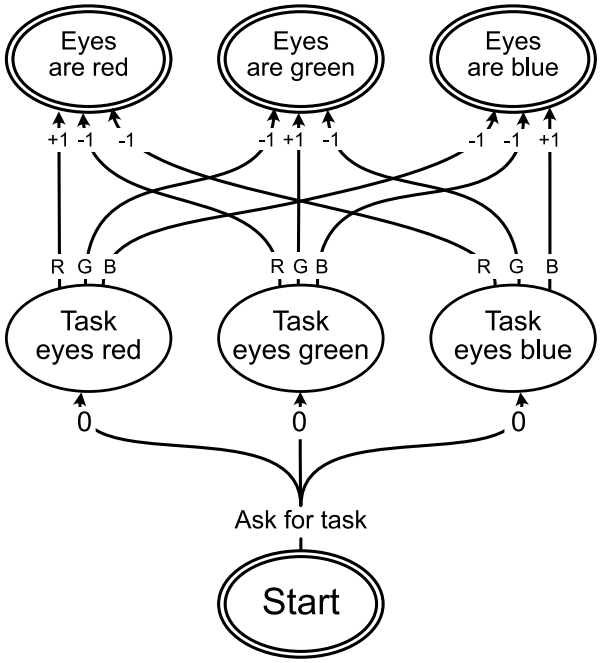


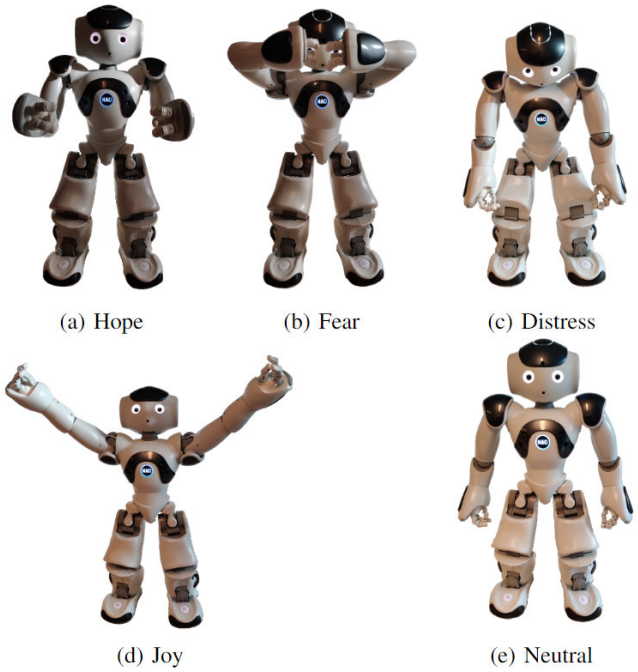
Figure 3.1: *A visualisation of the MDP for the learning process of the robot. In the ellipses the seven states can be seen. At the beginning of the arrow, the action can be seen and at the end of the arrow, the matching reward can be seen. The actions to change the eyes to a colour are indicated by the first letter of that colour.*

3.2. The robot

During this research, a robot named NAO will be used. Nao is a physically embodied robot of 58cm height, with a lot of built-in features. The ones that are used during this research are:

- The ability to produce human-like gestures by rotating its joints
- The ability to communicate with the participants via its microphones and speakers
- Speech recognition
- The ability to change the colour of the light around its eyes

The NAO robot is equipped with the ”Robot in the Classroom” software. But as the reaction of the robot with that software appeared to be quite slow, the main parts were programmed via Python. Pictures of NAO can be seen in Figure 3.2.



(a) Hope (b) Fear (c) Distress

(d) Joy (e) Neutral

Figure 3.2: The NAO in the poses for the fou emotions and its neutral pose.

3.3. Behaviour implementation

SARSA learns from the actions it actually performs, instead of learning from the best possible action, independent of the action performed (see Section 2.2). Therefore SARSA is a more realistic simulation of the learning process of a human, and thus SARSA was choosen to be more suited for this experiment. The algorithm for this can be seen in Section 2.2, in Algorithm 2. The general function for updating the values in the Q-table can be seen in eq. 2.2. The exact function for SARSA is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \overbrace{(r_t + \gamma * Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))}^{\text{Temporal Difference}} \tag{3.1}$$

For the policy, the ϵ-decay strategy has been chosen, which is explained in Section 2.3.2. During the learning process, the ϵ-value starts at 0.7 and decays by multiplying it with a factor of 0.85, and has a minimum factor of 0.1. The algorithm will start with a somewhat high exploration ratio, but as the experiment continues, the algorithm will increasingly choose the most efficient action. With the decay value of 0.85, the ϵ-value will reach a value of 0.1 after 12 multiplications. The high exploitation ratio and the end will give the participant the feeling that the robot has indeed successfully learnt the colours.

The value of $alpha$ has been set at 0.5 and the value for γ at 0.9 for this experiment.

With a high value for $alpha$, the learning rate, the Q-values will be updated rapidly, which also means that the TD will quickly reach 0. A value of 0.5 was found to be a good balance for the decrease of TD.

With a high value for γ, the discount factor, the algorithm places a high emphasis on future rewards. As the states ”Eyes are red”, ”Eyes are green”, and ”Eyes are blue” are terminal states, the Q-values for these states will remain 0. This means that the discount factor will not have an influence on the states ”Task eyes red”, ”Task eyes green”, and ”Task eyes blue” and only on the ”start” state. The immediate rewards for actions chosen from the ”start” state only have values of 0. The high emphasis on future rewards allows the Q-values to propagate more efficiently to the start state.

The robot emotions were based on the TD value in Equation 3.1. Joy and distress are based on the actually received TD. In Section 2.4, the threshold for the emotions was set at 0. To be able to easily convince people to participate, the goal was to ensure that the experiment would take no longer than 15 minutes. To ensure that the disappearance of the emotion would still be observed, this threshold was increased to 0.2. This resulted in the following thresholds for the activation of joy and distress:

$$if(TD > 0.2) \Rightarrow Joy \tag{3.2}$$

$$if(TD < -0.2) \Rightarrow Distress \tag{3.3}$$

Hope and fear are based on the forward simulated TDs in the MDP, using a $epsillon$ greedy simulation policy for the robot. TDs are calculated for the $epsillon$ greedy actions in the *Task* states (see Figure 3.1). At the start state, the maximum absolute value of the three simulated TDs for the three possible stochastic outcomes (human picks red, green or blue) is selected and that TD is used to express emotion. If that TD is positive, the emotion is hope, and if that TD is negative, the emotion is fear. Only those transitions that were actually observed are used. The calculations for hope and fear are:

$$\begin{aligned} &TD = signed(\text{max}(|TD_{a,s \in Taskstates}(argmax(Q(s,a)))|)) \\ &if(TD > 0.2) \Rightarrow Hope \\ &if(TD < -0.2) \Rightarrow Fear \end{aligned} \tag{3.4}$$

Joy and distress were expressed immediately after receiving the TD update for an action (i.e., after the transition to a new state). Hope and fear were expressed in the state before the robot performed its action. Multiple emotions could be expressed in a row. For example, upon arriving at a task state, the robot could express distress for a negative TD it received because the user chose a colour that it had not learned, followed by fear in the state for the possible wrong choice it may make. As an emotion is either on or off, a small threshold was introduced for the emotion elicitation, so that the robot would only express an emotion when a significant change in the TD occurred. This was done to stop the robot from expressing emotions when converging on the learning task.

The NAO has the ability to perform human-like movements but lacks visual expression. Due to this, the main focus for expressing emotions with the NAO is through the use of body language. The expression of emotions such as happiness, sadness, joy, and fear can be seen in figure 3.2.

The poses for the emotions sadness, happiness, and fear were inspired by research by Thoma et al. [26] and Wu et al. [27]. These articles provide examples of poses for six basic emotions, including anger, disgust, fear, happiness, sadness, and surprise. However, hope is not considered a basic emotion. Therefore, the pose for hope was determined by testing different poses and asking volunteers how they would label the pose.

After implementing these body expressions, a small pilot test was conducted to determine whether participants could recognise all the expressions. It was concluded that the expressions could be more recognisable by adding statements. The verbal expressions used for the robot in Dutch and English can be seen in Tables 3.1 and 3.2, respectively. During the interaction, the algorithm randomly chose one of the three expressions to avoid repeating statements.

Emotion	Expression 1	Expression 2	Expression 3
Hope	Ik heb er zin in	Kom maar op!	Dat gaat wel weer goed komen.
Fear	Oei dit vind ik spannend.	O nee dit gaat vast niet.	O nee, dit gaat fout.
Joy	Hoera	Jippie	Wat fijn
Distress	Drommels	Helaas	Wat jammer
Neutral	Oké	Bedankt	Prima

Table 3.1: The statements in Dutch made by the robot for the different emotions and for the neutral mode without any emotions

Emotion	Expression 1	Expression 2	Expression 3
Hope	I am looking forward to this	Let's go!	Okay. Let's go
Fear	This is a bit scary for me	O no, it will go wrong again	Oh no, it will go wrong
Joy	Hooray	Nice	Lovely
Distress	O bother	Let's pretend that did not happen	How unfortunate
Neutral	Okay	Thank you	Check

Table 3.2: The statements in English made by the robot for the different emotions and for the neutral mode without any emotions

The robot mode with matching emotional attributions explained to the user, in addition to expressing the emotions, what colour it was feeling hope or fear for (precisely: what colour the TD that generated that emotion is associated with). Therefore, it explained the cause of the prospect-based emotions, the attribution of hope and fear. The robot would explain to the user why it is feeling hope or fear at the start.

If the robot were to simulate hope, it would add the statement ”Ik hoop dat het colour wordt!” or ”I am hoping for colour!” after the hope expression.

After this algorithm was implemented, the robot behaviour underwent some iterations to make it as fast as possible and to fine-tune the speech-to-text behaviour of the robot. During one of the pilots between these iterations, it was decided to hardwire the actual reward to ensure that errors in the interpretation of the reward were not possible. This way, the learning progress would not be influenced if the robot were to hear the wrong statement after asking for a reward.

3.4. Measurements

A between-subject experiment was conducted, where each participant was randomly assigned to one of three different robot modes. which can be seen in table 1.1. Various variables were measured to examine the effect on experience, learning process, and learning outcome.

Apart from the measurements described in subsections 3.4.1, 3.4.2, and 3.4.3, the users were asked for their age, gender, experience with robots, and experience in programming. These data was used to investigate the spread of participants over the different groups.

3.4.1. Learning outcome

Learning progress and final outcome are measured based on the number of transitions to colour outcome states for which the Q-table has a maximum value above zero (robot has learned the correct action). The Q-table exhibits a value greater than zero when the correct colour has been performed at least once.

All possible values for the learning progress score are:

0. No colour learned
1. One colour learned
2. Two colours learned
3. Three colours learned

3.4.2. Learning process

To assess the learning process, the exploration and exploitation carried out by the human teacher were examined. Exploration was defined as assigning a task to the robot for which the correct max Q had not been found, while exploitation involved assigning a task for which the correct max Q had been found.

As the total number of iterations varied, the obtained scores were scaled based on the actual number of iterations performed. Consequently, a ratio for each variable was calculated by dividing the number of times it occurred by the total number of iterations.

We used the following variables to measure the learning process:

- Learning progress score over iterations
- The ratio of exploration commands given
- The ratio of exploitation commands given
- The ratio of inefficient exploitation commands (i.e. when the user selects a learned colour when there are unknown colours left)
- The ratios of selecting another command the next trial, depending on whether the previous command was correctly executed by the robot or not

3.4.3. User experience

After the experiment, the participants were asked to complete two questionnaires: the Godspeed Questionnaire [16] and the User Experience Questionnaire (UEQ) [28]. The Godspeed questionnaire was used to measure the degree of:

- Anthropomorphism: attributing human-like characteristics or behaviors to non-human entities (e.g., animals, objects, or even abstract concepts).
- Animacy: the perception of whether an entity is alive or not, based on its movement and behavior.
- Likeability: the degree to which an entity is pleasant, agreeable, or enjoyable.
- (Perceived) intelligence: the ability to acquire, understand, and apply knowledge and skills to solve problems or adapt to new situations.

The UEQ was used to determine the scores for:

- Novelty: the degree to which an entity is new, unusual, or unexpected.
- Stimulation: the degree to which an entity can capture and maintain someone's attention or interest.
- Efficiency: the degree to which an entity achieves its intended purpose with minimum wasted effort or resources.
- Attractiveness: the degree to which an entity is visually or aesthetically pleasing to someone's preferences.

The questions in the Godspeed questionnaire were initially organized by type - anthropomorphism, animacy, likeability, perceived intelligence, and perceived safety. For this experiment, the questions related to perceived safety were removed. Afterward, the remaining questions were sorted alphabetically instead of by type. This was done to make it less obvious which questions were measuring which specific factors. The final questionnaires can be found in Appendix A.

3.5. The experiment

We recruited 61 adult participants with a mean age of 30.64 (SD 13.147) to teach the robot three different colors (red, green, and blue). In the starting state (refer to Figure 3.1), the robot asked the participants which color to change its eyes to, and the participants responded with one of the three colors. The robot then chose one of three actions [Make eyes red, Make eyes green, Make eyes blue] and asked the participant if the color was correct, which the participant could confirm or deny. The actual reward was hardwired to eliminate any errors in reward interpretation, and no mismatch between user response and hardwired reward occurred.

The only influence that the participant had on the learning process was the order in which the tasks were assigned to the robot.

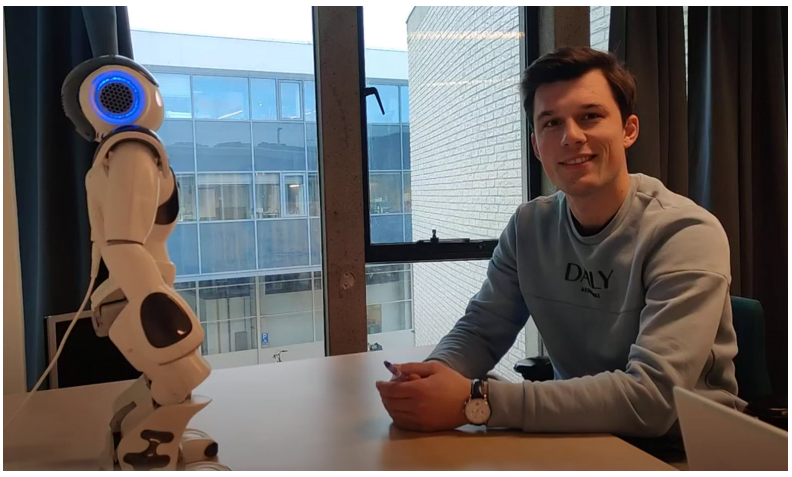
During the learning process, the robot used ϵ-greedy with ϵ-decay for action selection. Beforehand, the participants were informed that participation was voluntary and that they would have 10 minutes to teach the robot the three colours and they were assured that this was sufficient time to complete the task without feeling rushed. The participants were provided with a consent form before the beginning of the task, as well as the first page of the questionnaire, which summarized the consent form and asked the participant their age, gender, experience with robots, and experience in programming. The task ended after convergence.

After the learning task had been completed, the participants were asked to complete the UEQ and Godspeed questionnaires.

During the debriefing session, participants were informed about the distinctions in robot modes and the aim of the research, which was to investigate the impact of including simulated emotional expression. This was done after the experiment had been carried out and the questionnaires had been completed.

The study conducted a between-subject experiment, with each participant randomly assigned to one of three robot modes that can be seen in Table 1.1.

(a) A participant in Leiden with the NAO expressing fear



(b) A participant in Delft with the NAO in its neutral mode

Figure 3.3: Two participants interacting with the NAO.

4
Results

In Section 4.1 the reliability of the questionnaire results is checked. In Section 4.2 the results are presented. In subsection 4.2.1 the spread of participants over the three robot modes is investigated. In subsection 4.2.2 the Questionnaire results are presented and the effect of the different robot modes is investigated. Subsection 4.2.3 effects on the learning progress and learning result are researched. Subsection 4.2.4 describes the debriefing.

4.1. Reliability checks

To ensure the reliability of the data collected from the questionnaire, Cronbach's alpha [29] was calculated. A high value of Cronbach's alpha indicates that the items are highly correlated. To calculate Cronbach's alpha, the correlation between each item and the total score for this item is calculated, and the average of these correlations is used to estimate the reliability of the questionnaire. A Cronbach's alpha value below values 0.60 are generally regarded as poorly acceptable.

Due to their low consistency scores, the perspicuity and dependability scores from the UEQ were not used. The Cronbach's alpha for perspicuity was calculated to be 0.291 and for dependability, it was 0.494. On the other hand, the other UEQ scores had high Cronbach's alpha values, with 0.876 for attractiveness, 0.661 for efficiency, 0.733 for stimulation, and 0.628 for novelty. Similarly, Cronbach's alpha was calculated for Godspeed questionnaire with values of 0.791 for anthropomorphism, 0.637 for animacy, 0.847 for likeability, and 0.623 for perceived intelligence.

Score	Cronbach's Alpha
Animacy	.637
Anthropomorphism	.791
Attractiveness	.876
Dependability	.494
Efficiency	.661
Novelty	.628
Likeability	.847
Perceived intelligence	.623
Perspicuity	.291
Stimulation	.733

Table 4.1: This table displays the reliability of the questionnaire results measured by Chronbach's alpha. A Cronbach's alpha value lower than 0.60 is commonly considered as inadequate.

In summary, the study used Cronbach's alpha to verify the reliability of the questionnaire data and found that the perspicuity and dependability scores from the UEQ had low consistency. The study also included three conditions with similar participant demographics and found no significant difference in experience with robots and computer science among the conditions.

4.2. Results

After the reliability of the data had been investigated, an ANOVA (Analysis of Variance) [30] test had been performed. An ANOVA is a statistical test used to analyse whether there are significant differences between the means. An ANOVA is done by calculating the variance within and between groups, and comparing these variances to determine if they are significantly different.

ANOVA tests the null hypothesis that there are no significant differences between the means of the groups, and provides a p-value to determine whether this hypothesis should be rejected.

If the p-value obtained from the test is low (typically less than 0.05), the null hypothesis can be rejected. The low p-value indicates that the probability of obtaining the found result by chance, while the is no significant difference between the variance, is low.

A Multivariate Analysis of Variance (MANOVA) is similar to an ANOVA test, but it is used when considering multiple dependent variables simultaneously.

4.2.1. The participants

Each robot mode had different participants. If the groups were significantly different, that might influence the learning process, learning result, and user experience. To check for differences between the group, the participants had to enter their age, gender, experience with robots, and experience in programming.

For the ages, the mean value and the standard deviation (SD) have been calculated.

Age, gender, and the number of participants were the same across all three conditions. Mode 1 had 20 participants, mean age of 30.20 years (SD = 13.950) and consisted of 11 males, 8 females, and one other. Mode 2 had 20 participants, a mean age of 28.95 years (SD = 9.288), with 13 males and 7 females. Mode 3 had 21 participants, a mean age of 32.67 years (SD = 15.631) and 13 males and 8 females.

ANOVA was used to test whether the experience with robots and computer science differed significantly among the conditions, but it was found that there was no significant difference [F(2,58)=0.076, p=0.927].

	Post-Hoc between robot modes			ANOVA	Effect size Partial Eta Squared
Score	1-2	1-3	2-3		
Animacy	.027	.047	.786	.052	.097
Anthropomorphism	.586	.092	.250	.223	.050
Attractiveness	.053	.132	.639	.127	.069
Efficiency	.862	.370	.284	.512	.023
Likeability	.108	.205	.712	.238	.048
Novelty	.011	.296	.115	.037	.107
Perceived intelligence	.821	.890	.928	.974	.001
Stimulation	.172	.313	.705	.366	.034

Table 4.2: Results of Post Hoc LSD and ANOVA tests on the questionnaire answers for the comparison between robot modes. Yellow values indicate significant differences in only the LSD Post Hoc test, green values indicate significant differences in both tests. The effect size is indicated by the colour green for a medium effect and all other effect sizes are small.

4.2.2. Questionnaire results

To investigate the effect of the different robot modes on the user experience a MANOVA test has been conducted on the combined Godspeed Questionnaire and UEQ. Godspeed measured the user experience on perceived anthropomorphism, animacy, likeability, and intelligence. This MANOVA did not show any significant effects [F(8,108) = 1.325, $p = .239$]. The UEQ looked at the user experience on perceived novelty, stimulation, efficiency, and attractiveness. This MANOVA did not show any significant effects as well, [F(8,108) = 1.647, $p = .120$].

The p-values resulting from the univariate ANOVAs for each dependent variable can be seen in table 4.2, showing that only novelty has a significant [F(2, 58) = 3.485, $p = .037$, $\eta^2 = .107$] effect, with a near significant effect for animacy [F(2, 58) = 3.103, $p = .052$, , $\eta^2 = .097$].

The medium effect size was found for both novelty and animacy, with $\eta^2 = .107$ and $\eta^2 = .097$, respectively. Similarly, the effect size for attractiveness was also medium, with $\eta^2 = .069$.

To test for differences between the different robot modes, Post Hoc ANOVA tests had been performed. A Post Hoc ANOVA test is used to identify for which specific group the results differ (significantly) from each other.

Because more groups are compared to each other, the probability of making a Type I error (false positive) among a set of tests is higher. With the Bonferroni method the p-value is adjusted by dividing the significance level by the number of comparisons [31], which is three in this scenario. With Bonferroni correction for multiple comparisons, the only significant difference is on perceived novelty between the no emotions and the emotions condition [Mean = 3.2411, SD = 0.57141; Mean = 3.6875, SD = 0.53186 respectively, with $p = 0.034$].

However, adding the Bonferroni correction resulted in most of the p-values being equal to 1, which did not provide any useful information. To address this issue, a LSD Post Hoc test was performed, which removed the Bonferroni correction and generated p-values below 1. This allowed for a more in-depth analysis, but it was also noted that these results are prone to result in false positives.

Without correction for multiple comparisons, the LSD Post Hoc test showed more significant differences (see Table 4.2). In particular, a significant difference had been found between the means of Animacy for the robot without emotional expressions [Mean = 2.8917, SD = .57297] and the robot mode with emotional expressions [Mean = 3.2667, SD = .49971] with $p = .027$. A significant difference also had been found between the mean of Animacy for the robot without emotional expressions and the mean of Animacy with emotional expressions and attribution [Mean = 3.2222, SD = .48971] with $p = .027$.

A near significant difference was found between the means of Attractiveness for the robot without emotional expressions [Mean = 3.5298, SD = 0.70933] and the robot mode with emotional expressions [Mean = 3.8209, SD = .61618], $p = 0.053$. However, keep in mind that with the Bonferroni correction this would be $p = 0.159$, which is not a significant result.

In figure 4.1 the mean scores for the User Experience questionnaire can be seen. The results of the UEQ have been scaled to the scale of the Godspeed questionnaire, so for both questionnaires the answers are on a scale from 1 to 5.

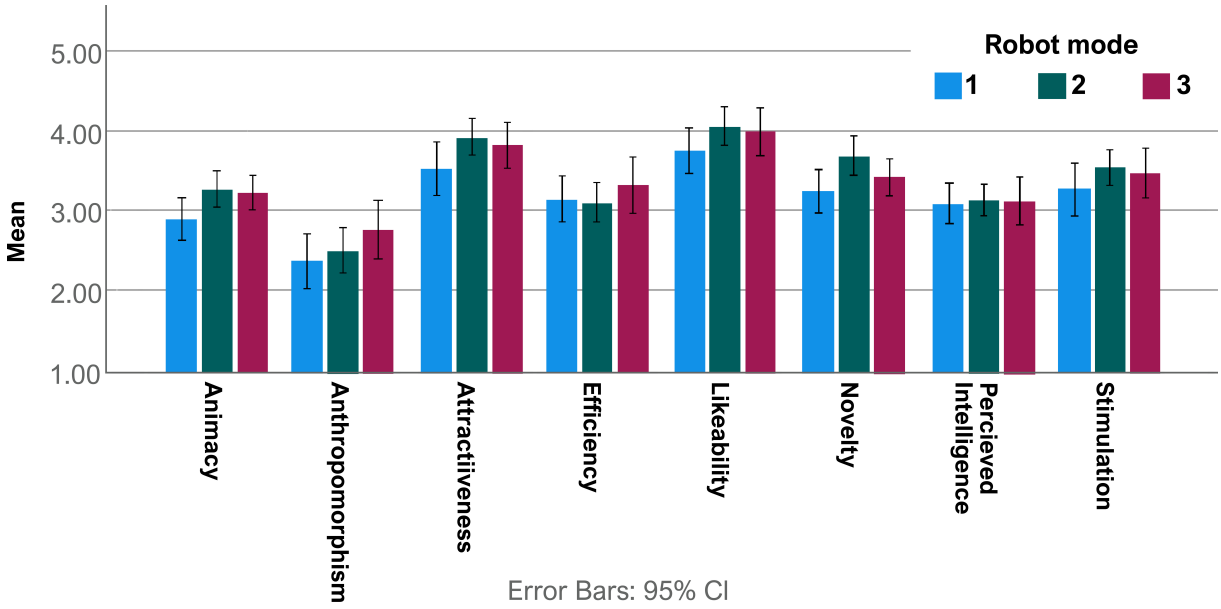


Figure 4.1: The means and 95% confidence interval error bars for the Godspeed questionnaire scores, including Anthropomorphism, Animacy, Likeability, and Intelligence, alongside the scaled means for the UEQ scores of Novelty, Stimulation, Efficiency, and Attractiveness, for different robot modes. Both the Godspeed and UEQ scores are on a 1 to 5 scale.

	Post-Hoc between robot modes			ANOVA	Effect size Partial Eta Squared
Score	1-2	1-3	2-3		
Learning Progress Score = 3 Location	.401	.577	.769	.692	.013
Loops with Learning Progress Score = 0	.313	.749	418	.583	.018
Loops with Learning Progress Score = 1	.775	.667	.473	.768	.009
Loops with Learning Progress Score = 2	.395	.895	.467	.655	.014
Inefficient Exploitation Ratio	.929	.429	.483	.683	.013
Exploitation Ratio	.719	.745	.492	.788	.008
Exploration Ratio	.719	.745	.492	.788	.008
Different command after failure	.428	.170	.563	.385	.032
Different command after succes	.641	.497	.835	.785	.008
Same command after failure	.909	.444	.379	.628	.016
Same command after succes	.569	.872	.462	.741	.010

Table 4.3: Results of LSD Post Hoc LSD and ANOVA tests on the measure values for the learning process and learning result for the comparison between robot modes.

4.2.3. Learning process and result

For the leanring process and the learning result the p-values resulting from the univariate ANOVAs and the Post Hoc ANOVA's for each dependent variable can be seen in table 4.3.

None of the results on learning progress/outcome were significantly different between the robot conditions. No differences were observed for the number of iterations needed to converge, the colour selection of the human, the exploration/exploitation behavior of the human, the learning progress over time, or the exploration/exploitation varying over time.

Figure 4.3 displays the distribution of the number of iterations for each learning progress score, as well as the total number of iterations required to achieve a learning progress score of 3, which indicated that the robot had successfully completed each colour task at least once. Figure 4.2 displays the mean learning progress score over the participants for each iteration/trial number. The learning progress score represents the amount of colours for which the task has been performed correctly at least once, which means that the score can only increase.

The distribution of command selection, based on the robot's success in executing the previous command, is presented in figure 4.6. The ratios for each variable were computed by dividing the number of times it was selected by the user, by the total number of iterations.

Figures 4.4 and 4.5 depict the exploration and exploitation scores. An exploration action was defined as a task for which the robot had not yet learned the corresponding colour, while an exploitation action was defined as a task for which the robot had already learned the colour. Inefficient exploitation was defined as performing an exploitation action before reaching a learning progress score of 3.

Figure 4.4 displays the mean ratios of exploration commands, exploitation commands, and inefficient exploitation commands, along with 95% confidence interval error bars. The ratios were computed by dividing the number of times each command was executed by the total number of iterations. The sum of the exploration and exploitation ratios in the figure equals 1.00. The chosen action can only be either an exploration or an exploitation action, which explains the equal p-values in table 4.3

Figure 4.5 shows the mean exploration/exploitation trade-off throughout the iterations, with each curve representing the mean for participants in that condition. At the beginning of the experiment, the only option was an exploration action, as none of the colours had been learned. As the experiment progressed, the robot learned the colours, and participants had the option to choose between exploration and exploitation actions. By the end of the experiment, all colours had been learned, and only exploitation actions were available.

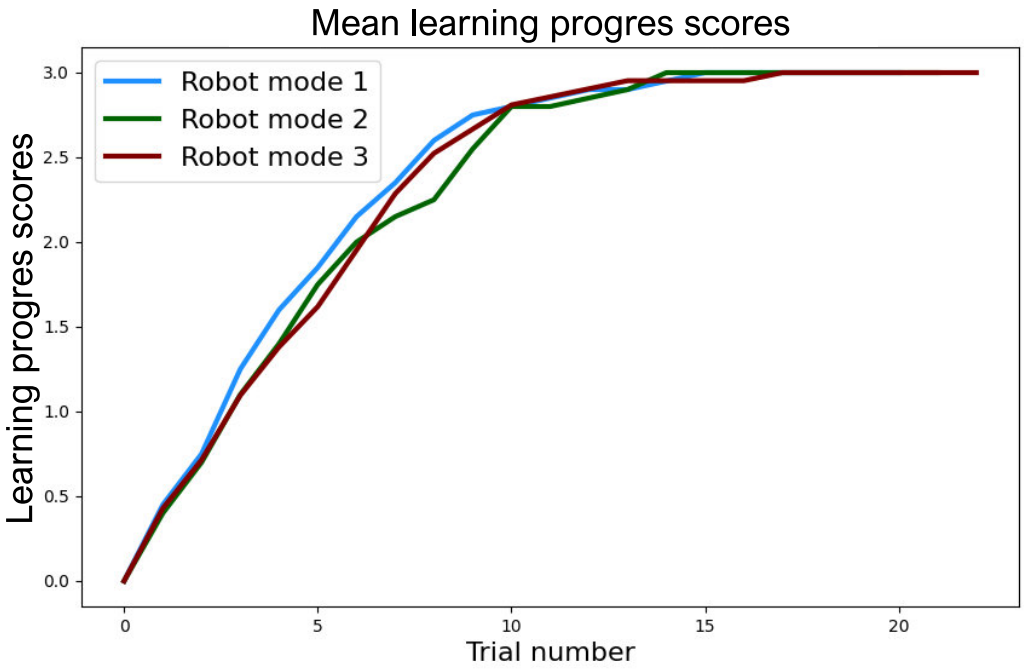


Figure 4.2: Learning curves for three robot conditions, calculated as means of the number of correct colours learned over the iterations, which is a value between 0 and 3.

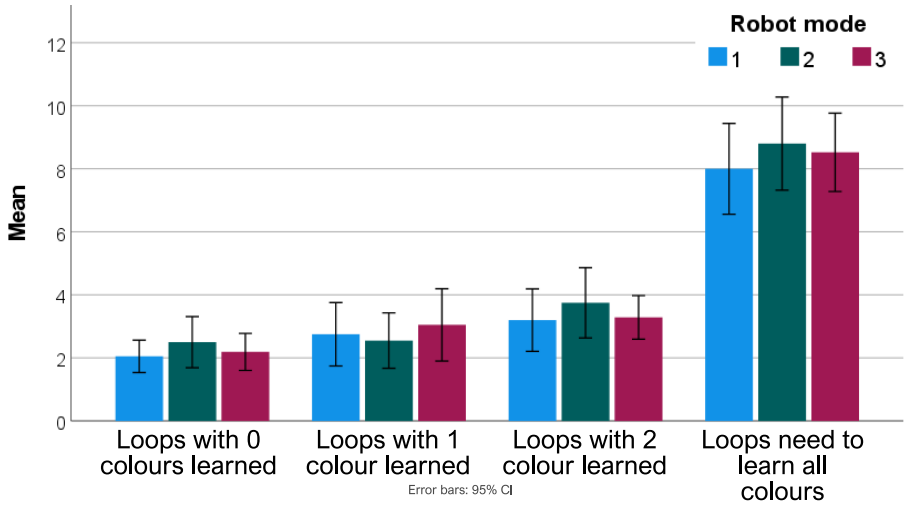


Figure 4.3: Bar plot with 95% confidence interval error bars of the mean of the number of iterations the robot took while it had learned no correct colours, 1 correct colour, 2 correct colours, and with the total sum of iterations needed before it had learned all colours correctly, which is the sum of the previous iterations.

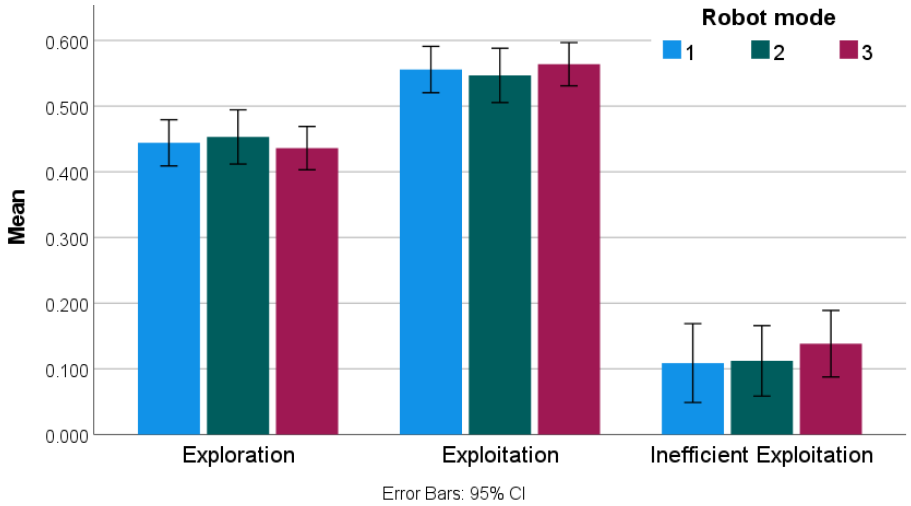


Figure 4.4: The 95% Confidence interval error bars and the means of the ratio of exploration commands, exploitation commands, and inefficient exploitation commands, calculated for each variable by dividing the number of times it occurred by the total number of iterations. The sum of the exploration and exploitation ratios in the figure adds up to 1.00.

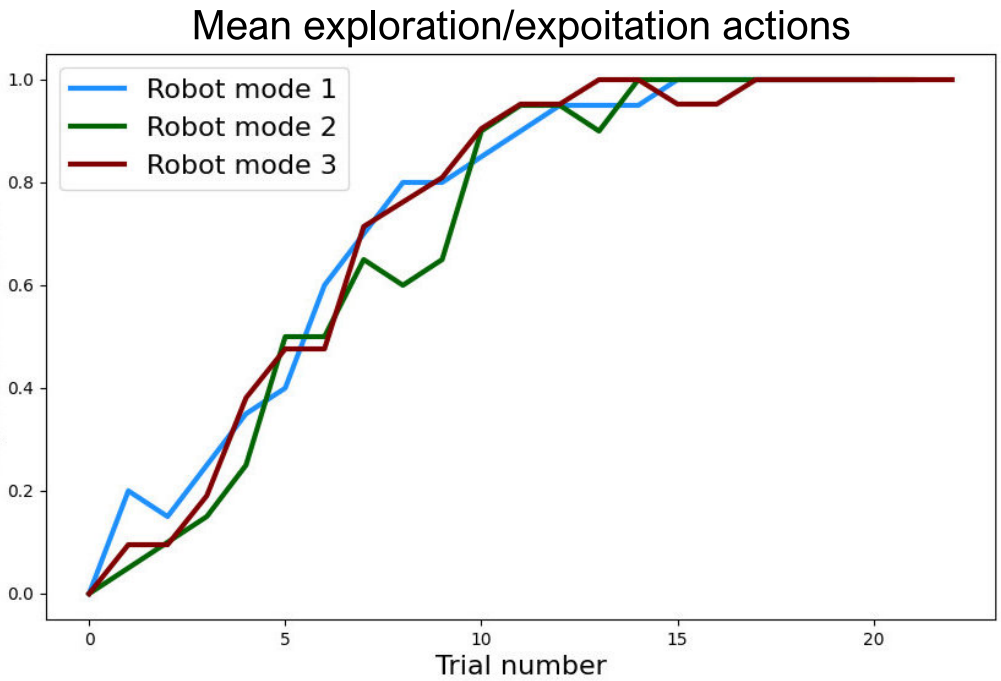


Figure 4.5: Mean of the exploration/exploitation trade-off throughout the iterations. A value of 1 indicates exploitation and a value of 0 indicates exploration.

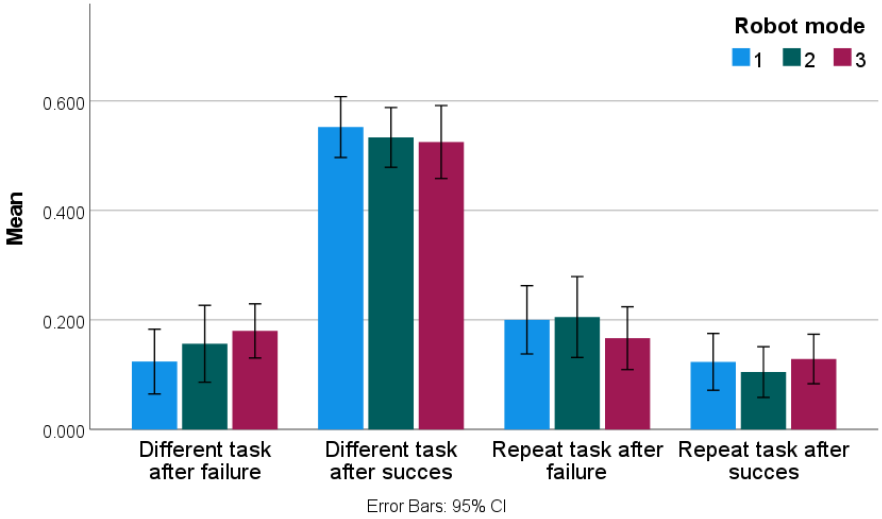


Figure 4.6: The means and 95% confidence interval error bars for the command selection ratios. The ratios were calculated for each variable by dividing the number of times it was performed by the user by the total number of iterations. The command selections were categorized based on the robot's success in executing the previous command. The sum of ratios equals 1.

4.2.4. The debriefing

During the debriefing, all of the participants for robot mode 2 and 3 were able to identify all expressed emotions.

Two participants shared their theories about the robot with emotions and their corresponding emotional attributions. One participant observed that when the robot expressed fear for a certain colour, it seemed to struggle with learning that colour. This observation was supported by instances where the robot expressed fear and subsequently made errors in displaying the correct colour. Although this participant managed to teach the robot all three colours in the end, they remained confident in their theory.

The other participant hypothesized that the robot's expression of fear was based on colours associated with negativity. This was supported by the fact that the only colour that the robot showed fear for in this experiment coincidentally happened to be red.

5
Conclusion/Discussion

The results demonstrate minimal differences between a robot without emotional expressions, one with emotional expressions based on the learning process, and one with emotional expressions and an explanation for its emotion.

This suggests that for uncomplicated tasks, emotional expressions grounded in RL neither help nor hurt. Participants perceived the robot's expressions as intended, the emotional model was thoroughly evaluated in previous studies [14, 22, 23, 24, 25], pilot experiments were conducted to test the setup, all the participants understood the task indicated by the completion of the task and lack of questions that indicated otherwise, and the differences between the conditions were very noticeable. Therefore, it is highly unlikely that this lack of a clear result can be explained by a lack of manipulation or a methodological flaw in the setup. Furthermore, since there was no negative effect on the user experience, the expressions were apparently seen as natural and did not hinder the human in the task.

The learning task was carefully designed: it was simple enough to comprehend, involved a small RL problem, ran smoothly and responded fast on the NAO robot, and was a clear teaching challenge that was easy for the human teacher to monitor. It is believed that the minimal effect found in this experiment has three reasons.

Firstly, the task is straightforward and easy, for a human, to oversee. The participant could easily keep track of the colours that the robot had correctly shown. As emotions are simulated based on the ”mental state” of the RL system of the robot, the additional information this emotion gives to the human teacher is perhaps not necessary. Emotion simulation might be more beneficial for tasks that are more complex.

Secondly, even though the robot's expressed emotion in this task provides information about the state of the robot, there is no reason for the human to adapt the teaching strategy, as the goal is to teach the robot the colours, whether or not it is happy, sad, hopeful, or fearful. For instance, if the robot displays fear towards a particular colour command, there is no appropriate action for the human to take, such as refraining from teaching that colour.

Thirdly, due to the task being so simple and fast, there was never a moment when the user had to or could intervene. For example: if fear is a genuine signal of danger ahead, the human teacher may stop the robot from going on and steer it towards another area of the task. Alternatively, if the robot autonomously explores, and when the human sees the robot's expression of hope, based on a falsely predicted future benefit, the human can stop the robot. For this reason, the robot needs to have some autonomy in the task, and a better impact measure of robot conditions may be the number of (constructive) interventions the human teacher performs.

It is a valuable finding that emotional expressions based on RL did not hinder the human teacher for this uncomplicated task. In fact, the neutral robot mode outperformed other two robot modes due to the movements and sounds used to simulate the emotions. The neutral mode was easier to program than the emotional expression mode. Once the learning algorithm and sound recognition were established, the neutral mode was operational, while programming for the emotional expression mode required setting up future TD prediction and movements for expressing emotions. Thus, knowing that a simple learning task does not require complex behaviour for the human teacher to perform effectively can save both time and effort.

Upon reflection, it is believed that the task may not have been suitable. This realization in itself is considered a valuable contribution to the field. It is important that future research on interactive robot learning with human teachers, where emotions are used as social signals to the teacher, takes into account the aforementioned three aspects:

- The task complexity needs to be such that the human teacher cannot easily oversee the complete process.
- Emotions need to convey information about how the human can influence the actions of the robot.
- The robot needs to have some meaningful autonomy in solving the task so that the human can intervene triggered by the robot's emotion.

References

[1] R.R. Hoffman et al. “Metrics for explainable AI: Challenges and prospects”. In: *arXiv preprint arXiv:1812.04608* (2018).

[2] J.Y.C. Chen and M.J. Barnes. “Human–agent teaming for multirobot control: A review of human factors issues”. In: *IEEE Transactions on Human-Machine Systems* 44.1 (2014), pp. 13–29.

[3] C.Breazeal et al. “Effects of nonverbal communication on efficiency and robustness in human-robot teamwork”. In: *2005 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2005, pp. 708–713.

[4] J. Kober, J.A. Bagnell, and J. Peters. “Reinforcement learning in robotics: A survey”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274. DOI: 10.1177/0278364913495721. URL: http://ijr.sagepub.com/content/32/11/1238.abstract.

[5] J. Broekens and M. Chetouani. “Towards transparent robot learning through TDRL-based emotional expressions”. In: *IEEE Transactions on Affective Computing* 12.2 (2019), pp. 352–362.

[6] E. T. Mueller. *Transparent Computers: Designing Understandable Intelligent Systems*. CreateSpace Independent Publishing Platform, 2016. ISBN: 1523408340.

[7] M. Boden et al. “Principles of robotics: regulating robots in the real world”. In: *Connection Science* 29.2 (2017), pp. 124–129. DOI: 10.1080/09540091.2016.1271400. eprint: https://doi.org/10.1080/09540091.2016.1271400. URL: https://doi.org/10.1080/09540091.2016.1271400.

[8] C. Duhigg. “What Google Learned From Its Quest to Build the Perfect Team.” In: *The New York Times* (2016). URL: https://www.nytimes.com/2016/02/28/magazine/what-google-learned-from-its-quest-to-build-the-perfect-team.html.

[9] A. Edmondson, R. M. Kramer, and K. S. Cook. “Psychological safety, trust, and learning in organizations: A group-level lens building the future-how cross-industry teaming works”. In: *Trust and distrust in organizations: Dilemmas and approaches* (December 2004).

[10] E. J. de Visser et al. “Towards a Theory of Longitudinal Trust Calibration in Human–Robot Teams”. In: *International Journal of Social Robotics* 12 (2 May 2020), pp. 459–478. ISSN: 18754805. DOI: 10.1007/s12369-019-00596-x.

[11] J. Broekens. “A temporal difference reinforcement learning theory of emotion: A unified view on emotion, cognition and adaptive behavior”. In: *arXiv preprint arXiv:1807.08941* (2018).

[12] J. Broekens, E. Jacobs, and C.M. Jonker. “A reinforcement learning model of joy, distress, hope and fear”. In: *Connection Science* (2015), pp. 1–19. ISSN: 0954-0091. DOI: 10.1080/09540091.2015.1031081. URL: http://dx.doi.org/10.1080/09540091.2015.1031081.

[13] J. Broekens and L. Dai. “A TDRL Model for the Emotion of Regret”. In: *2019 8th International Conference on Affective Computing and Intelligent Interaction (ACII)*. IEEE, 2019, pp. 150–156. ISBN: 1728138884.

[14] L. Dai and J. Broekens. “Simulating Fear as Anticipation of Temporal Differences: an experimental investigation”. In: *2021 9th International Conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW)*. IEEE. 2021, pp. 1–8.

[15] A. Hinderks, M. Schrepp, and J. Thomaschewski. *User Experience Questionnaire*. 2018. URL: https://www.ueq-online.org/ (visited on 11/26/2022).

[16] B. Christoph et al. “Measurement instruments for the anthropomorphism, animacy, likeability, perceived intelligence, and perceived safety of robots”. In: vol. 1. 2009. DOI: 10.1007/s12369-008-0001-3.

[17] L. P. Kaelbling, M. L. Littman, and A. W. Moore. “Reinforcement Learning: A Survey”. In: *Journal of Artificial Intelligence Research* 4 (1996). DOI: 10.1613/jair.301.

[18] R.S. Sutton and A.G. Barto. *Reinforcement Learning*. Adaptive computation and machine learning series. Cambridge, MA : The MIT Press, 2018. ISBN: 9780262039246.

[19] .S. Sutton and A.G. Barto. “Reinforcement Learning: An Introduction(2nd Ediction Draft)”. In: *Kybernetes* (2017). ISSN: 0368492X.

[20] A. Asghari, M.K. Sohrabi, and F. Yaghmaee. “Task scheduling, resource provisioning, and load balancing on scientific workflows using parallel SARSA reinforcement learning agents and genetic algorithm”. In: *The Journal of Supercomputing* 77 (3 2021). ISSN: 1573-0484. DOI: 10.1007/s11227-020-03364-1.

[21] M. Tokic ad G. Palm. “Value-Difference Based Exploration: Adaptive Control between Epsilon-Greedy and Softmax”. In: (2011). Ed. by Joscha Bach and Stefan Edelkamp, pp. 335–346.

[22] T. Moerland, J. Broekens, and C.M. Jonker. “Fear and Hope Emerge from Anticipation in Model-Based Reinforcement Learning”. In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*. Ed. by Subbarao Kambhampati. AAAI Press, 2016, pp. 848–854.

[23] M.B. Moussa and N. Magnenat-Thalmann. “Toward socially responsible agents: integrating attachment and learning in emotional decision-making”. In: *Computer Animation and Virtual Worlds* 24.3-4 (2013), pp. 327–334.

[24] A. Rossi et al. “Evaluation of a humanoid robot’s emotional gestures for transparent interaction”. In: *Social Robotics: 13th International Conference, ICSR 2021, Singapore, Singapore, November 10–13, 2021, Proceedings 13*. Springer. 2021, pp. 397–407.

[25] G. Angelopoulos et al. “Transparent Interactive Reinforcement Learning Using Emotional Behaviours”. In: *Social Robotics: 14th International Conference, ICSR 2022, Florence, Italy, December 13–16, 2022, Proceedings, Part I*. Springer. 2023, pp. 300–311.

[26] P. Thoma, D. Bauser, and B. Suchan. “BESST (Bochum Emotional Stimulus Set)-A pilot validation study of a stimulus set containing emotional bodies and faces from frontal and averted view.” In: *Psychiatry research* 209 (Dec. 2012). DOI: 10.1016/j.psychres.2012.11.012.

[27] J. Wu et al. “Generalized zero-shot emotion recognition from body gestures”. In: *Applied Intelligence* 52 (June 2022). DOI: 10.1007/s10489-021-02927-w.

[28] M. Schrepp, A. Hinderks, and J. Thomaschewski. “Applying the user experience questionnaire (UEQ) in different evaluation scenarios”. In: *Design, User Experience, and Usability. Theories, Methods, and Tools for Designing the User Experience: Third International Conference, DUXU 2014, Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014, Proceedings, Part I 3*. Springer. 2014, pp. 383–392.

[29] Lee J. Cronbach. “Coefficient alpha and the internal structure of tests”. In: *Psychometrika* 16 (3 1951). ISSN: 00333123. DOI: 10.1007/BF02310555.

[30] P. Stoker, G. Tian, and J.Y. Kim. “Analysis of variance (Anova)”. In: 2020. DOI: 10.4324/9780429325021-11.

[31] W. Haynes. “Bonferroni correction”. In: *Encyclopedia of systems biology* (2013), pp. 154–154.

A
Consent form and Questionnaires

A.1. Consent form

Effect of robot expressing emotions on user experience and reinforced learning process and reinforced learning result

Informed consent form for participants

Researchers
Floortje Lijcklama à Nijeholt
E-mail:

Dr.ir. J. C. F. de Winter
E-mail:

Location
Faculty of Mechanical Engineering Delft University of Technology Mekelweg 2, 2628 CD, Delft, The Netherlands

Research purpose and experiment procedure

The purpose of this experiment is to investigate how robots can learn from humans. You will be asked to teach the robot the colours red, green, and blue, within 10 minutes. There is no need to rush the task but do try to achieve this goal. After that, you will be asked to fill in a questionnaire about how you experienced your interaction with the robot, as well as a few demographic questions. The total duration of the experiment will be about 15 minutes.

Risk of participating

There are no expected risks. If you experience any discomfort, please inform the experiment supervisor so that the experiment can be stopped.

Right to withdraw

Your participation is completely voluntary, and you may stop at any time during the experiment for any reason. There will be no negative consequences for withdrawing from the experiment.

Data treatment

All data will be collected anonymously and used for academic research only. You will not be personally identifiable in future publications based on this work, in data files shared with other researchers, or in data repositories. This signed consent form will be kept in a dedicated locker.

Prevention of the spread of COVID-19

You may not participate if you show any symptoms indicative of COVID-19. You will be asked to disinfect your hands before touching any equipment and surfaces. All equipment used in the experiment and surfaces will be disinfected before participation.

Please respond to the following statements

Statement	Yes	No
I consent to participate voluntarily in this study.	○	○
I have read and understood the information provided in this document.	○	○
I adhere to the preventative measures with regard to COVII-19 explained above.	○	○
I understand that I can withdraw from the study at any time without anynegative consequences.	○	○
I agree that the data collected during the experiment will be used for academic research and may be presented ina publication and public data repositor.	○	○

Signature

Name:

Date:

Signature: _____

A.2. English questionnaire

Please make your evaluation now.

This experiment is done to test interactions between a teacher and a robot. In this interaction the robot was tasked with learning different colours, with the participant as teacher.

For the experiment we will use your following data: age, gender, experience with programming, experience with robots, and the answers in the following questionnaires. This data will only be used for this research.

Do only continue if you agree to the usage of the before mentioned data!

For the assessment of the NAO, please fill out the following questionnaires. The questionnaire consists of pairs of contrasting attributes that may apply to the product. The circles between the attributes represent gradations between the opposites. The first questionnaire has scores between 1 and 5 and the second questionnaire has scores between 1 and 7. You can express your agreement with the attributes by ticking the circle that most closely reflects your impression.

Example:

Attractive	O ⊗ O O O O O	Unattractive

This response would mean that you rate the application as more attractive than unattractive. Please decide spontaneously. Don't think too long about your decision to make sure that you convey your original impression.

Sometimes you may not be completely sure about your agreement with a particular attribute, or you may find that the attribute does not apply completely to the particular product. Nevertheless, please tick a circle in every line.

It is your personal opinion that counts. Please remember: there is no wrong or right answer!

Before you start with the questionnaire about the robots behaviour, please answer the following questions about yourself.

Age:							
Gender:							
How many experience do you have with robots?	0	O	O	O	O	O	5
How many experience do you have with programming?	0	O	O	O	O	O	5

Pleaserate your impression of Nao on these scales:

Dislike	O	O	O	O	O	Like
Awful	O	O	O	O	O	Nice
Apathetic	O	O	O	O	O	Responsive
Dead	O	O	O	O	O	Alive
Foolish	O	O	O	O	O	Sensible
Moving rigidly	O	O	O	O	O	Moving elegantly
Artificial	O	O	O	O	O	Lifelike
Machinelike	O	O	O	O	O	Humanlike
Mechanical	O	O	O	O	O	Organic
Unkind	O	O	O	O	O	Kind
Incompetent	O	O	O	O	O	Competent
Unconscious	O	O	O	O	O	Conscious
Fake	O	O	O	O	O	Natural
Unintelligent	O	O	O	O	O	Intelligent
Unpleasant	O	O	O	O	O	Pleasant
Irresponsible	O	O	O	O	O	Responsible
Unfriendly	O	O	O	O	O	Friendly
Ignorant	O	O	O	O	O	Knowledgeable
Inert	O	O	O	O	O	Interactive
Stagnant	O	O	O	O	O	Lively

Pleasealso rate your impression of Nao on these scales:

Annoying	O	O	O	O	O	O	O	Enjoyable
Not understandable	O	O	O	O	O	O	O	Understandable
Creative	O	O	O	O	O	O	O	Dull
Easyto learn	O	O	O	O	O	O	O	Difficult to learn
Valuable	O	O	O	O	O	O	O	Inferior
Boring	O	O	O	O	O	O	O	Exciting
Not interesting	O	O	O	O	O	O	O	Interesting
Unpredictable	O	O	O	O	O	O	O	Predictable
Fast	O	O	O	O	O	O	O	Slow
Inventive	O	O	O	O	O	O	O	Conventional
Obstructive	O	O	O	O	O	O	O	Supportive
Good	O	O	O	O	O	O	O	Bad
Complicated	O	O	O	O	O	O	O	Easy
Unlikable	O	O	O	O	O	O	O	Pleasing
Usual	O	O	O	O	O	O	O	Leading edge
Unpleasant	O	O	O	O	O	O	O	Pleasant
Secure	O	O	O	O	O	O	O	Not secure
Motivating	O	O	O	O	O	O	O	Demotivating
Meets expectations	O	O	O	O	O	O	O	Doesnot meet expectations
Inefficient	O	O	O	O	O	O	O	Efficient
Clear	O	O	O	O	O	O	O	Confusing
Impractical	O	O	O	O	O	O	O	Practical
Organized	O	O	O	O	O	O	O	Cluttered
Attractive	O	O	O	O	O	O	O	Unattractive
Friendly	O	O	O	O	O	O	O	Unfriendly
Conservative	O	O	O	O	O	O	O	Innovative

A.3. Dutch questionnaire

De vragenlijst

Dit experiment wordt gedaan om interacties te testen tussen een leraar en de robot. In deze interactie kreeg de robot de taak om kleuren te leren met de deelnemer als leraar.
Van u als deelnemer wordt uw leeftijd, geslacht, programmeerervaring, ervaring met robots, en uw antwoorden op deze vragenlijst opgeslagen. Deze gegevens worden alleen gebruikt voor het verwerken van de resultaten in dit onderzoek.

Als u het ermee eens bent dat deze gegevens gebruikt worden, kan u verder gaan met deze vragenlijst.

Voor de beoordeling van het product, vragen we u de twee vragenlijsten op de volgende pagina in te vullen. De vragenlijsten bestaat uit verschillende punten met ieder twee tegengestelde eigenschappen die van toepassing zijn op het product. De rondjes staan voor verschillende gradaties. De ene vragenlijst op een schaal van 1 tot 5 en de andere schaal gaan van 1 tot 7. U kunt uw beoordeling geven door het rondje, die het meest uw indruk weerspiegelt, aan te vinken.

Voorbeeld:

Aantrekkelijk	O	⊗	O	O	O	O	O	Onaantrekkelijk

Dit antwoord zou betekenen dat u het product beoordeelt als meer aantrekkelijk dan onaantrekkelijk.

Het is de bedoeling dat u uw eerste ingeving invult. Wacht niet te lang met invullen om te voorkomen dat u gaat twijfelen over uw eerste ingeving.

Het kan zijn dat u niet helemaal zeker bent van uw antwoord of dat u de eigenschap niet volledig van toepassing vindt, kruis dan toch een rondje aan.

Het is uw mening die telt. Er is geen goed of fout antwoord!

Voordat u met de vragenlijst begint over het gedrag van de robot, worden er eerst 4 vragen over uzelf gesteld:

Leeftijd:							
Geslacht:							
Hoeveel ervaring heeft u met robots?	0	O	O	O	O	O	5
Hoeveel ervaring heeft u met programmeren?	0	O	O	O	O	O	5

Beoordeel de interactie met NAO door een cijfer te geven op de volgende punten:

Afkeer	O	O	O	O	O	Geliefd
Afschuwelijk	O	O	O	O	O	Mooi
Apatisch	O	O	O	O	O	Responsief
Dood	O	O	O	O	O	Levend
Dwaas	O	O	O	O	O	Gevoelig
Houterige bewegingen	O	O	O	O	O	Vloeiende bewegingen
Kunstmatig	O	O	O	O	O	Levensecht
Lijkend op een machine	O	O	O	O	O	Lijkend op een mens
Mechanisch	O	O	O	O	O	Organisch
Niet lief	O	O	O	O	O	Lief
Onbekwaam	O	O	O	O	O	Bekwaam
Onbewust	O	O	O	O	O	Bewustzijn
Onecht	O	O	O	O	O	Natuurlijk
Onintelligent	O	O	O	O	O	Intelligent
Onplezierig	O	O	O	O	O	Plezierig
Onverantwoordelijk	O	O	O	O	O	Verantwoordelijk
Onvriendelijk	O	O	O	O	O	Vriendelijk
Onwetend	O	O	O	O	O	Veel wetend
Passief	O	O	O	O	O	Interactief
Stilstaand	O	O	O	O	O	Levendig

Beoordeel daarna ook de interactie met NAO op de volgende punten

Onplezierig	O	O	O	O	O	O	O	Plezierig
Onbegrijpelijk	O	O	O	O	O	O	O	Begrijpelijk
Creatief	O	O	O	O	O	O	O	Saai
Makkelijk te leren	O	O	O	O	O	O	O	Moeilijk te leren
Waardevol	O	O	O	O	O	O	O	Inferieur
Vervelend	O	O	O	O	O	O	O	Spannend
Oninteressant	O	O	O	O	O	O	O	Interessant
Onvoorspelbaar	O	O	O	O	O	O	O	Voorspelbaar
Snel	O	O	O	O	O	O	O	Langzaam
Origineel	O	O	O	O	O	O	O	Conventioneel
Belemmerend	O	O	O	O	O	O	O	Ondersteunend
Goed	O	O	O	O	O	O	O	Slecht
Complex	O	O	O	O	O	O	O	Eenvoudig
Afstotend	O	O	O	O	O	O	O	Aantrekkelijk
Gebruikelijk	O	O	O	O	O	O	O	Nieuw
Onaangenaam	O	O	O	O	O	O	O	Aangenaam
Vertrouwd	O	O	O	O	O	O	O	Niet vertrouwd
Motiverend	O	O	O	O	O	O	O	Demotiverend
Volgens verwachtingen	O	O	O	O	O	O	O	Niet volgens verwachtingen
Inefficiënt	O	O	O	O	O	O	O	Efficiënt
Overzichtelijk	O	O	O	O	O	O	O	Verwarrend
Onpraktisch	O	O	O	O	O	O	O	Praktisch
Ordelijk	O	O	O	O	O	O	O	Rommelig
Aantrekkelijk	O	O	O	O	O	O	O	Onaantrekkelijk
Aardig	O	O	O	O	O	O	O	Onaardig
Conservatief	O	O	O	O	O	O	O	Innovatief

B
Source Code

The main code

```
from agent import Agent
from environment import Environment
from robot import Robot
import time
import numpy as np
from autobahn.twisted.component import Component, run
from twisted.internet.defer import inlineCallbacks
from autobahn.twisted.util import sleep
import csv
states = 7
actions = 4
s = (states, actions, states)
transition_table = np.zeros(s)
agent = Agent(7, 4)
EMOTIONS = True
REASONS = True
DOUBLE_EMOTIONS = True
OptimismFactor  = 1.1 # how much more the robot values the positive expected TD's over the negative once for state 0
COLOURS_LEARNED = {0:False, 1: False, 2: False}

@inlineCallbacks
def main(session, details):
    datafile = open('Data.csv', 'a')
    writer = csv.writer(datafile, lineterminator='\n')
    prev_state = []
    next_state = []
    prev_action = []
    reward = 0
    STANDING = True
    LEARNING = True
    TD = 0
    state = 0
    Language_settings = input("Language settings [1=Nederlands, 2=English]:\n")
    Robot_mode = input("The emotive mode for the robot [1=no emotions, 2=simple emotions, 3=emotions and reasoning]:\n")
    User_name = "x" #input("The name of the user is:\n")
    Language_settings = int(Language_settings)
    environment = Environment(Language_settings)
    if int(Robot_mode) == 1:
        EMOTIONS = False
        REASONS = False
    elif int(Robot_mode) == 2:
        EMOTIONS = True
        REASONS = False
    elif int(Robot_mode) == 3:
        EMOTIONS = True
        REASONS = True

    robot = Robot(Language_settings, User_name, EMOTIONS, REASONS)
    yield robot.startup(session)
    startTime =  time.perf_counter()
    while LEARNING == True:
        if state==0:
            if (Language_settings == 1): yield session.call("rie.dialogue.say", text="Daar gaan we")
            else: yield session.call("rie.dialogue.say", text="Here we go")
```

```
55          if transition_table[state,:].max()>0:
56              Scaled_Expected_TD, reasons = agent.EmotionTD(state, transition_table,
                    OptimismFactor)
57              if EMOTIONS:
58                  if Scaled_Expected_TD > 0.2:
59                      yield robot.hopefull(session, state, reasons)
60                      STANDING = False
61                  elif Scaled_Expected_TD < -0.0:
62                      yield robot.fear(session, state, reasons)
63                      STANDING = False
64          # ask the environment for the possible actions
65          pos_actions = environment.possible(state)
66          # agent selects an action
67          action = agent.select_action(state, pos_actions)
68          # print(f"Am I standing: {STANDING}")
69          if STANDING ==  False:    #If the robot is still striking a pose, it will go back to
                standing
70              yield robot.stand(session)
71              STANDING = True
72              # print(f"Am I standing: {STANDING}")
73          if isinstance(action, int): # only if performing an action is possible
74              if isinstance(prev_state, int):
75                  # agent performs internal updates based on sampled experience
76                  agent.step(prev_state, prev_action, reward, state, action)
77          # agent performs the selected action
78          next_state, reward = yield environment.step(session, state, action, prev_state,
                prev_action)
79          if reward != 0:
80              TD  = agent.td(prev_state, prev_action, reward, state)
81          elif (DOUBLE_EMOTIONS and action == 0):
82              TD  = agent.td(state, action, reward, next_state)
83          if EMOTIONS:
84              if TD > 0.2:
85                  yield robot.happy(session)
86                  yield sleep(1)
87                  yield robot.slowstand(session)
88                  STANDING = True
89              elif TD < -0.2:
90                  yield robot.sad(session)
91                  yield sleep(1)
92                  yield robot.stand(session)
93                  STANDING = True
94          else:
95              robot.no_emotions(session)
96
97          TD = 0
98          # update the transition table
99
100         if isinstance(action, list): # if next action is go back to start
101             FORCED = True
102             # agent performs internal updates based on sampled experience
103             agent.step(prev_state, prev_action, reward, state, prev_action, FORCED)
104             prev_action = []
105             prev_state = []
106         else:
107             prev_action = action
108             prev_state = state
109         transition_table[state, action, next_state] += 1
110         state = next_state
111         route, iterations, flipRoute, learningScore, totalExploration, totalExplotation,
                ExplorationRatio, qtable0, qtable1, qtable2, qtable3 = agent.ExportValues(state,
                reward)
112         print(f"learning score count {learningScore.count(3)}")
113         if ((time.perf_counter() - startTime)>=600.0) or (learningScore.count(3)>5) and (state
                 == 0): # If 10 minutes have passed or colours have been learned enough, we will
                stop the test when the learning loop will restart or session.call("rom.optional.
                behavior.play", name="BlocklyTouchToes")
114             yield robot.end(session)
115             LEARNING = False
116
117     session.leave() # Sluit de verbinding met de robot
```

```
118     new_data = User_name, Robot_mode, iterations, route, flipRoute, learningScore,
            totalExploration, totalExplotation, ExplorationRatio, qtable0, qtable1, qtable2,
            qtable3
119     writer.writerow(new_data)
120     datafile.close()
121
122 wamp = Component(
123         transports=[{
124                 "url": "ws://wamp.robotsindeklas.nl",
125                 "serializers": ["msgpack"],
126                 "max_retries": 0
127         }],
128         realm="rie.641037a6cd363302f221068e",
129 )
130
131
132 wamp.on_join(main)
133
134 if __name__ == "__main__":
135         run([wamp])
```

The agent code

```python
import numpy as np
from q_table import QTable
import math
from environment import Environment
import random

class Agent:

    def __init__(
        self,
        observation_space=7,
        action_space=4,
        alpha=0.5,
        gamma=0.9,
        epsilon=0.7,
        epsilon_decay=0.85,
        epsilon_min=0.1
    ):
        """ Initialize agent.

        Params
        ======
        - nA: number of actions available to the agent
        """
        self.states           = observation_space
        self.nA               = action_space
        self.possible_actions = np.arange(self.nA)
        self.all_actions      = np.arange(self.nA)
        self.epsilon_decay    = epsilon_decay
        self.epsilon          = epsilon
        self.epsilon_min      = epsilon_min
        self.q_table          = QTable(
            observation_space=observation_space,
            action_space=action_space,
            alpha=alpha,
            gamma=gamma)
        self.environment      = Environment(1)
        self.flipRoute        = []
        self.route            = []
        self.taskDescription = {0: "ask", 1: "red", 2: "green", 3: "blue"}
        self.learningScore    = []
        self.totalExploration = 0
        self.totalExplotation = 0
        self.ExplorationRatio = []
        self.RewardValue = 0


    def update_epsilon(self):
        self.epsilon = max(self.epsilon * self.epsilon_decay, self.epsilon_min)

    def epsilon_greedy(self, state, pos_actions):
        policy                  = np.ones(self.nA) * (self.epsilon/self.nA)
        best_action_idx         = np.argmax(self.q_table.q(state))
        if pos_actions.count(best_action_idx)==0:    #if the max value is an action that the
            robot shouldn't know yet
            if list(self.q_table.q(state)).count(0)==len(self.q_table.q(state)): # if
                everything is 0, the odds are all equal
                policy                  = np.ones(self.nA) * (1/self.nA)
            else: # if not
                index =  np.where(self.q_table.q(state)[pos_actions]==0.)[0] # find all zero's
                new_max_val = (1 - ((self.nA-len(index))*(self.epsilon / self.nA)))/len(index)
                for i in index:
                    policy[i+min(pos_actions)]=new_max_val
        else:
            policy[best_action_idx] = (1 - self.epsilon) + (self.epsilon / self.nA)
        print(f"policy is {policy}, epsillon is {self.epsilon}")
        return policy

    def select_action(self, state, pos_actions):
```

```
        """ Given the state, select an action.
        Params
        ======
        - state: the current state of the environment
        - pos_actions: the possible actions the agent could perform from this state

        Returns
        =======
        - action: an integer, compatible with the task's action space
        """
        if len(pos_actions) == 0:
            action = []
            return action
        elif len(pos_actions) == 1:
            action = pos_actions[0]
            return action
        else:
            # self.possible_actions = np.array(pos_actions)
            action_probabilities  = self.epsilon_greedy(state, pos_actions)
            # agent updates the variablele epsillon
            self.update_epsilon()
            checked = 0
            #return the chosen action
            while checked == 0:
                choice = np.random.choice(self.all_actions, p=action_probabilities)
                checked = pos_actions.count(choice)
            return int(choice)

    def step(self, state, action, reward, next_state, next_action, FORCED = False):
        """ Update the agent's knowledge, using the most recently sampled tuple.

        Params
        ======
        - state: the previous state of the environment
        - action: the agent's previous choice of action
        - reward: last reward received
        - next_state: the current state of the environment
        - FORCED; when the agent is at it last state the update will be "forced", True or
            False, if not identified, it is assumed to be false
        """
        self.q_table.sarsa_update(state, action, reward, next_state, next_action, FORCED)

    def td_table(self, transition_table):
        """ Generate a table existing out of the expected TD values for all actions that the
            agent has once performed

        Params
        ======
        - transition_table: all actions that the agent has once performed, in the form of
            state, chosen action, new state
        - state: the state for the simulation for the TD
        - action: possible actions in states
        - newstate: state that the action will take the robot to
        - TD_table: a table will all expected TD values based the states that the robot has
            once visited
        """
        s = (self.states, self.nA, self.states) # all available combi's
        TD_table = np.zeros(s)                  # first set all to zero
        for state in range(self.states):
            for action in range(self.nA):
                for newstate in range(self.states):
                    if transition_table[state, action, newstate]!= 0: # the action needs to be
                         performed at least once
                        reward = self.environment.reward(action, state)
                        TD = self.q_table.temporal_difference(state, action, reward, newstate)
                             #calculate the TD value
                        TD_table[state, action, newstate] = TD
        return TD_table

    def EmotionTD(self, s, transition_table, OptimismFactor):
        reasons = 0
```

```
133         TD_table = self.td_table(transition_table)
134         # for the task states we just calculate the maximum value, as the robot chooses via
                greedy
135         if s < 4 and s!=0:
136             future_max = self.TaskStateTD(s, TD_table)
137             # Expected_TD = future_max
138             Scaled_Expected_TD = future_max
139 
140         # for the main state we base the expexted TD on the amount of times the robot has
                visited the task state and the expected TD in that state
141         elif s == 0:
142             max_values = [0, 0, 0, 0]
143             scaled_max_values = [0, 0, 0, 0]
144             # we will check for each task state what the TD would be
145             for i in range(1,4):
146                 future_max = self.TaskStateTD(i, TD_table)
147                 max_values[i] = future_max
148                 scaled_max_values[i] = future_max
149                 if np.isnan(future_max): scaled_max_values[i] = 0
150                 elif future_max > 0: scaled_max_values[i] = future_max * OptimismFactor
151             Scaled_Expected_TDs = scaled_max_values #np.multiply(transition_table[0,0,:4],
                    scaled_max_values)
152             print(f'For the next states the TD is officially {max_values} but will be {
                    Scaled_Expected_TDs}, and the transitions are {transition_table[0,0,:4]}')
153             # print(f"scaled expected tds are: {Scaled_Expected_TDs}, abs are {abs(
                    Scaled_Expected_TDs)}, max is {max(abs(Scaled_Expected_TDs))}")
154 
155             reasons = np.where(abs(Scaled_Expected_TDs) == max(abs(Scaled_Expected_TDs)))
156             reasons = int(reasons[0][0])
157             Scaled_Expected_TD = Scaled_Expected_TDs[reasons]
158             # Added_TD = [scaled_max_values[i] + TD_table[0,0,i] for i in range(len(
                    scaled_max_values))]
159 
160         return Scaled_Expected_TD, reasons
161 
162     def TaskStateTD(self, s, TD_table):
163         # for the task states we just calculate the maximum value, as the robot chooses via
                greedy
164         Future_TD = TD_table[s, :, :]
165         for action in range(self.nA):
166             for newstate in range(self.states):
167                 if Future_TD[action, newstate]== 0:
168                     Future_TD[action, newstate] = None
169         future_max = np.nanmax(Future_TD)
170         # print(f'Future_TD looks like {Future_TD}')
171         # print(f'Future_Max is {future_max} abs max is {AbsMax}')
172         # print(f'Is this the actual value of the absMax? {BetterMax}')
173         return future_max
174 
175     def td(self, state, action, reward, newstate):
176         """Calculate the indivudual TD values
177 
178         Params
179         ======
180         - TD: The temporal difference based on the following parameters:
181         - state: the state from which we will calculate the TD
182         - action: the chocen action for that state
183         - newstate: state that the action will take the robot to
184         - reward: the simulated reward the robot will get with this combination
185         """
186 
187         TD = self.q_table.temporal_difference(state, action, reward, newstate) #calculate the
                TD value
188         return TD
189 
190     # def q_table(self):
191     #     return QTable.all_q()
192 
193     def ExportValues(self, state, reward):
194         Flip = False
195         print(f"state is {state}")
```

```
196         if reward != 0:
197             self.RewardValue = reward
198         if 4 <= state <= 6:         # for this values the robot has tried to perform an colour
                action, which indicates the end of a single loop
199             self.oldTask = self.task
200             learningScore = 0
201             for i in range (1,4):   # see for how many of the tasks the max q value has been
                    found
202                 if self.q_table.max_q(i) > 0.0:
203                     learningScore += 1
204             self.learningScore.append(learningScore)
205             print(f"Current learning score is {self.learningScore}")
206             self.qtable0.append(self.q_table.q(0))
207             self.qtable1.append(self.q_table.q(1))
208             self.qtable2.append(self.q_table.q(2))
209             self.qtable3.append(self.q_table.q(3))
210         elif 1 <= state <= 3:       # for this values the user has recently given a task
211             self.task = self.taskDescription[state]
212             if self.q_table.max_q(state) > 0.0: # if route with a positive reward has been
                    found, the task will be seen as an explotation task
213                 self.totalExplotation += 1
214             else:                                   # else it is exploration
215                 self.totalExploration += 1
216             if self.totalExplotation is not 0:      # Calculating the exploration/explotation
                    ratio
217                 ExplorationRatio = self.totalExploration/self.totalExplotation
218             else:                                    # If we are going to devide by zero, we
                    don't do that
219                 ExplorationRatio = self.totalExploration
220             self.ExplorationRatio.append(ExplorationRatio)
221 
222             try:
223                 if self.oldTask == self.task: # the user decided to repeat the same task
224                     Flip = False
225                 else:               # the user switched tasks
226                     Flip = True
227                 self.flipRoute.append(self.task)
228                 self.flipRoute.append(self.RewardValue)
229                 self.flipRoute.append(Flip)
230             except AttributeError:
231                 print("first loop probably?")
232                 self.flipRoute.append(self.task)
233                 self.flipRoute.append(0)
234                 self.qtable0 = []
235                 self.qtable1 = []
236                 self.qtable2 = []
237                 self.qtable3 = []
238 
239         self.route.append(state)
240         iterations = self.route.count(0) # see how often the "main" state has been visited,
                which will indicate the amount of iterations that has been done
241         return self.route, iterations, self.flipRoute, self.learningScore, self.
                totalExploration, self.totalExplotation, self.ExplorationRatio, self.qtable0, self
                .qtable1, self.qtable2, self.qtable3
```

The environment code

```
from autobahn.twisted.component import Component, run
from twisted.internet.defer import inlineCallbacks
from autobahn.twisted.util import sleep
from robot import Robot

class Environment:
    def __init__(
        self,
        Language_settings
    ):
        """ Initialize environment.

        Params
        ======
        - Language_settings: 1 for Dutch, 2 for English
        """
        self.Language_settings          = Language_settings
        self.TimesAsked                 = 0
        self.Reward                     =[[0,  0,  0,  0, 0, 0, 0],
                                          [0,  1, -1, -1, 0, 0, 0],
                                          [0, -1,  1, -1, 0, 0, 0],
                                          [0, -1, -1,  1, 0, 0, 0]]
        self.robot                      = Robot(1, "x", False, False)

    @inlineCallbacks
    def step(self, session, state, action, prev_state, prev_action):
        """ Update the environment based on the action that the agent has chosen

        Params
        ======
        - action: the agent's choice of action
            0 for "ask", 1 for "red", 2 for "green", 3 for "blue'
        - state: the current state the agent is in
            0 for "home", 1 for "taks red", 2 for "task green", 3 for "task blue', 4 for "
                perform red", 5 for "perform green", 6 for "perform blue"

        Returns
        - reward: reward received
            only a value in state 4, 5, 6, else 0
        - next_state: the current state of the environment
            1 for "taks red", 2 for "task green", 3 for "task blue', 4 for "perform red", 5
                for "perform green", 6 for "perform blue", [] for "restart"

        """
        if action == 0: # if action is "ask"
            next_state = yield self.perform(session, action)
        elif action == 1 or action == 2 or action == 3: # if action is change to a colour
            yield self.perform(session, action)
            next_state = 3 + action
        else:
            fake_reward = yield self.receive_reward(session, state)
            next_state = 0
        if isinstance(prev_action, int):
            reward = self.reward(prev_action, prev_state)
            # print(f"reward is {reward}")
        else: reward = 0
        return next_state, reward#(action, state)


    def reward(self, action, state):
        """ ask the environment what actions are possible

        Params
        ======
        - state: the current state of the environment
        - action: the action that has been chosen to get to this state

        Returns
        =======
```

```
68          - reward: the reward based on the action/state combination
69          """
70          reward = self.Reward[action][state]
71          return reward
72
73      def possible(self, state):
74          """ ask the environment what actions are possible
75
76          Params
77          ======
78          - state: the current state of the environment
79
80          Returns
81          =======
82          - pos_action: the possible actions the agent can perform in this state
83          """
84          if state == 0: #if in "home state"
85              pos_action = [0]  # only action is to ask for a task
86          elif state == 1 or state == 2 or state == 3: # if in a task state
87              pos_action = [1, 2, 3] #actions are changing eye colours
88          else:
89              pos_action = []
90          return pos_action
91
92      @inlineCallbacks
93      def receive_reward(self, session, state):
94          """ if possible, ask what the rewards is from the performed action, if not possible,
                rewards is zero
95
96          Params
97          ======
98          - state: the current state of the environment
99
100         Returns
101         =======
102         - reward: the rewards based on the performed action
103         """
104         # global keyword_listener2
105         self.answer = 0
106         reward = 0
107         session.call("rom.optional.behavior.play", name="BlocklyStand")
108         if (self.Language_settings == 1): #Nederlandse settings
109             yield session.call("rie.dialogue.keyword.clear")
110             question = "Is dit de juiste kleur?"
111             yield session.call("rie.dialogue.keyword.add",
112                 keywords=["ja", "nee"])
113         elif (self.Language_settings == 2): #Engelse settings
114             yield session.call("rie.dialogue.keyword.clear")
115             question = "Is this the right colour?"
116             yield session.call("rie.dialogue.keyword.add",
117                 keywords=["yes", "no"])
118         yield session.call("rie.dialogue.say", text=question)
119         print ("Ik luister nu")
120         session.call("rom.actuator.light.write", mode="linear", frames=[{"time": 100, "data":
                {"body.head.eyes": [50, 50, 50]}}])
121         if state == 4: session.call("rom.actuator.light.write", mode="linear", frames=[{"time"
                : 100, "data": {"body.head.eyes": [250, 0, 0]}}])
122         elif state == 5: session.call("rom.actuator.light.write", mode="linear", frames=[{"
                time": 100, "data": {"body.head.eyes": [0, 250, 0]}}])
123         elif state == 6: session.call("rom.actuator.light.write", mode="linear", frames=[{"
                time": 100, "data": {"body.head.eyes": [0, 0, 250]}}])
124
125         keyword_listener = yield session.subscribe(self.on_keyword,
126             "rie.dialogue.keyword.stream")
127         yield session.call("rie.dialogue.keyword.stream")
128         # answer = input("Was dat goed? [j/n]")
129
130         while self.answer == 0:
131             yield sleep(0.01) # hier moet een timeout die gecancelled kan worden door de event
                    handler
132
```

```
133         if self.answer == 4: #goed
134             reward = 1
135         elif self.answer == 5: # fout
136             reward = -1
137 
138         yield keyword_listener.unsubscribe()
139         yield session.call("rie.dialogue.keyword.close")
140         yield session.call("rie.dialogue.keyword.clear")
141         session.call("rom.actuator.light.write",
142             mode="linear",
143             frames=[{"time": 1000, "data": {"body.head.eyes": [0, 0, 0]}}])
144         return reward
145 
146     def on_keyword(self, frame):
147         print(frame)
148         if ("certainty" in frame["data"]["body"] and frame["data"]["body"]["certainty"] > self
                .certainty and
149                 ((frame["data"]["body"]["text"] == 'rood') or (frame["data"]["body"]["text"]
                        == 'red'))):
150             self.answer = 1
151             print("ik hoorde rood")
152         elif ("certainty" in frame["data"]["body"] and frame["data"]["body"]["certainty"] >
                self.certainty and
153                 ((frame["data"]["body"]["text"] == 'groen') or (frame["data"]["body"]["text"]
                        == 'green'))):
154             # sess.call("rie.dialogue.say", text= "groen")
155             self.answer = 2
156             print("ik hoorde groen")
157         elif ("certainty" in frame["data"]["body"] and frame["data"]["body"]["certainty"] >
                self.certainty and
158                 ((frame["data"]["body"]["text"] == 'blauw') or (frame["data"]["body"]["text"]
                        == 'blue'))):
159             # sess.call("rie.dialogue.say", text= "blauw")
160             self.answer = 3
161             print("ik hoorde blauw")
162         elif ("certainty" in frame["data"]["body"] and frame["data"]["body"]["certainty"] >
                self.certainty and
163                 ((frame["data"]["body"]["text"] == 'yes') or (frame["data"]["body"]["text"] ==
                         'ja'))):
164             self.answer = 4
165             print("ik hoorde dat het goed was")
166         elif ("certainty" in frame["data"]["body"] and frame["data"]["body"]["certainty"] >
                self.certainty and
167                 ((frame["data"]["body"]["text"] == 'no') or (frame["data"]["body"]["text"] ==
                        'nee'))):
168             self.answer = 5
169             print("ik hoorde dat het fout was")
170 
171     @inlineCallbacks
172     def ask(self, session):
173         """ performing action "ask", in which the agent will ask the user what the next task
                should be
174 
175         Params
176         ======
177 
178         Returns
179         =======
180         - answer: the task that the user has given the robot. 1 for task "red", 2 for task "
                green", and 3 for task "blue"
181         """
182         # global waiting
183         self.sess = session
184         self.answer = 0
185         self.certainty = 0.45
186 
187         if (self.Language_settings == 1): #Nederlandse settings
188             yield session.call("rie.dialogue.keyword.clear")
189             if self.TimesAsked < 8: question = "Welke kleur zou ik mijn ogen moeten maken,
                    rood, groen of blauw?"
190             else: question = "Welke kleur zou ik mijn ogen moeten maken?"
```

```
191             yield session.call("rie.dialogue.say", text=question)
192             yield session.call("rie.dialogue.keyword.add",
193                 keywords=["rood", "blauw", "groen"])
194         elif (self.Language_settings == 2): #Engelse settings
195             yield session.call("rie.dialogue.keyword.clear")
196             if self.TimesAsked < 4: question = "To what colour should I change my eyes, red,
                    green, or blue?"
197             else: question = "To what colour should I change my eyes?"
198             yield session.call("rie.dialogue.say", text=question)
199             yield session.call("rie.dialogue.keyword.add",
200                 keywords=["red", "blue", "green"])
201 
202         task_listener = yield session.subscribe(self.on_keyword,
203             "rie.dialogue.keyword.stream")
204         yield session.call("rie.dialogue.keyword.stream")
205         print ("Ik luister nu")
206         session.call("rom.actuator.light.write", mode="linear", frames=[{"time": 100, "data":
                {"body.head.eyes": [50, 50, 50]}}])
207 
208         while self.answer == 0:
209             yield sleep(0.01)
210 
211         yield task_listener.unsubscribe()
212         yield session.call("rie.dialogue.keyword.close")
213         yield session.call("rie.dialogue.keyword.clear")
214         self.TimesAsked += 1
215         session.call("rom.actuator.light.write", mode="linear", frames=[{"time": 100, "data":
                {"body.head.eyes": [0, 0, 0]}}])
216         return self.answer
217 
218     @inlineCallbacks
219     def perform(self, session, action): #, session
220         """ perform the action
221 
222         Params
223         ======
224         - action: the chosen action that has to be performed
225 
226         Returns
227         =======
228         - answer: in the scenario in which the task is "ask", the agent asks which task it has
                to perform,
229                 this will return 1 for task "red", 2 for task "green", and 3 for task "
                    blue"
230         """
231         if (action == 0):
232             answer = yield self.ask(session)
233             return answer
234         elif (action == 1):
235         # maak ogen rood
236             session.call("rom.actuator.light.write", mode="linear",
237                 frames=[{"time": 1000, "data": {"body.head.eyes": [255, 0, 0]}}])
238             print("ik voer nu taak rood uit")
239         elif (action == 2):
240         # maak ogen groen
241             session.call("rom.actuator.light.write", mode="linear",
242                 frames=[{"time": 1000, "data": {"body.head.eyes": [0, 255,0]}}])
243             print("ik voer nu taak groen uit")
244         elif (action == 3):
245         # maak ogen blauw
246             session.call("rom.actuator.light.write", mode="linear",
247                 frames=[{"time": 1000, "data": {"body.head.eyes": [0, 0, 255]}}])
248             print("ik voer nu taak blauw uit")
```

The robot code

```
from autobahn.twisted.component import Component, run
from twisted.internet.defer import inlineCallbacks
from autobahn.twisted.util import sleep
import random

class Robot:
    def __init__(
        self,
        language_settings,
        user_name,
        EMOTIONS,
        REASONS
    ):
        self.language_settings  = language_settings
        self.user_name          = user_name
        self.EMOTIONS           = EMOTIONS
        self.REASONS            = REASONS
        self.lidwoord           = {0:"dit", 1:"dat", 2:{"this"}, 3:{"that"}}
        # statements for Dutch
        self.makkelijk2 = {0:"Ik heb er zin in", 1: "Kom maar op!", 2: "Dat gaat wel weer goed komen"}
        self.lastig2 = {0: "Oei dit vind ik spannend", 1: "O nee dit gaat vast niet.", 2: "O nee, dit gaat fout"}
        self.succesvol = {0: "hoera", 1: "Jippie", 2: "Wat fijn"}
        self.mislukt = {0: "Drommels", 1: "Helaas", 2: "Wat jammer"}
        self.neutraal = {0: "Oké", 1: "Bedankt", 2: "Prima"}
        self.makkelijk1 = {0:"Ik heb er zin in", 1: "Kom maar op!", 2: "Dit gaat wel goed komen"}
        self.lastig1 = {0: "Oei dat vind ik spannend", 1: "O nee dat gaat vast niet.", 2: "O nee, dit gaat fout"}
        self.taak = {0: "vragen", 1: "rood", 2: "groen", 3: "blauw "}

        # Statements for English
        self.easy1 = {0:"I am looking forward to this", 1: "Let's go!", 2: "Okay. Let's go"}
        self.tough1 = {0: "This is a bit scary for me", 1: "O no, it will go wrong again", 2: "oh no, it will go wrong"}
        self.succesful = {0: "hooray", 1: "Nice", 2: "Lovely"}
        self.failed = {0: "O bother", 1: "Let's pretend that did not happen", 2: "How unfortunate"}
        self.neutral = {0: "Okay", 1: "Thank you", 2: "Check"}
        self.easy2 = {0:"I am looking forward to this", 1: "Let's go!", 2: "Okay. Let's go"}
        self.tough2 = {0: "This is a bit scary for me", 1: "O no, it will go wrong again", 2: "oh no, it will go wrong"}
        self.task = {0: "ask", 1: "red", 2: "green", 3: "blue"}

    @inlineCallbacks
    def startup(self, session):
        yield session.call("rom.optional.behavior.play",
            name="BlocklyStand")
        yield session.call("rom.actuator.light.write",
            mode="linear",
        frames=[{"time": 1000, "data": {"body.head.eyes": [50, 50, 50]}}])
        if (self.language_settings == 1): # Nederlands
            # Zet de taal naar Nederlands:
            yield session.call("rie.dialogue.keyword.language", lang="nl")
            yield session.call("rie.dialogue.config.language", lang="nl")
            session.call("rie.dialogue.say",
                text=f"Hallo! Leuk dat je me wil helpen met kleuren leren! Laten we aan de slag gaan.")
        elif (self.language_settings == 2): #Engels
            # Zet de taal naar Engels:
            yield session.call("rie.dialogue.keyword.language", lang="en")
            yield session.call("rie.dialogue.config.language", lang="en")
            session.call("rie.dialogue.say",
                text=f"Hello! Thank you for helping me learn colours. Let's start!")
        yield session.call("rom.optional.behavior.play", name="BlocklyWaveRightArm")

    @inlineCallbacks
    def handsOpen(self, session):
```

```
62          # being sad
63          yield session.call("rom.actuator.motor.write",
64              mode="linear",
65              frames=[
66                  {"time":  1500, "data": {"body.arms.right.hand": "Open"}},
67                  {"time":  1500, "data": {"body.arms.left.hand": "Open"}}
68                      ],
69              force=True)
70
71      @inlineCallbacks
72      def stand(self, session):
73          yield session.call("rom.optional.behavior.play",
74              name="BlocklyStand")
75          # session.leave() # Sluit de verbinding met de robot
76
77      @inlineCallbacks
78      def happy(self, session):
79          print ("I am happy")
80          versie = random.randint(0, 2)
81          if (self.language_settings == 1): session.call("rie.dialogue.say", text=self.succesvol
                [versie])
82          else: session.call("rie.dialogue.say", text=self.succesful[versie])
83          yield session.call("rom.actuator.motor.write",
84              mode="linear",
85              frames=[
86                  {"time":  1500, "data": {"body.head.pitch": -0.35}},
87                  {"time":  1500, "data": {"body.legs.right.upper.pitch": 0.01}},
88                  {"time":  1500, "data": {"body.legs.left.upper.pitch": 0.01}},
89                  {"time":  1500, "data": {"body.legs.right.lower.pitch": 0.45}},
90                  {"time":  1500, "data": {"body.legs.left.lower.pitch": 0.45}},
91                  {"time":  1500, "data": {"body.arms.right.upper.roll": -0.7}},
92                  {"time":  1500, "data": {"body.arms.left.upper.roll": 0.7}},
93                  {"time":  1500, "data": {"body.arms.right.upper.pitch": -1.1}},
94                  {"time":  1500, "data": {"body.arms.left.upper.pitch": -1.1}},
95                  {"time":  1500, "data": {"body.arms.right.lower.roll": -0.75}},
96                  {"time":  1500, "data": {"body.arms.left.lower.roll": 0.75}}
97                      ],
98              force=True)
99
100     def slowstand(self, session):
101         yield session.call("rom.actuator.motor.write",
102             mode="linear",
103             frames=[
104                 {"time":  1500, "data": {"body.head.pitch": 0.05}},
105                 {"time":  1500, "data": {"body.legs.right.upper.pitch": -0.1}},
106                 {"time":  1500, "data": {"body.legs.left.upper.pitch": -0.1}},
107                 {"time":  1500, "data": {"body.legs.right.lower.pitch": 0.55}},
108                 {"time":  1500, "data": {"body.legs.left.lower.pitch": 0.55}},
109                 {"time":  1500, "data": {"body.arms.right.upper.roll": -0.25}},
110                 {"time":  1500, "data": {"body.arms.left.upper.roll": 0.25}},
111                 {"time":  1500, "data": {"body.arms.right.upper.pitch": 1.8}},
112                 {"time":  1500, "data": {"body.arms.left.upper.pitch": 1.8 }},
113                 {"time":  1500, "data": {"body.arms.right.lower.roll": 0.5}},
114                 {"time":  1500, "data": {"body.arms.left.lower.roll": -0.5}}
115                     ],
116             force=True)
117         session.call("rom.optional.behavior.play", name="BlocklyStand")
118
119     @inlineCallbacks
120     def sad(self, session):
121         print ("I am sad")
122         # being sad
123         versie = random.randint(0, 2)
124         if (self.language_settings == 1): session.call("rie.dialogue.say", text=self.mislukt[
                versie])
125         else: session.call("rie.dialogue.say", text=self.failed[versie])
126         yield session.call("rom.actuator.motor.write",
127             mode="linear",
128             frames=[
129                 {"time":  1000, "data": {"body.head.pitch": 0.35}},
130                 {"time":  1000, "data": {"body.legs.right.upper.pitch": -0.75}},
```

```
131                 {"time":  1000, "data": {"body.legs.left.upper.pitch": -0.75}},
132                 {"time":  1000, "data": {"body.legs.right.lower.pitch": 1}},
133                 {"time":  1000, "data": {"body.legs.left.lower.pitch": 1}},
134                 {"time":  1000, "data": {"body.arms.right.upper.roll": -0.1}},
135                 {"time":  1000, "data": {"body.arms.left.upper.roll": 0.1}},
136                 {"time":  1000, "data": {"body.arms.right.upper.pitch": 1}},
137                 {"time":  1000, "data": {"body.arms.left.upper.pitch": 1}},
138                 {"time":  1000, "data": {"body.arms.right.lower.roll": -0.9}},
139                 {"time":  1000, "data": {"body.arms.left.lower.roll": 0.9}}
140                     ],
141             force=True)
142         # session.leave() # Sluit de verbinding met de robot
143 
144     @inlineCallbacks
145     def hopefull(self, session, state, reasons):
146         print ("I am hopeful")
147         versie = random.randint(0, 2)
148 
149         # being being hopefull
150         session.call("rom.actuator.motor.write",
151             mode="linear",
152             frames=[
153                 {"time":  1000, "data": {"body.head.pitch": 0.05}},
154                 {"time":  1000, "data": {"body.arms.right.upper.roll": -0.1}},
155                 {"time":  1000, "data": {"body.arms.left.upper.roll": 0.1}},
156                 {"time":  1000, "data": {"body.arms.right.upper.pitch": 1.25}},
157                 {"time":  1000, "data": {"body.arms.left.upper.pitch": 1.25}},
158                 {"time":  1000, "data": {"body.arms.right.lower.roll": 2}},
159                 {"time":  1000, "data": {"body.arms.left.lower.roll": -2}},
160                                 {"time":  1500, "data": {"body.legs.right.upper.pitch":
                                        -0.1}},
161                 {"time":  1500, "data": {"body.legs.left.upper.pitch": -0.1}},
162                 {"time":  1500, "data": {"body.legs.right.lower.pitch": 0.55}},
163                 {"time":  1500, "data": {"body.legs.left.lower.pitch": 0.55}}
164                     ],
165             force=True)
166         if state == 0:
167             if (self.language_settings == 1):
168                 yield session.call("rie.dialogue.say", text=self.makkelijk1[versie])
169                 if self.REASONS:
170                     yield session.call("rie.dialogue.say", text=f"Ik hoop dat het {self.taak[
                            reasons]} wordt!")
171 
172             else:
173                 yield session.call("rie.dialogue.say", text=self.easy1[versie])
174                 if self.REASONS:
175                     yield session.call("rie.dialogue.say", text=f"I am hoping for {self.task[
                            reasons]}!")
176         else:
177             if (self.language_settings == 1): yield session.call("rie.dialogue.say", text=self
                    .makkelijk2[versie])
178             else: yield session.call("rie.dialogue.say", text=self.easy2[versie])
179 
180         # session.leave() # Sluit de verbinding met de robot
181 
182     @inlineCallbacks
183     def fear(self, session, state, reasons):
184         print ("I am afraid")
185         # being afraid
186         session.call("rom.actuator.motor.write",
187             mode="linear",
188             frames=[
189                 {"time": 1500, "data": {"body.head.pitch": 0.35}},
190                                 {"time":  1500, "data": {"body.legs.right.upper.pitch":
                                        -0.2}},
191                 {"time":  1500, "data": {"body.legs.left.upper.pitch": -0.2}},
192                 {"time":  1500, "data": {"body.legs.right.lower.pitch": 0.65}},
193                 {"time":  1500, "data": {"body.legs.left.lower.pitch": 0.65}},
194                 {"time": 1500, "data": {"body.arms.right.upper.roll": -0.35}},
195                 {"time": 1500, "data": {"body.arms.left.upper.roll": 0.35}},
196                 {"time": 1500, "data": {"body.arms.right.upper.pitch": -0.5}},
```

```
197                 {"time": 1500, "data": {"body.arms.left.upper.pitch": -0.5}},
198                 {"time": 1500, "data": {"body.arms.right.lower.roll": 2}},
199                 {"time": 1500, "data": {"body.arms.left.lower.roll": -2}},
200                 {"time": 1500, "data": {"body.arms.right.lower.yaw": -0.1}},
201                 {"time": 1500, "data": {"body.arms.left.lower.yaw": 0.1}},
202                 {"time": 1500, "data": {"body.arms.right.hand.yaw": 2}},
203                 {"time": 1500, "data": {"body.arms.left.hand.yaw": -2}}
204                     ],
205             force=True)
206         versie = random.randint(0, 2)
207         if state == 0:
208             if (self.language_settings == 1):
209                 yield session.call("rie.dialogue.say", text=self.lastig1[versie])
210                 if self.REASONS:
211                     yield session.call("rie.dialogue.say", text=f"Ik vrees dat het {self.taak[
                            reasons]} wordt.")
212 
213             else:
214                 yield session.call("rie.dialogue.say", text=self.tough1[versie])
215                 if self.REASONS:
216                     yield session.call("rie.dialogue.say", text=f"I am afraid it will be {self
                            .task[reasons]}.")
217         else:
218             if (self.language_settings == 1): session.call("rie.dialogue.say", text=self.
                    lastig2[versie])
219             else: session.call("rie.dialogue.say", text=self.tough2[versie])
220 
221         # session.leave() # Sluit de verbinding met de robot
222 
223     @inlineCallbacks
224     def no_emotions(self, session):
225         print ("I received feedback")
226         versie = random.randint(0, 2)
227         if (self.language_settings == 1): yield session.call("rie.dialogue.say", text=self.
                neutraal[versie])
228         else: yield session.call("rie.dialogue.say", text=self.neutral[versie])
229         # session.leave() # Sluit de verbinding met de robot
230 
231     @inlineCallbacks
232     def end(self, session):
233         yield session.call("rom.optional.behavior.play",
234             name="BlocklyStand")
235         if (self.language_settings == 1): # Nederlands
236             session.call("rie.dialogue.say",
237                 text=f"Jippie! Ik heb nu de kleuren geleerd. Dank je wel!")
238         elif (self.language_settings == 2): #Engels
239             session.call("rie.dialogue.say",
240                 text=f"Hoeray! I have learned the colours! Thank you!")
241         yield session.call("rom.optional.behavior.play", name="BlocklyWaveRightArm")
242         yield session.call("rom.optional.behavior.play", name="BlocklyStand")
243 
244     @inlineCallbacks
245     def colourLearned(self, session, state):
246         yield session.call("rom.optional.behavior.play", name="BlocklyStand")
247         if state == 1: session.call("rom.actuator.light.write", mode="linear", frames=[{"time"
                : 100, "data": {"body.head.eyes": [250, 0, 0]}}])
248         elif state == 2: session.call("rom.actuator.light.write", mode="linear", frames=[{"
                time": 100, "data": {"body.head.eyes": [0, 250, 0]}}])
249         elif state == 3: session.call("rom.actuator.light.write", mode="linear", frames=[{"
                time": 100, "data": {"body.head.eyes": [0, 0, 250]}}])
250         if (self.language_settings == 1): # Nederlands
251             yield session.call("rie.dialogue.say",
252                 text=f"Kijk, ik weet nu goed wat {self.taak[state]} is!")
253         elif (self.language_settings == 2): #Engels
254             yield session.call("rie.dialogue.say",
255                 text=f"Look! I know what {self.task[state]} is!")
256         session.call("rom.actuator.light.write", mode="linear", frames=[{"time": 100, "data":
                {"body.head.eyes": [50, 50, 50]}}])
```

C
Scatter plots

Here scatter plots between different variables can be seen. In table C.1 the figure for specific combinations can be found.

Animacy, Anthropomorphism, Attractiveness, Efficiency, Intelligence, Likeability, Novelty, and Stimulation are on a 1 to 5 scale. Inefficient Exploitation is a count for how often an exploitation action before reaching a learning progress score of 3 has been performed. This value can not be lower than 0 and does not have an upper limit. L.P. = 3 location refers to the total number of iterations required to achieve a learning progress score of 3. Very lucky participants could reach this score after3 iterations, and there is no upper limit.

Although some relations can be found for some combinations of variables, no relations with regard to the different robot modes can be found.

	Animacy	Anthropomorphism	Attractiveness	Efficiency	Intelligence	Likeability	Novelty	Stimulation	Inefficient Exploitation	L.P. = 3 location
Animacy	X	figC.1	figC.2	figC.3	figC.4	figC.5	figC.6	figC.7	figC.8	figC.9
Anthropomorphism	figC.1	X	figC.10	figC.11	figC.12	figC.13	figC.14	figC.15	figC.16	figC.17
Attractiveness	figC.2	figC.10	X	figC.18	figC.19	figC.20	figC.21	figC.22	figC.23	figC.24
Efficiency	figC.3	figC.11	figC.18	X	figC.25	figC.26	figC.27	figC.28	figC.29	figC.30
Intelligence	figC.4	figC.12	figC.19	figC.25	X	figC.31	figC.32	figC.33	figC.34	figC.35
Likeability	figC.5	figC.13	figC.20	figC.26	figC.31	X	figC.36	figC.37	figC.38	figC.39
Novelty	figC.6	figC.14	figC.21	figC.27	figC.32	figC.36	X	figC.40	figC.41	figC.42
Stimulation	figC.7	figC.15	figC.22	figC.28	figC.33	figC.37	figC.40	X	figC.43	figC.44
Inefficient Exploitation	figC.8	figC.16	figC.23	figC.29	figC.34	figC.38	figC.41	figC.43	X	figC.45
L.P. = 3 location	figC.9	figC.17	figC.24	figC.30	figC.35	figC.39	figC.42	figC.44	figC.45	X

Table C.1: Post Hoc LSD results between robot modes and ANOVA P-values. Yellow values indicate a significance in a LSD Post Hoc test, but no significance in a Bonferroni Post Hoc Test. Green indicates significance for both Post Hoc tests. All values have been calculated with a univariance general linear model. For all ANOVA's the degrees of freedom were: F(2, 58).

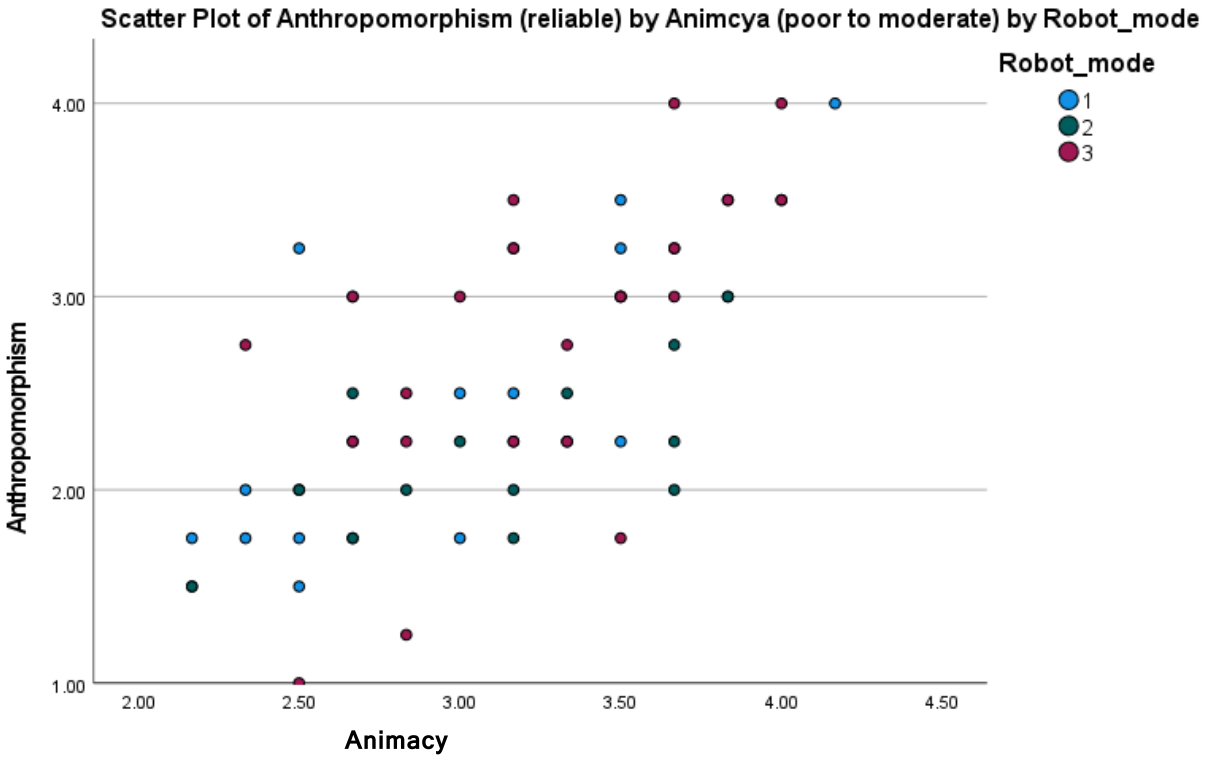


Figure C.1

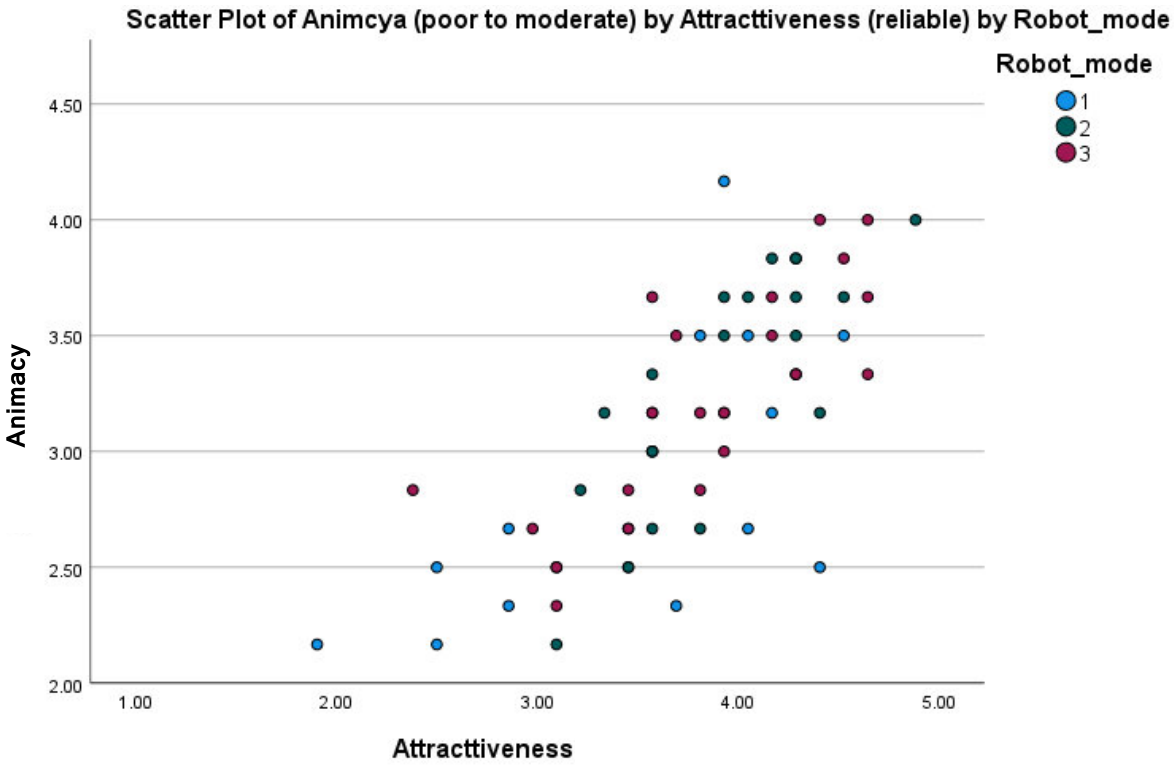


Figure C.2

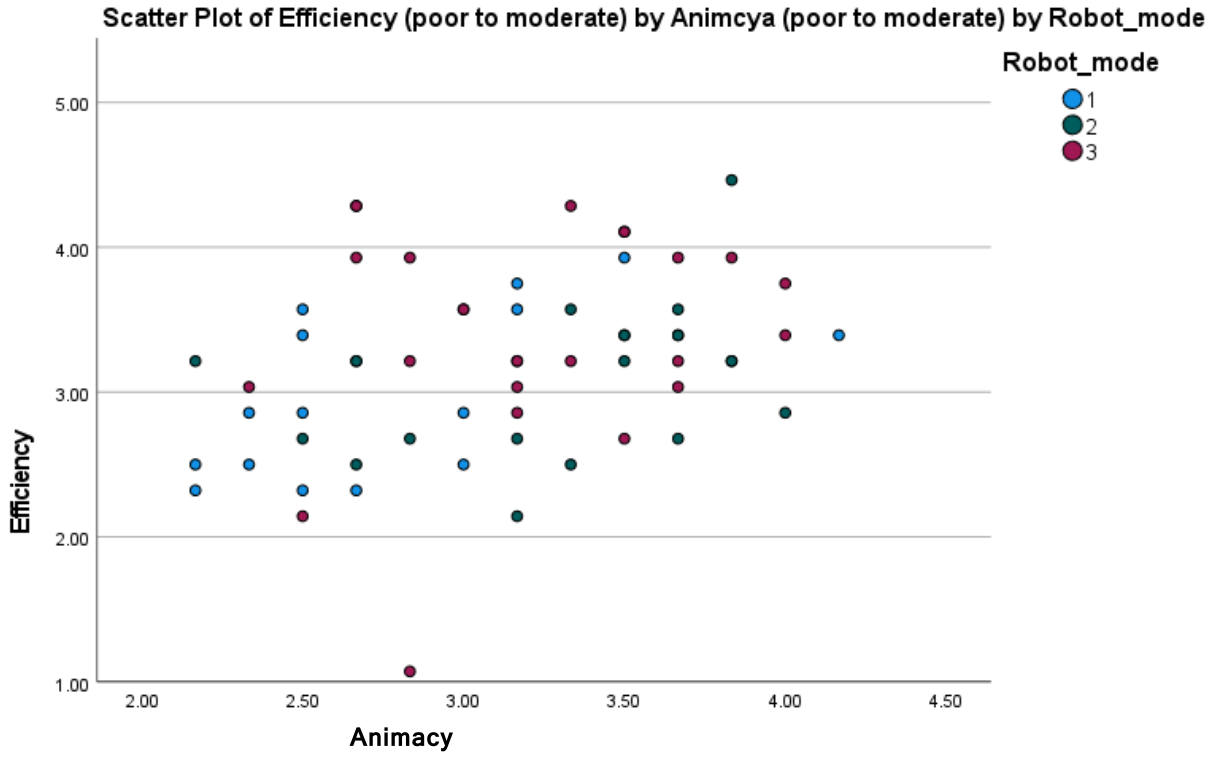


Figure C.3

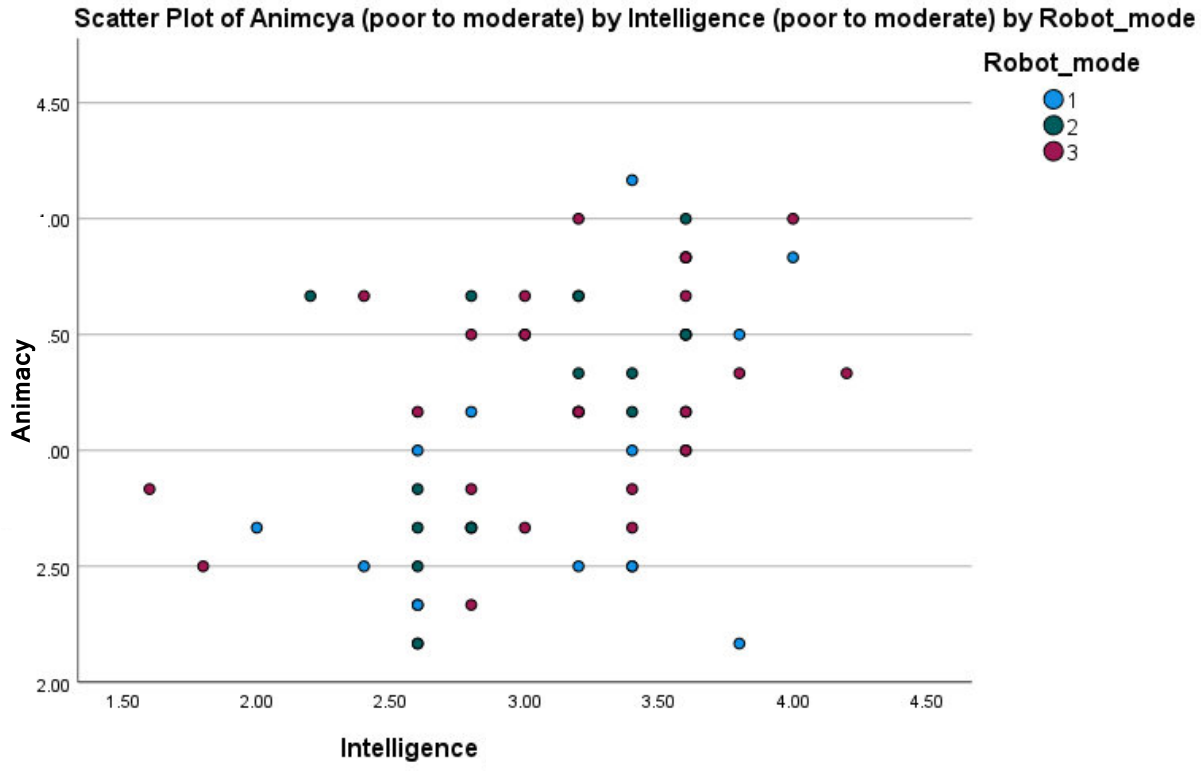


Figure C.4

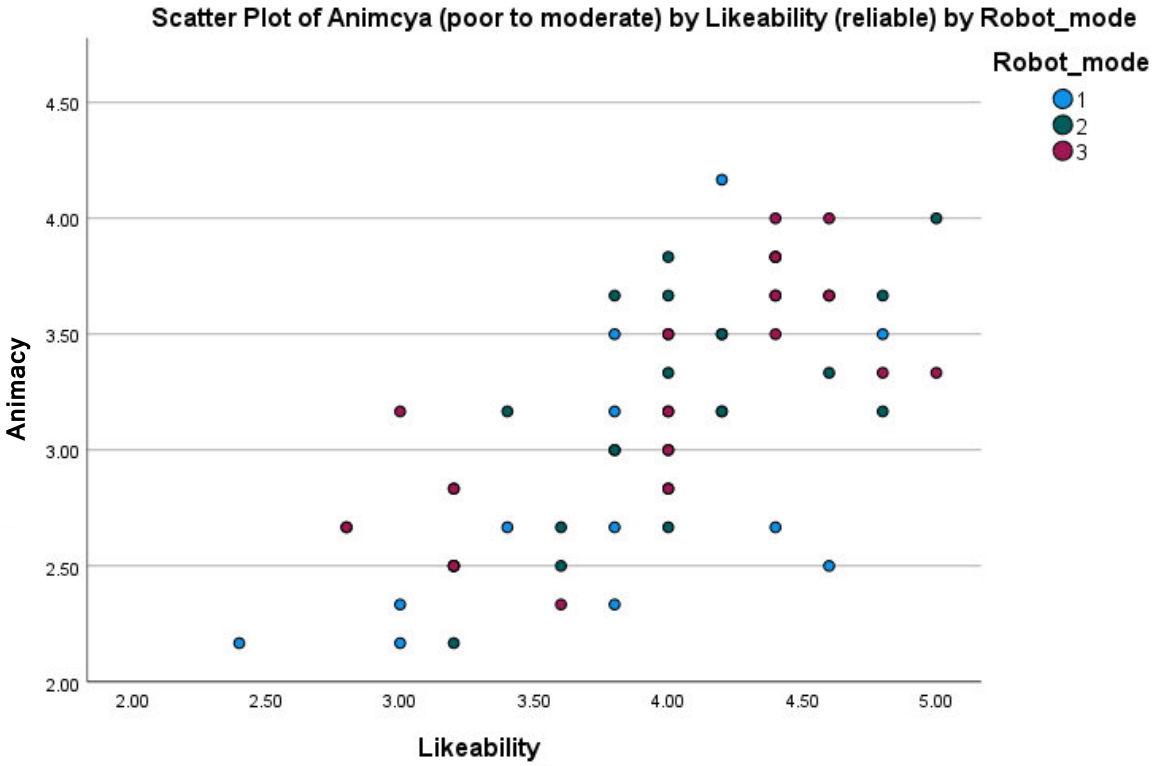


Figure C.5

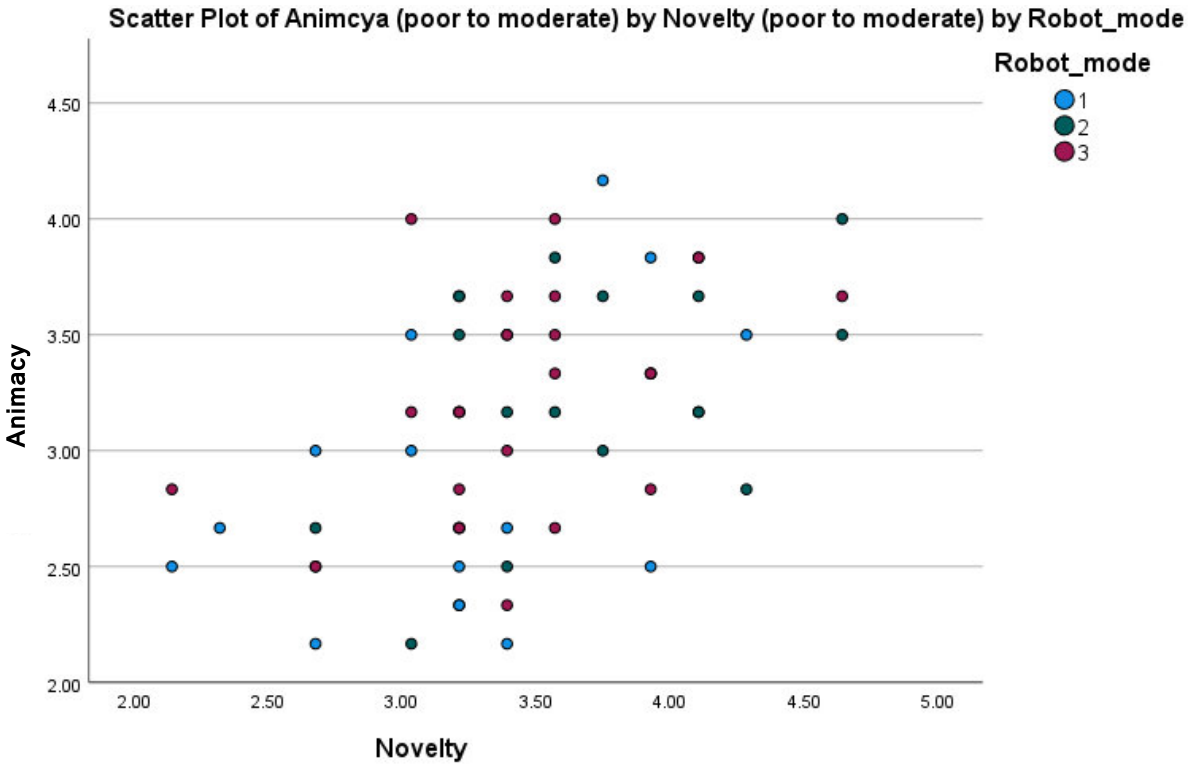


Figure C.6

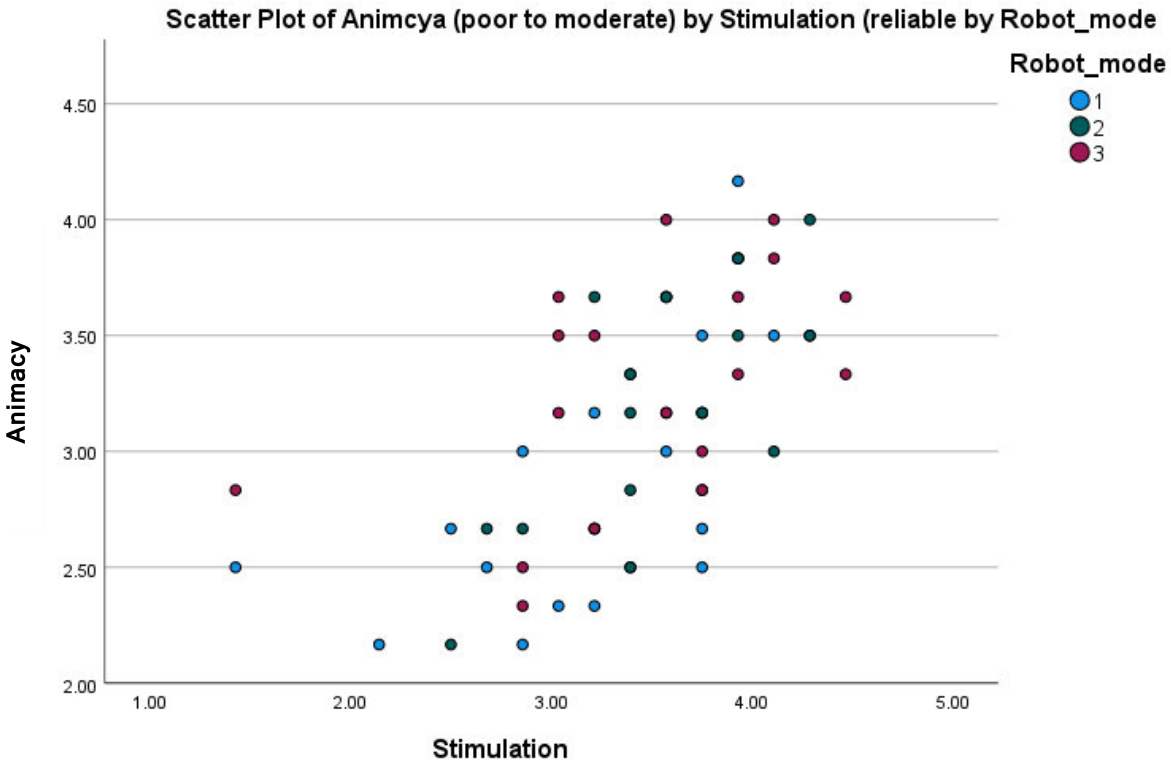


Figure C.7

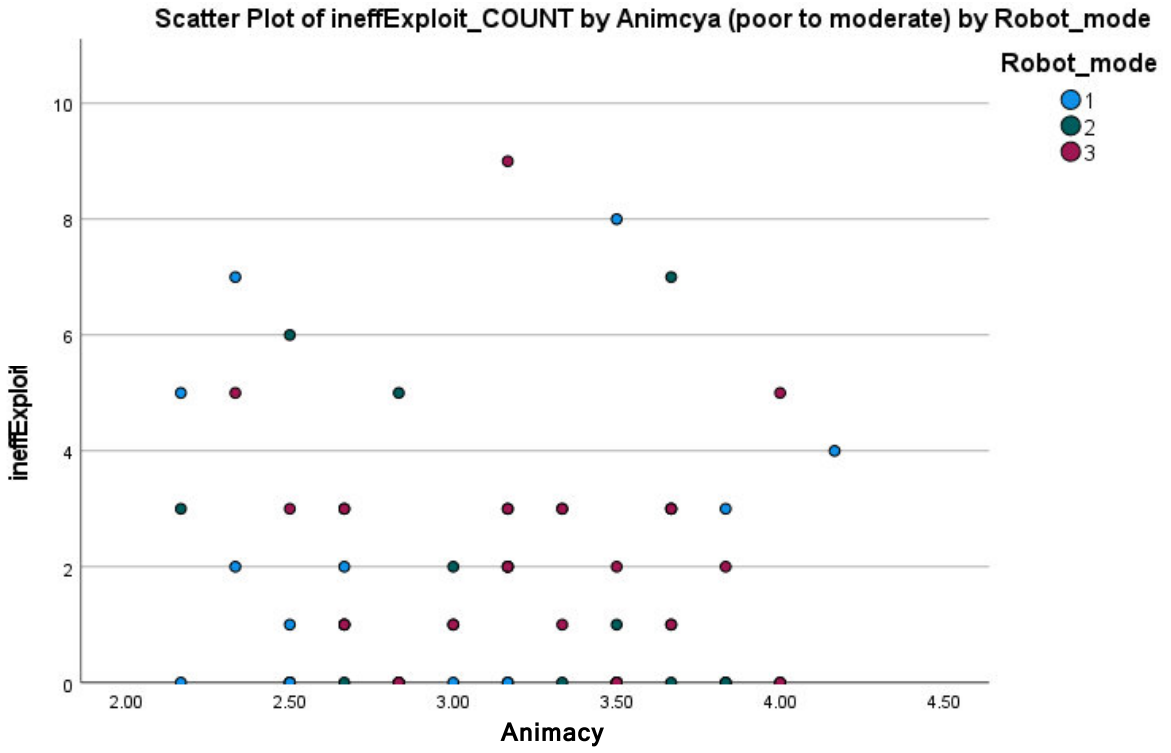


Figure C.8

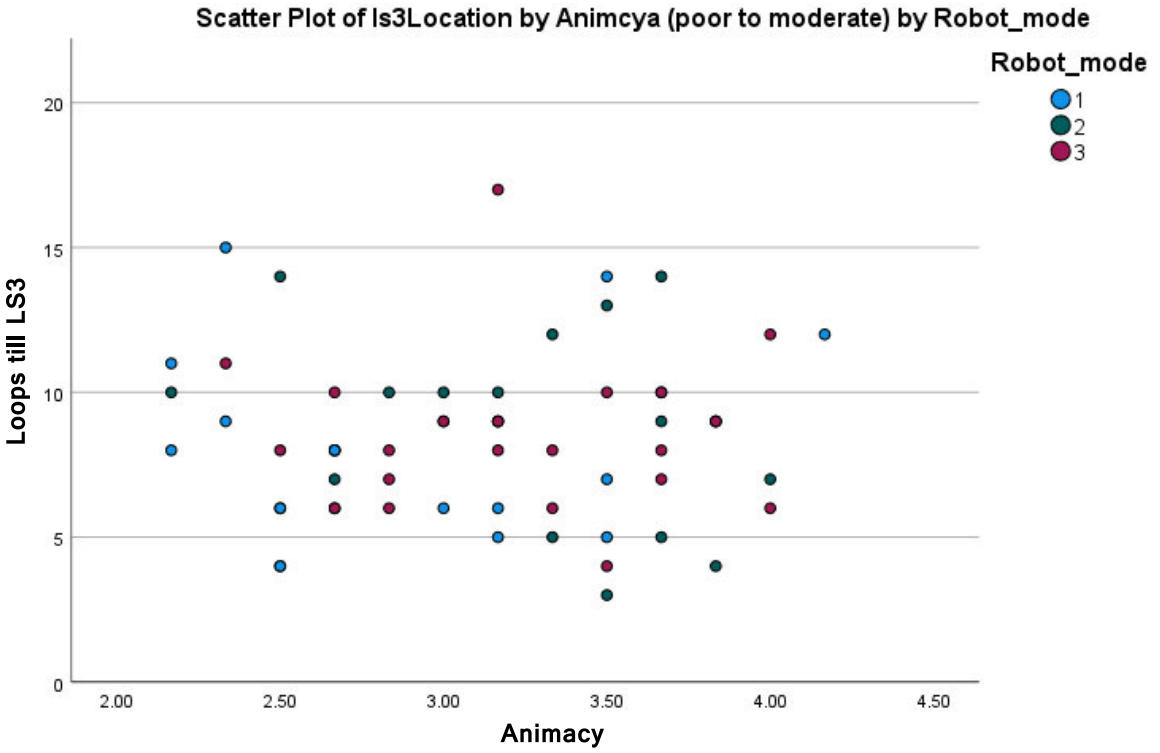


Figure C.9

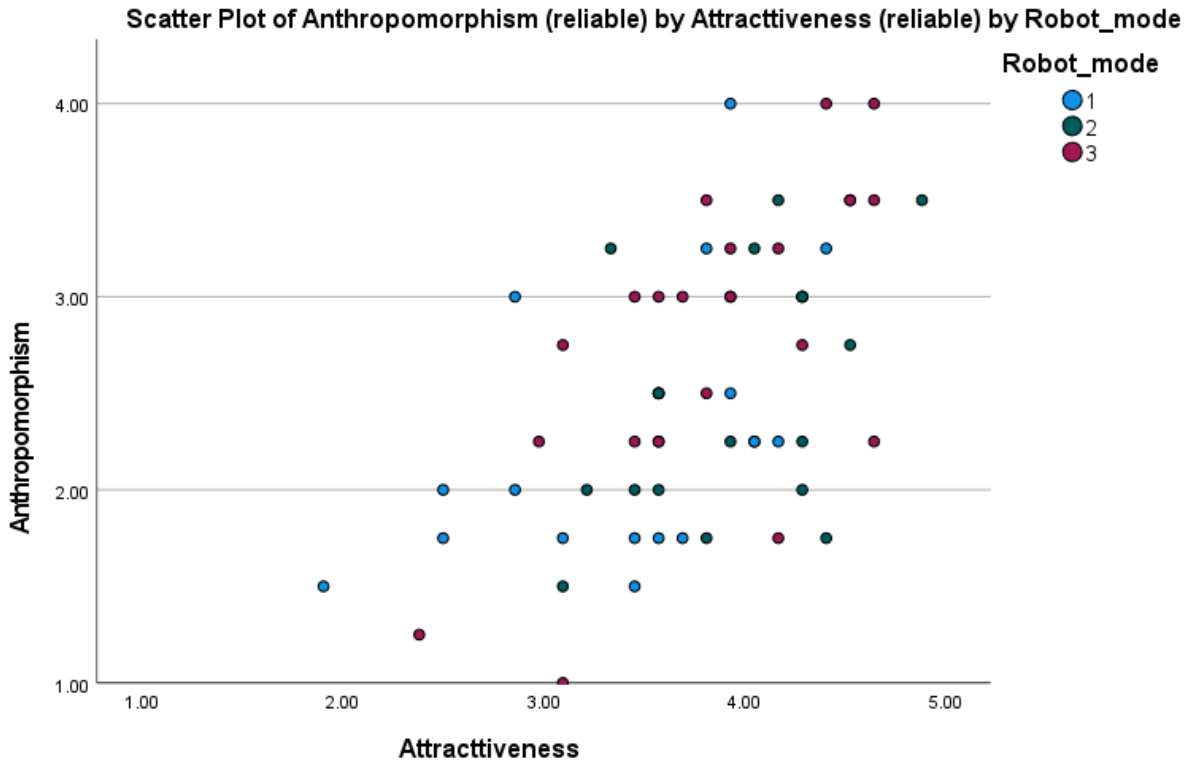


Figure C.10

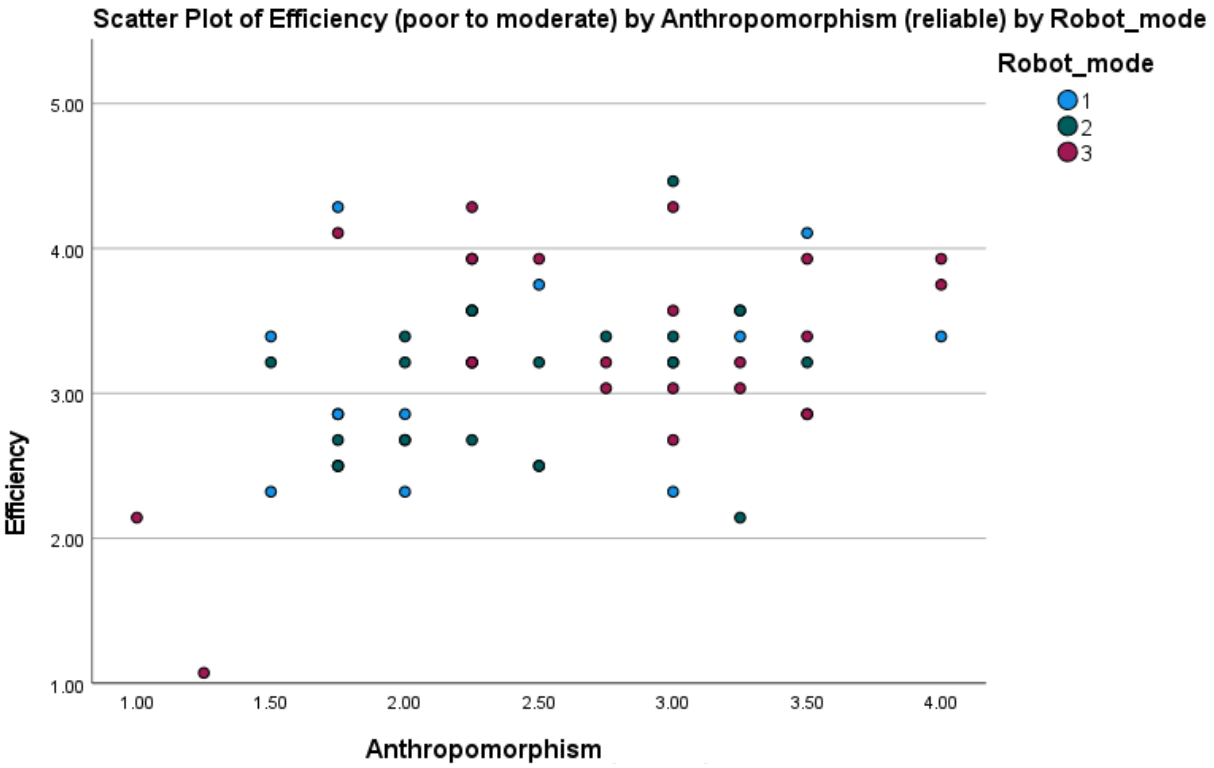


Figure C.11

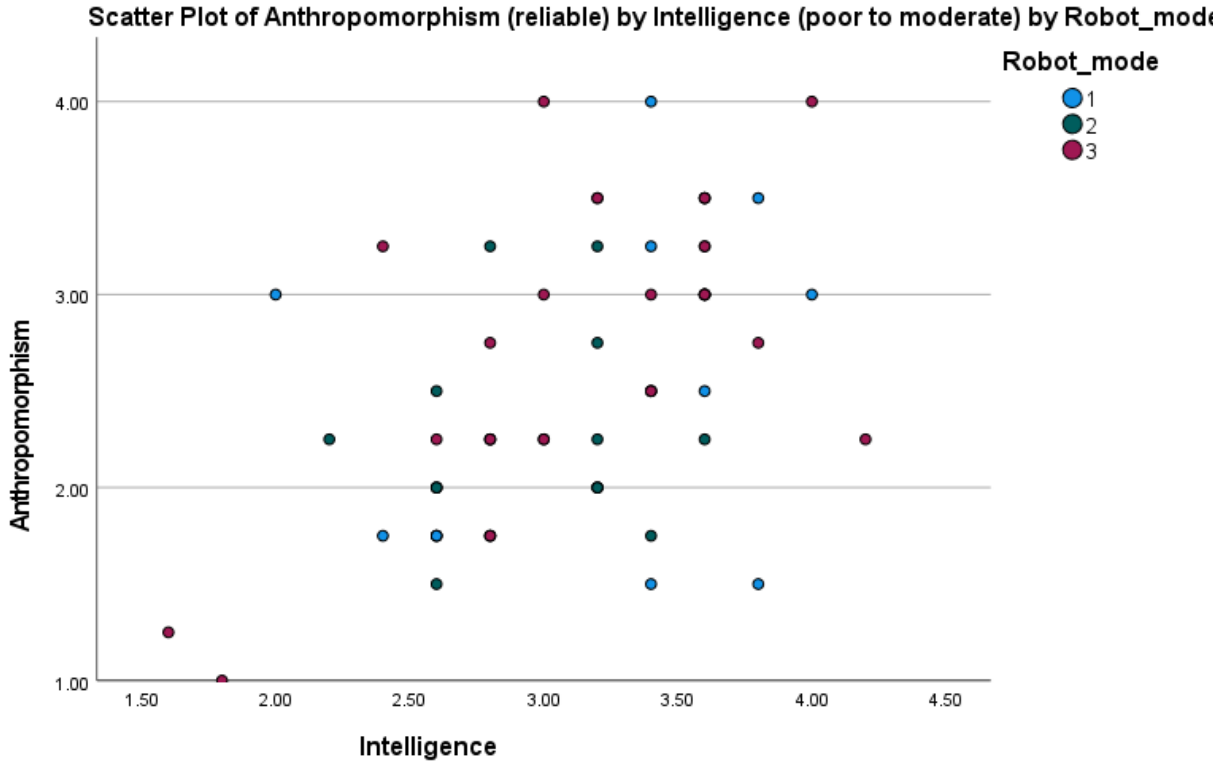


Figure C.12

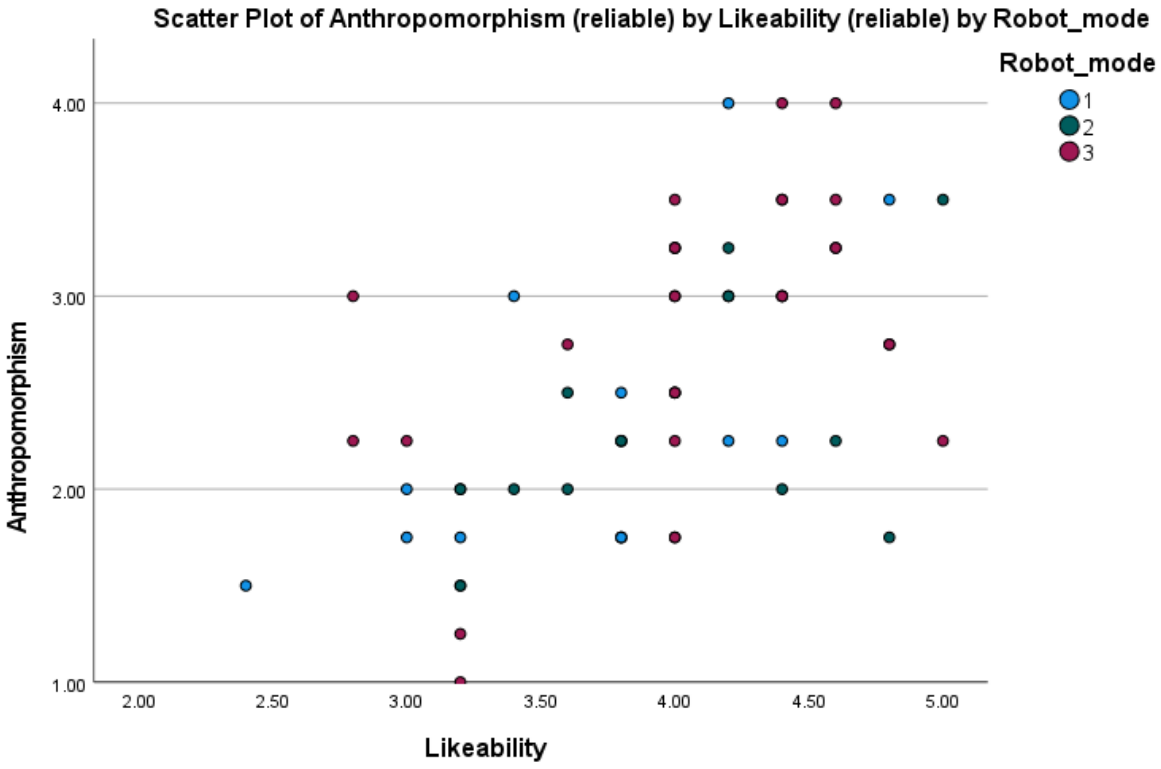


Figure C.13

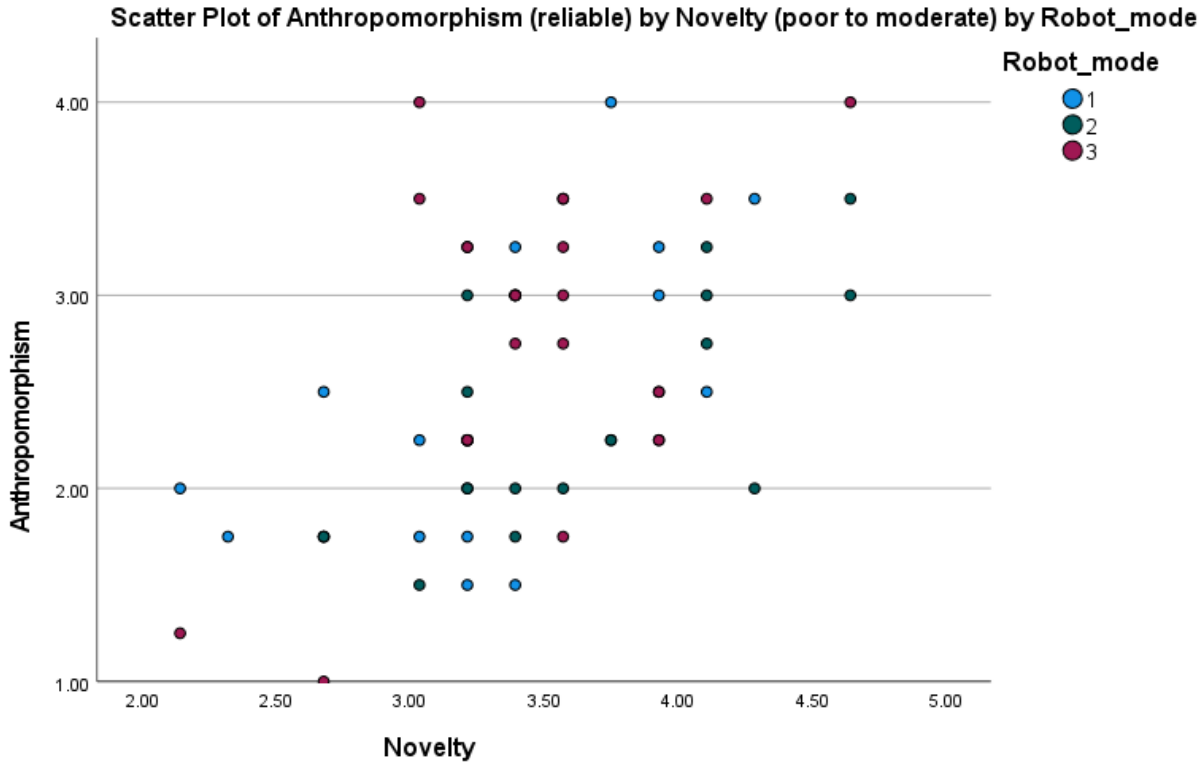


Figure C.14

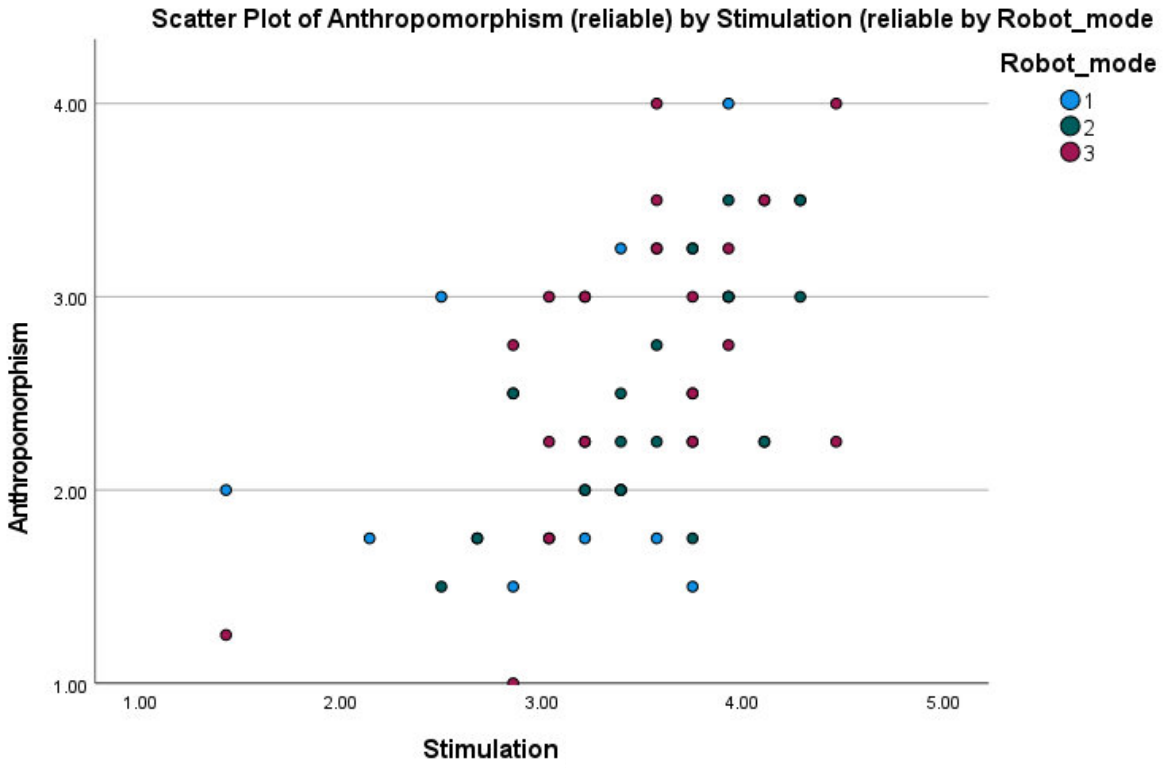


Figure C.15

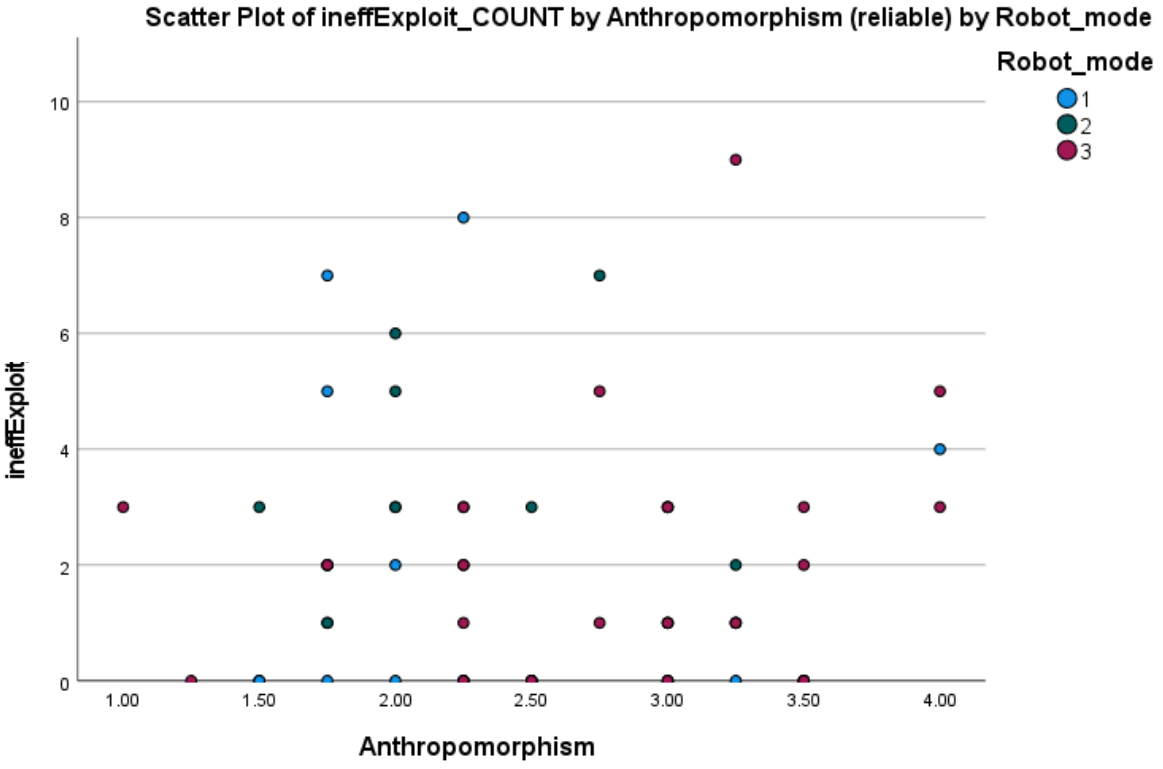


Figure C.16

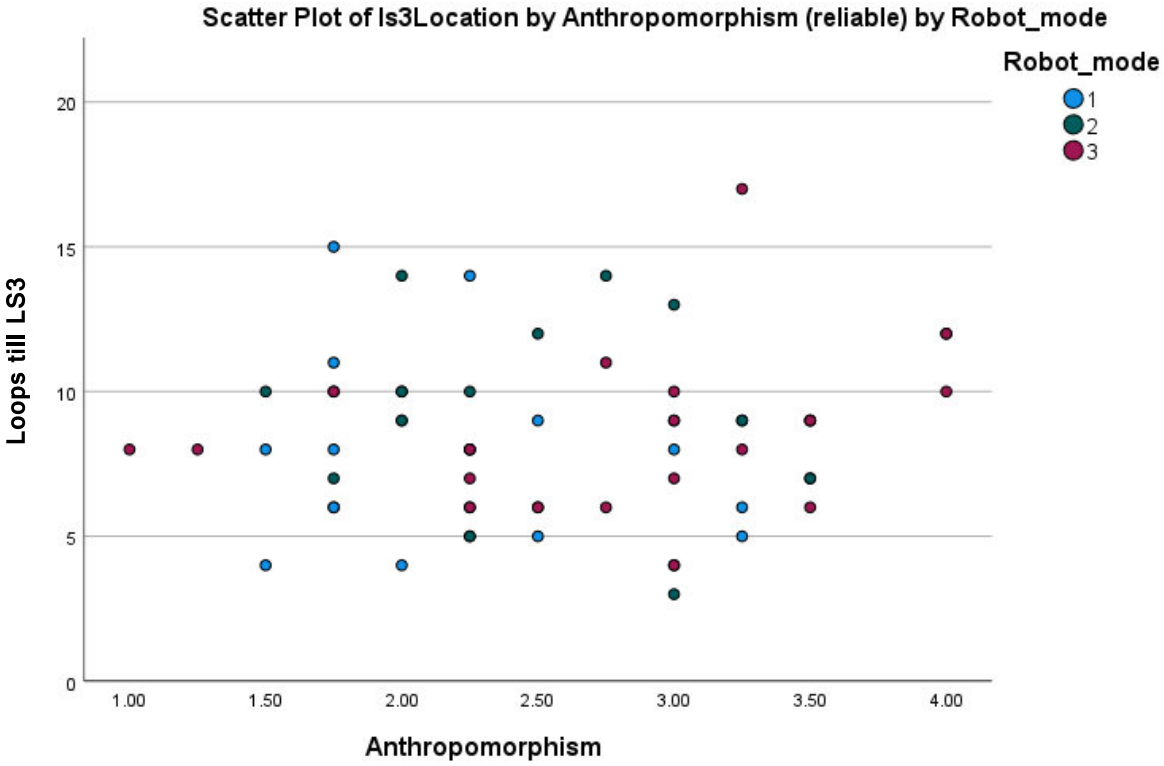


Figure C.17

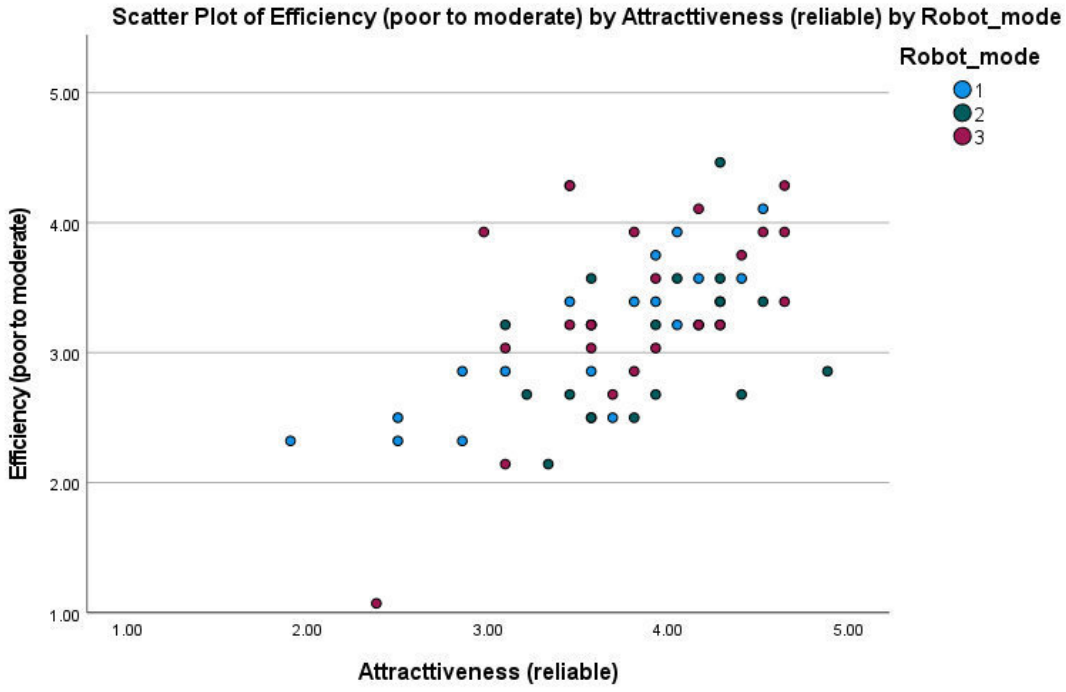


Figure C.18

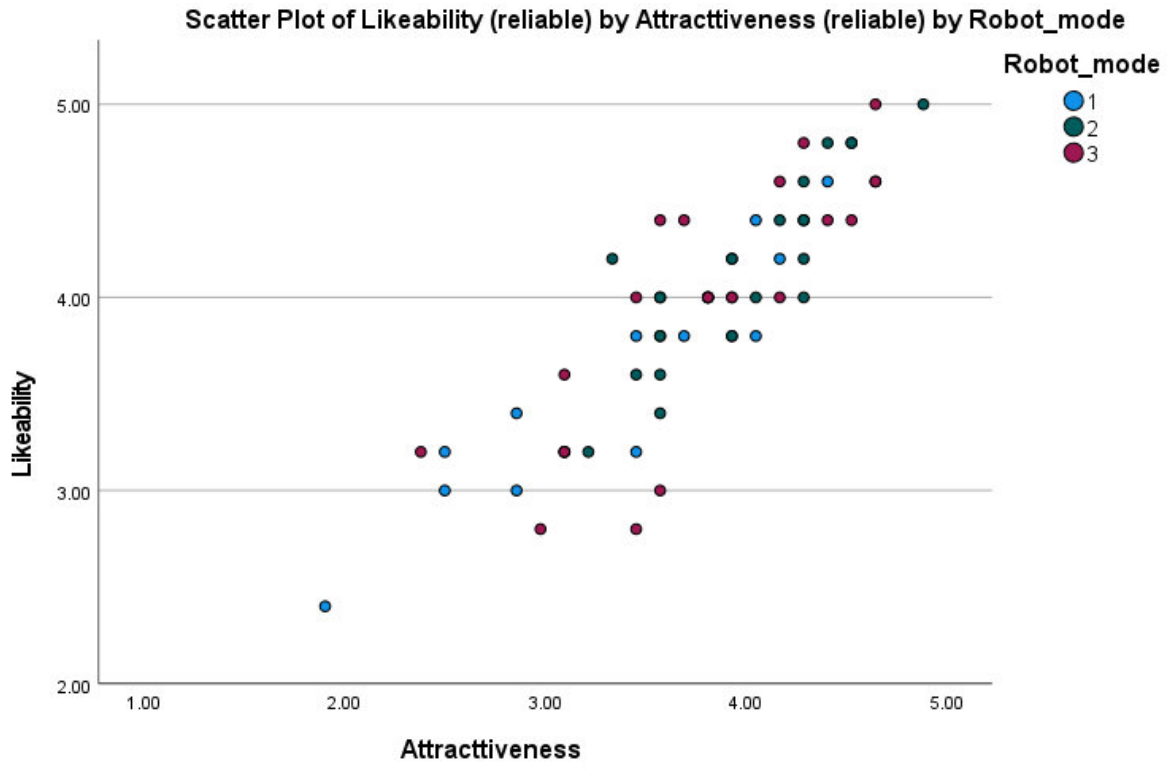


Figure C.19

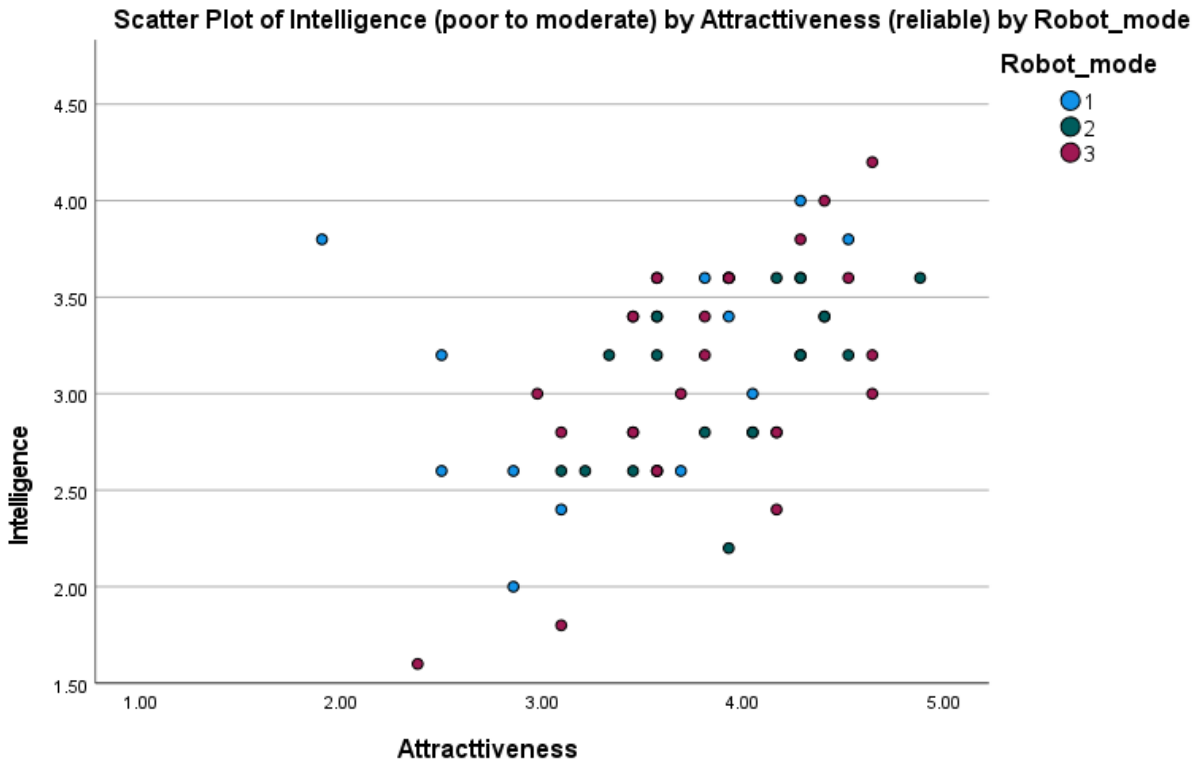


Figure C.20

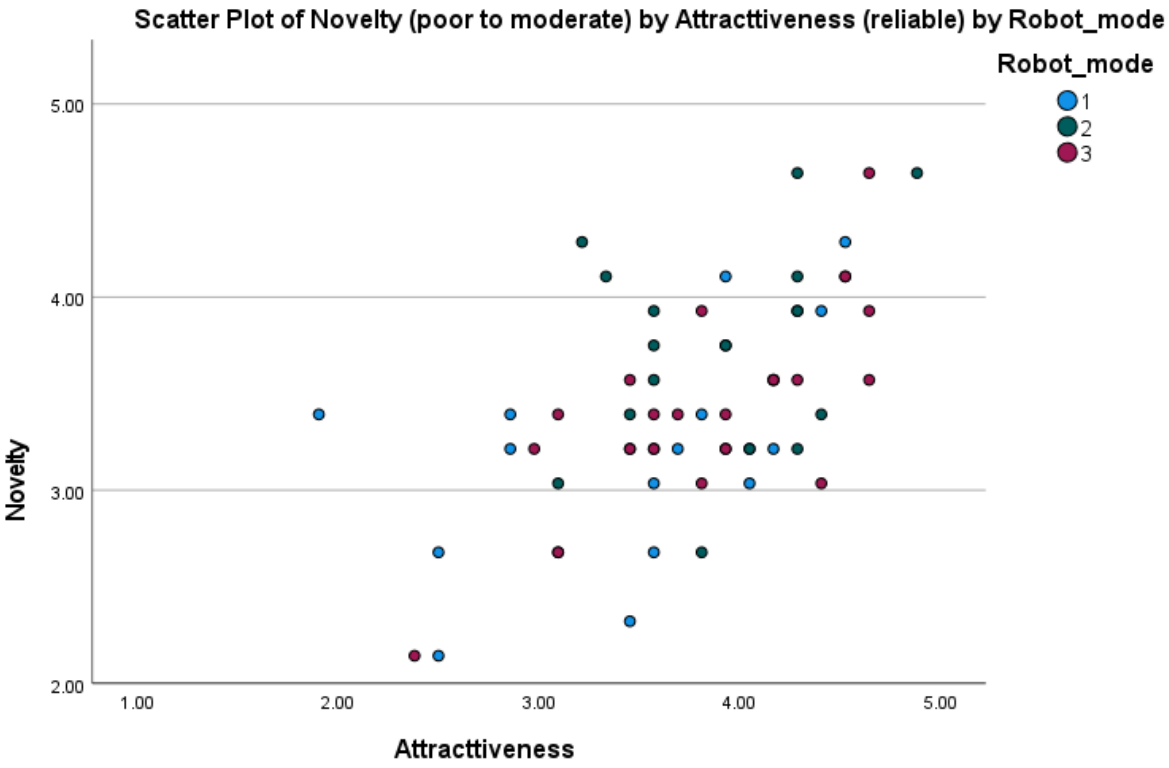


Figure C.21

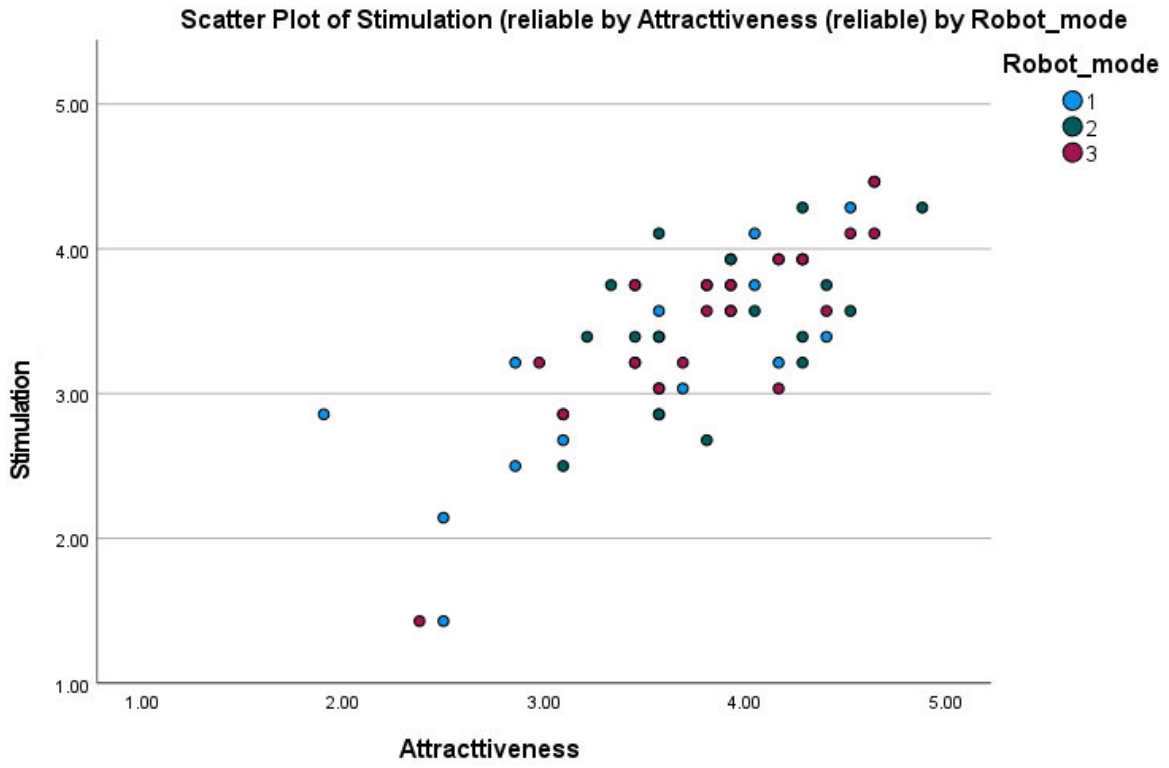


Figure C.22

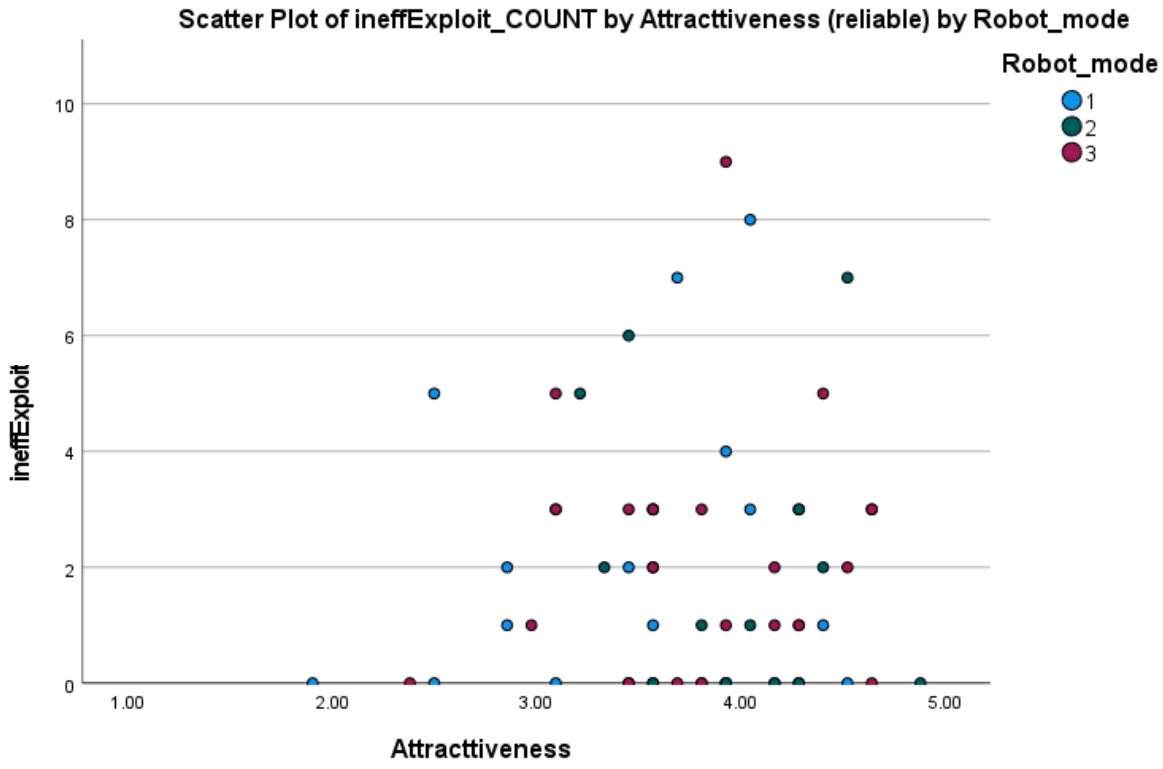


Figure C.23

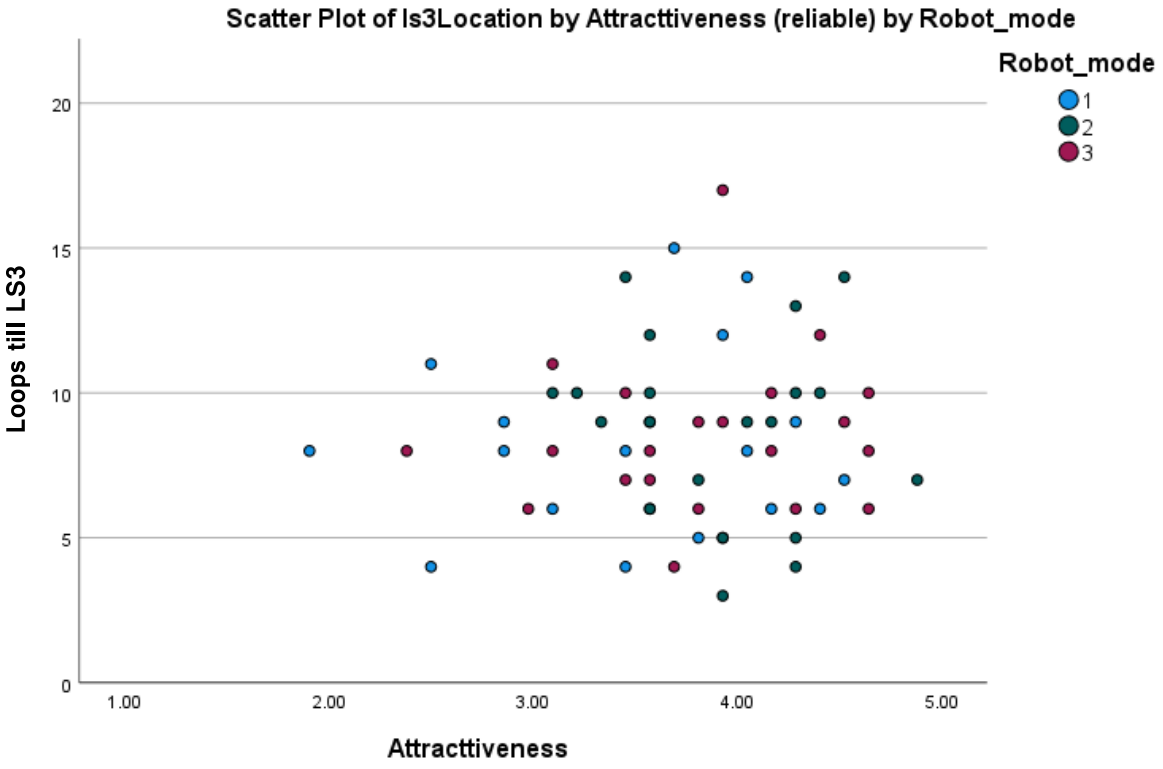


Figure C.24

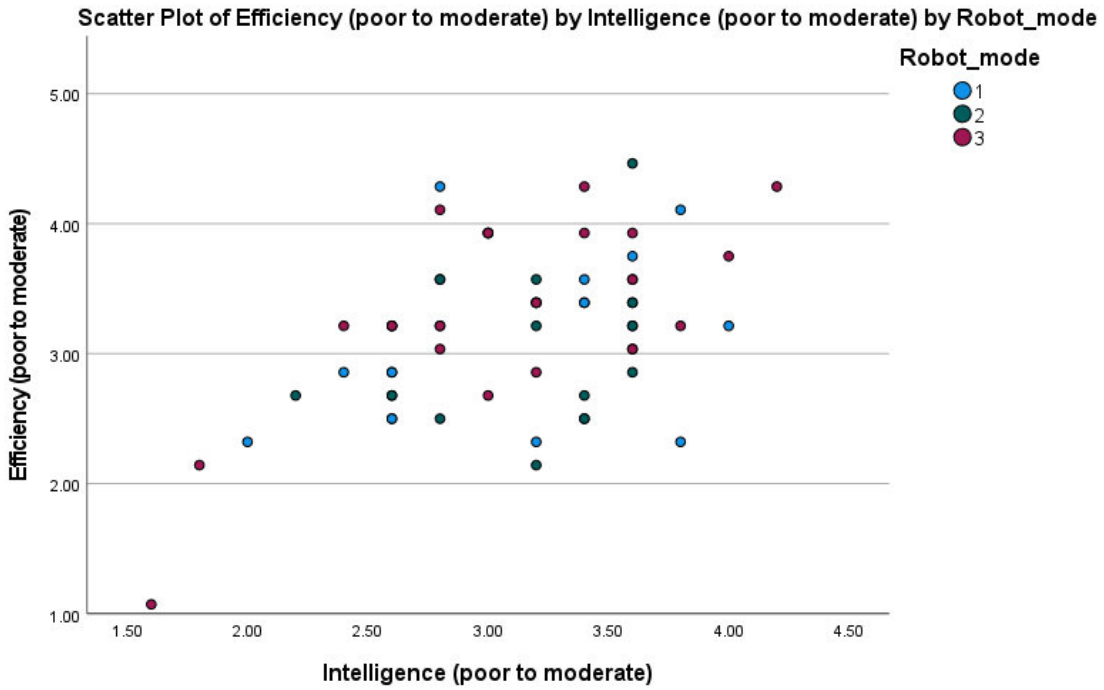


Figure C.25

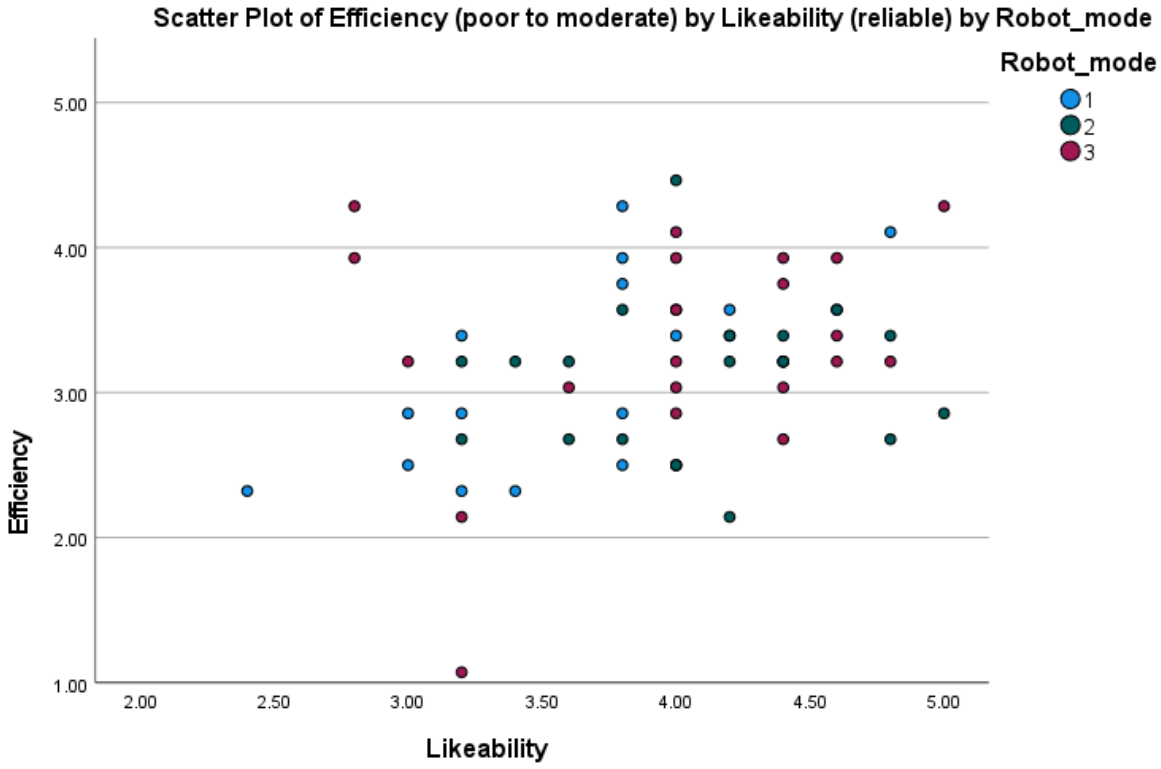


Figure C.26

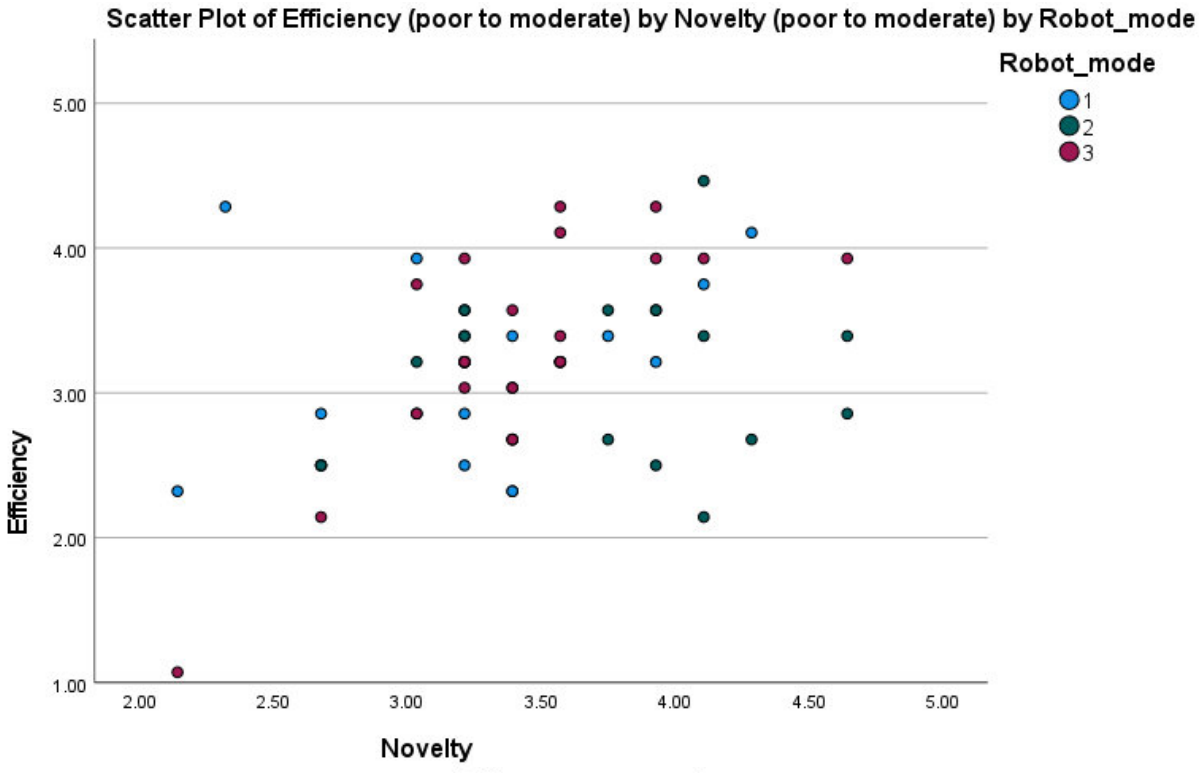


Figure C.27

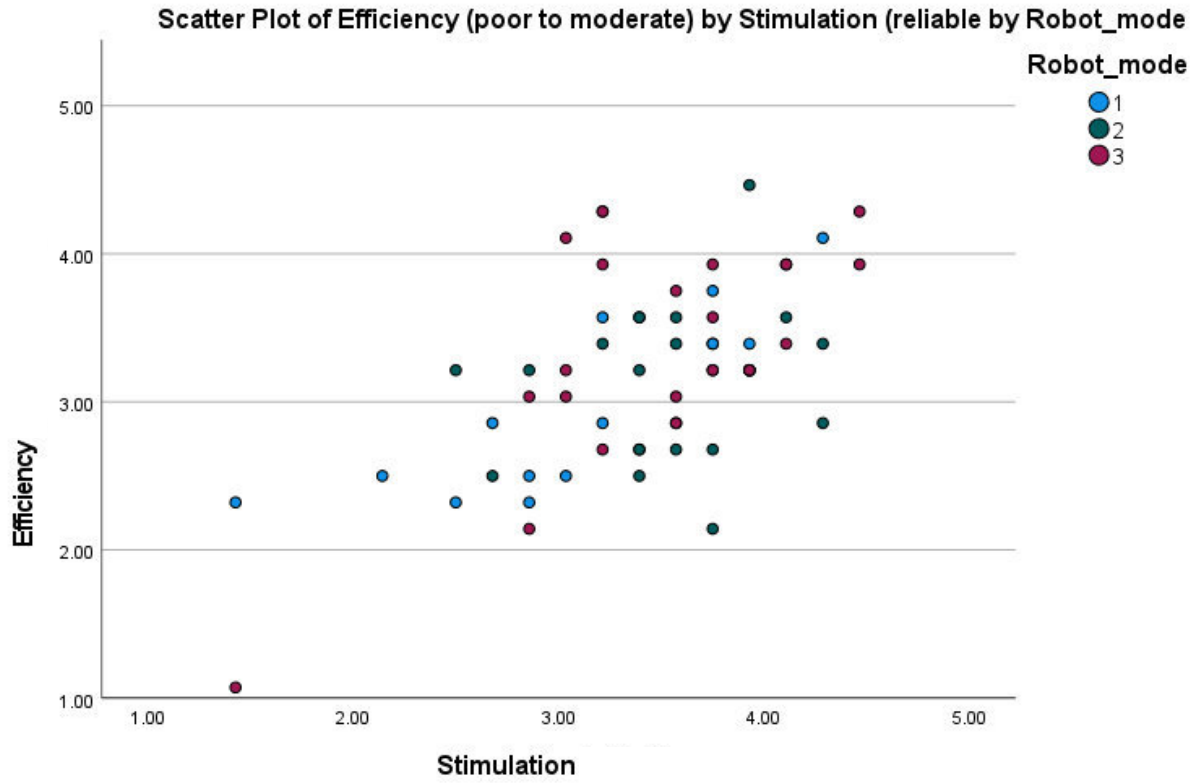


Figure C.28

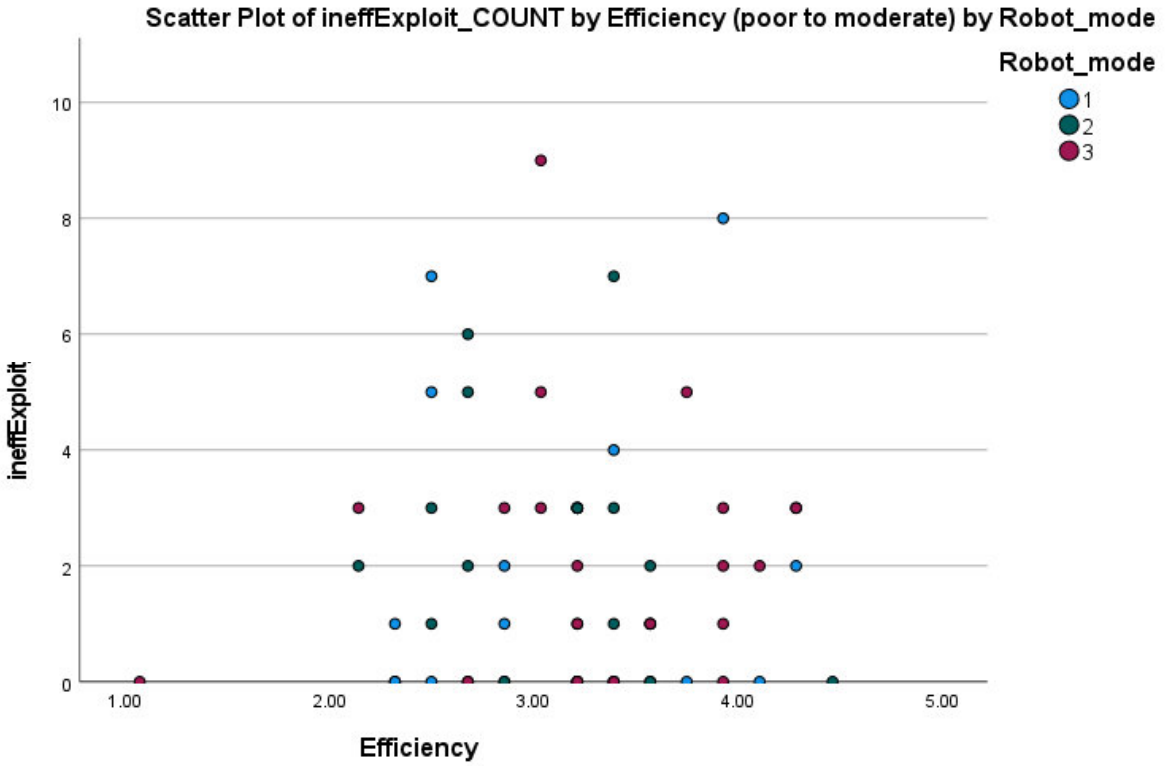


Figure C.29

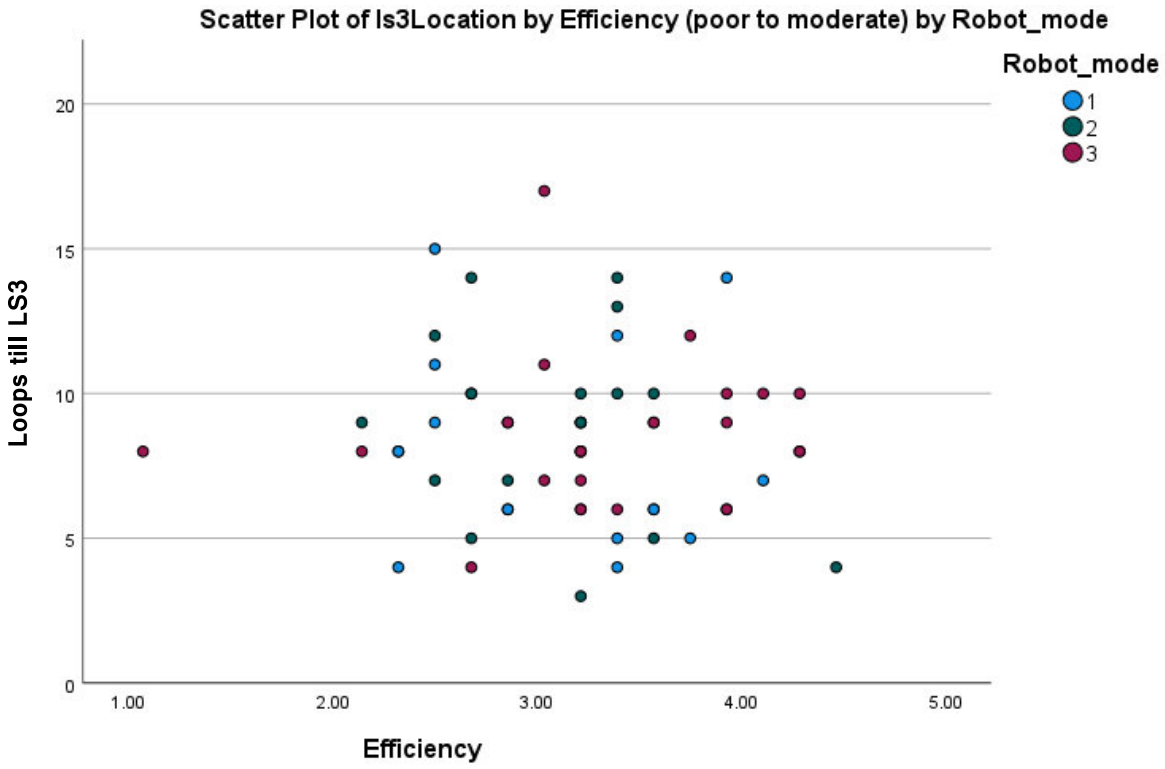


Figure C.30

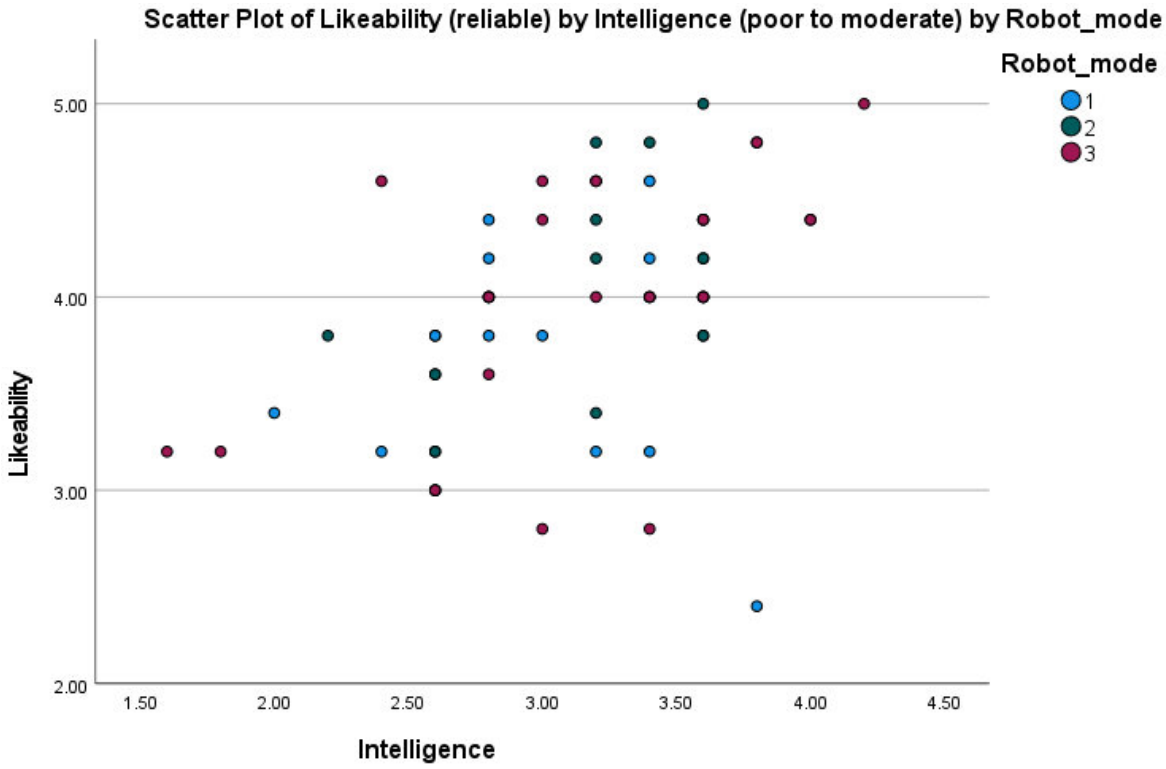


Figure C.31

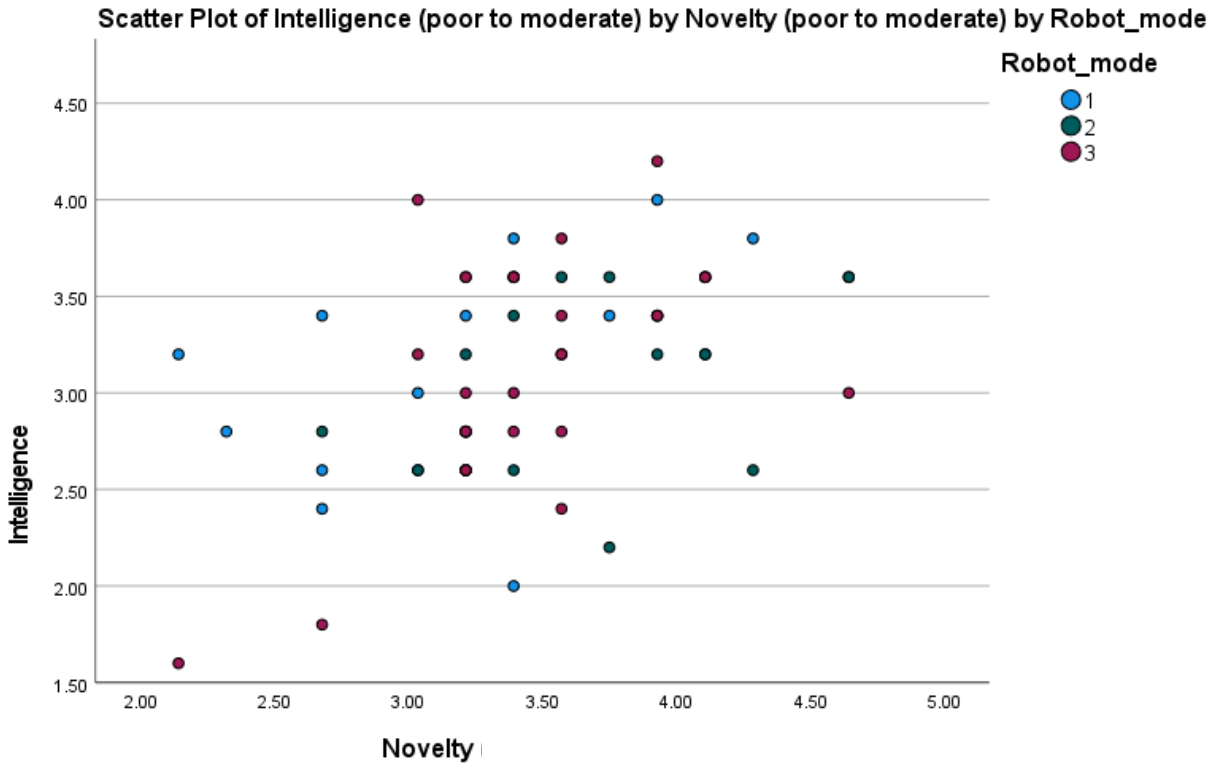


Figure C.32

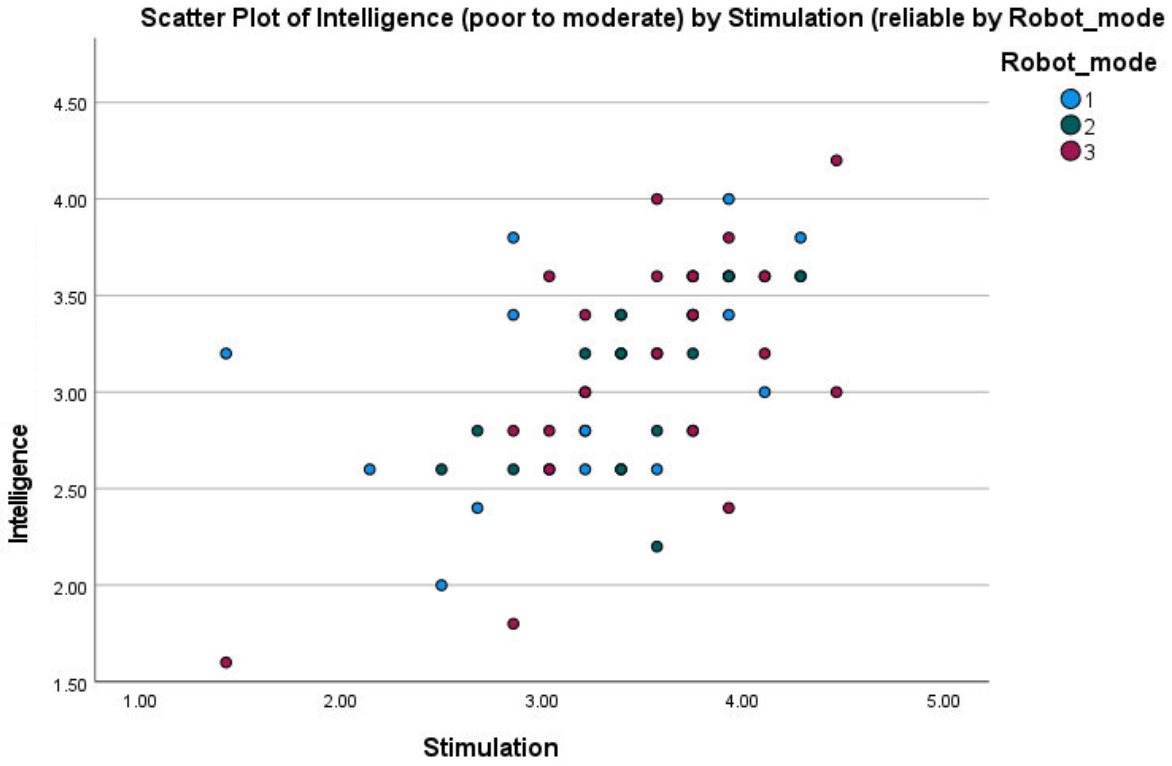


Figure C.33

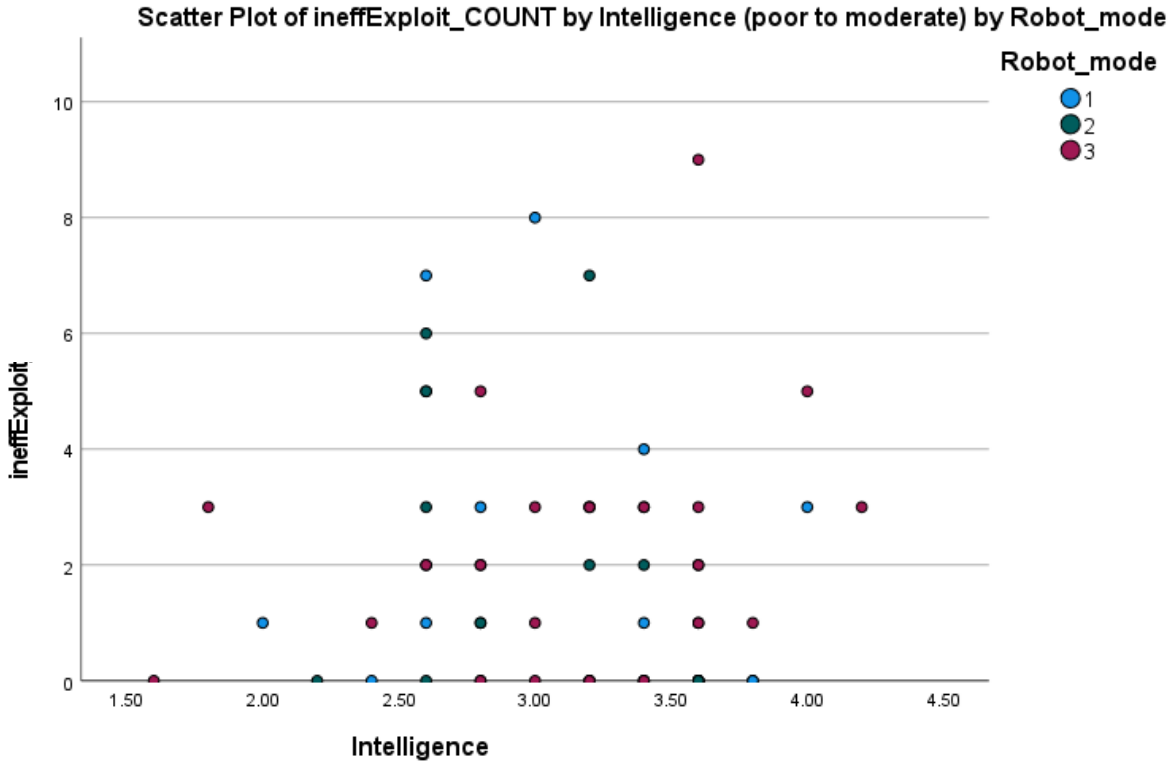


Figure C.34

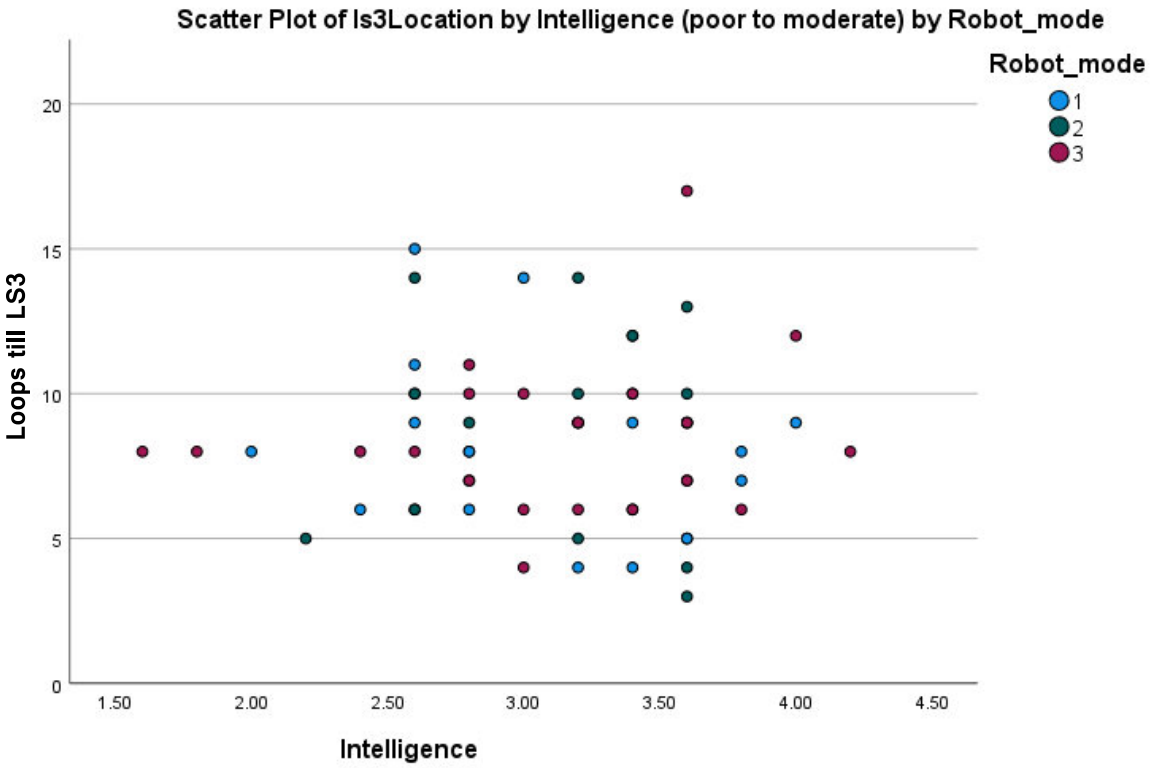


Figure C.35

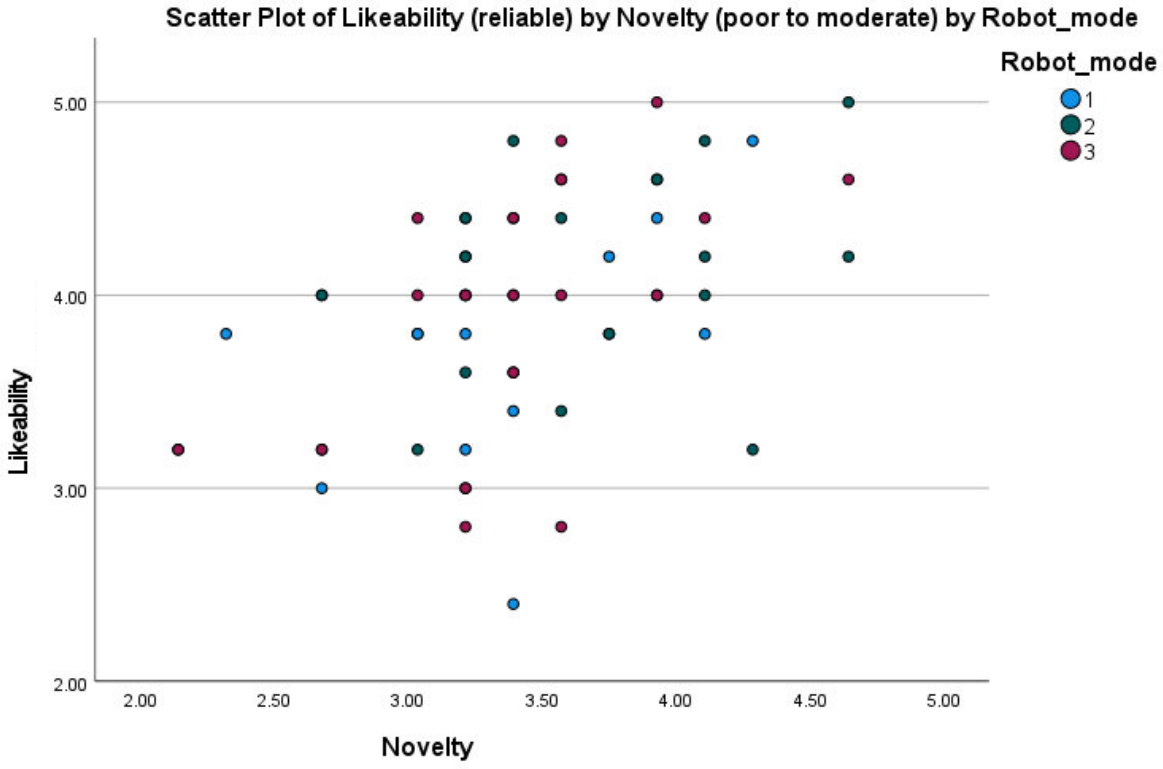


Figure C.36

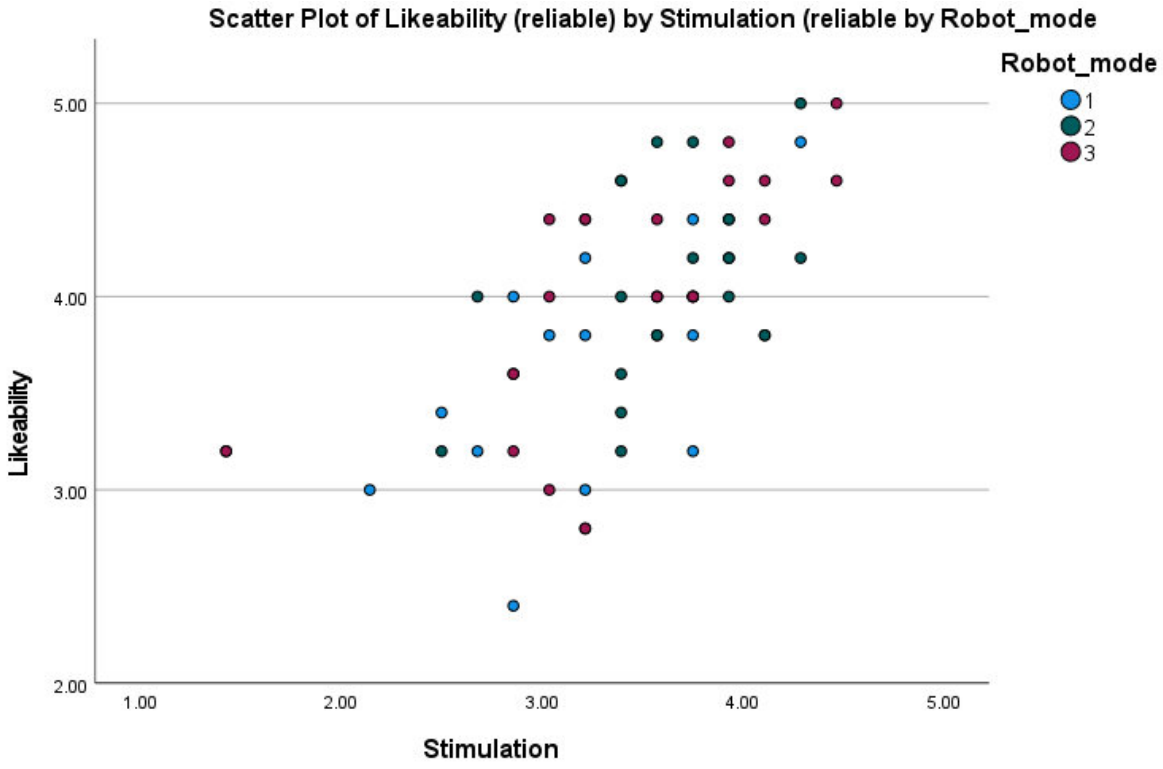


Figure C.37

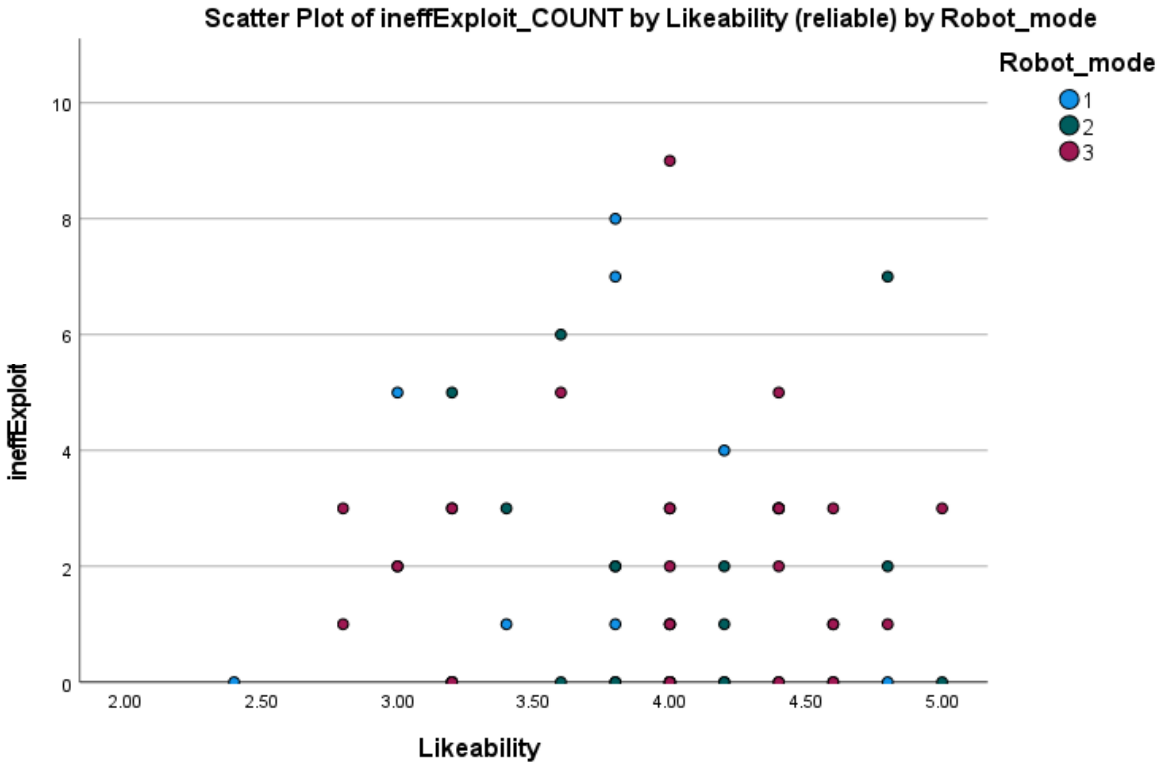


Figure C.38

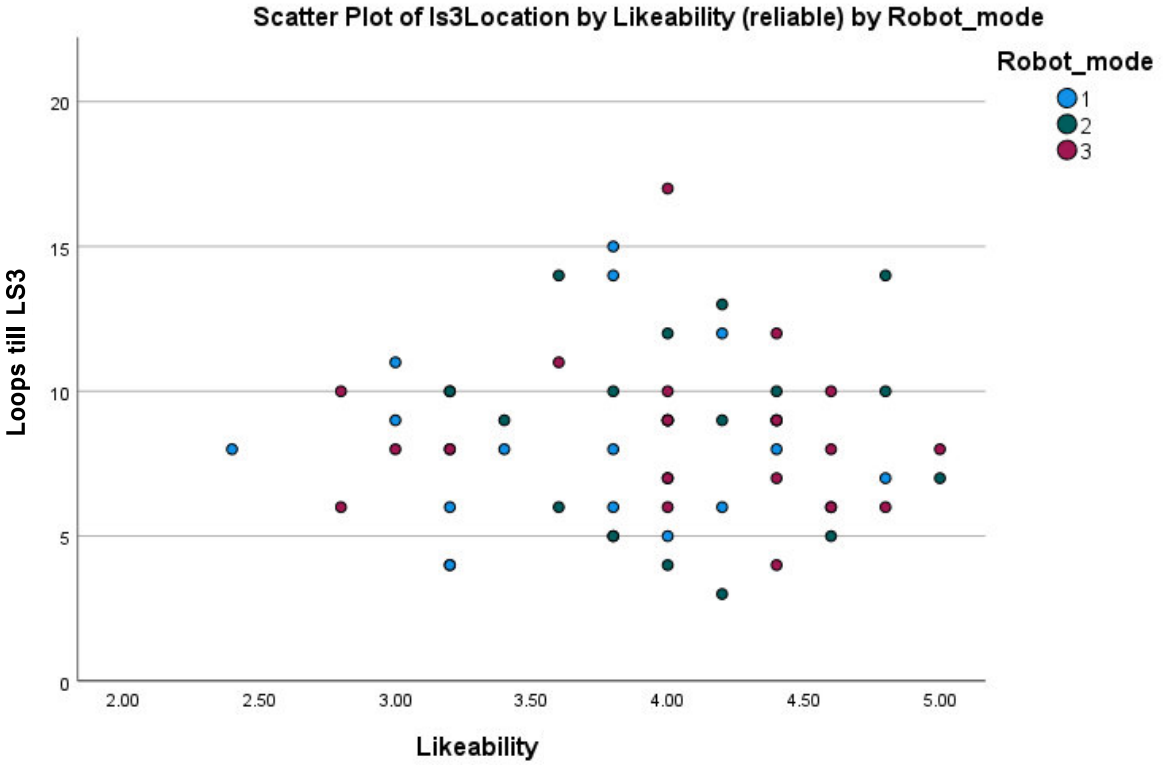


Figure C.39

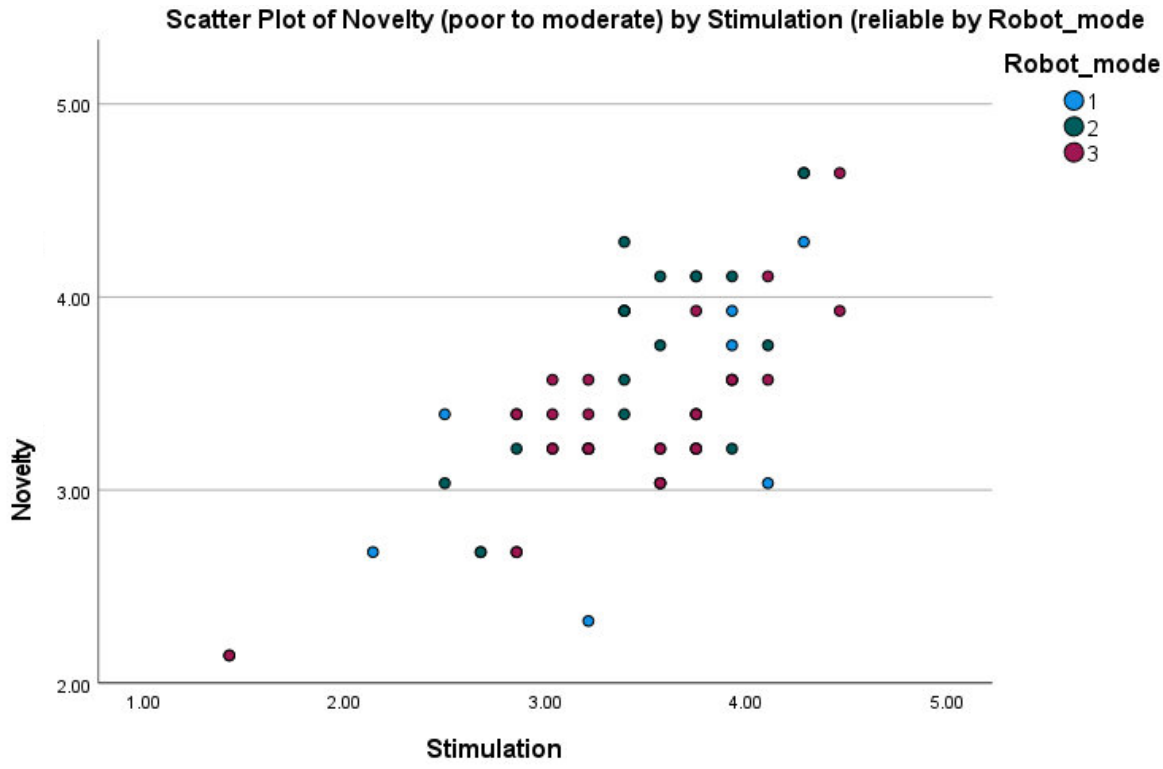


Figure C.40

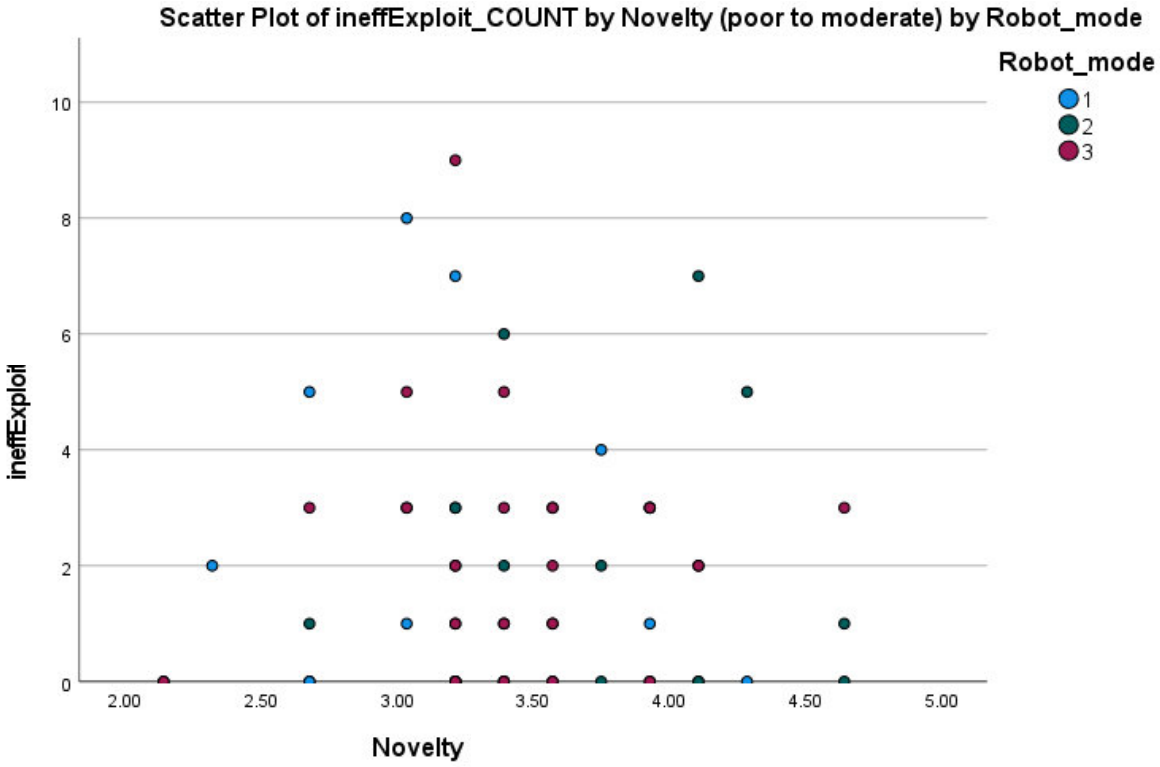


Figure C.41

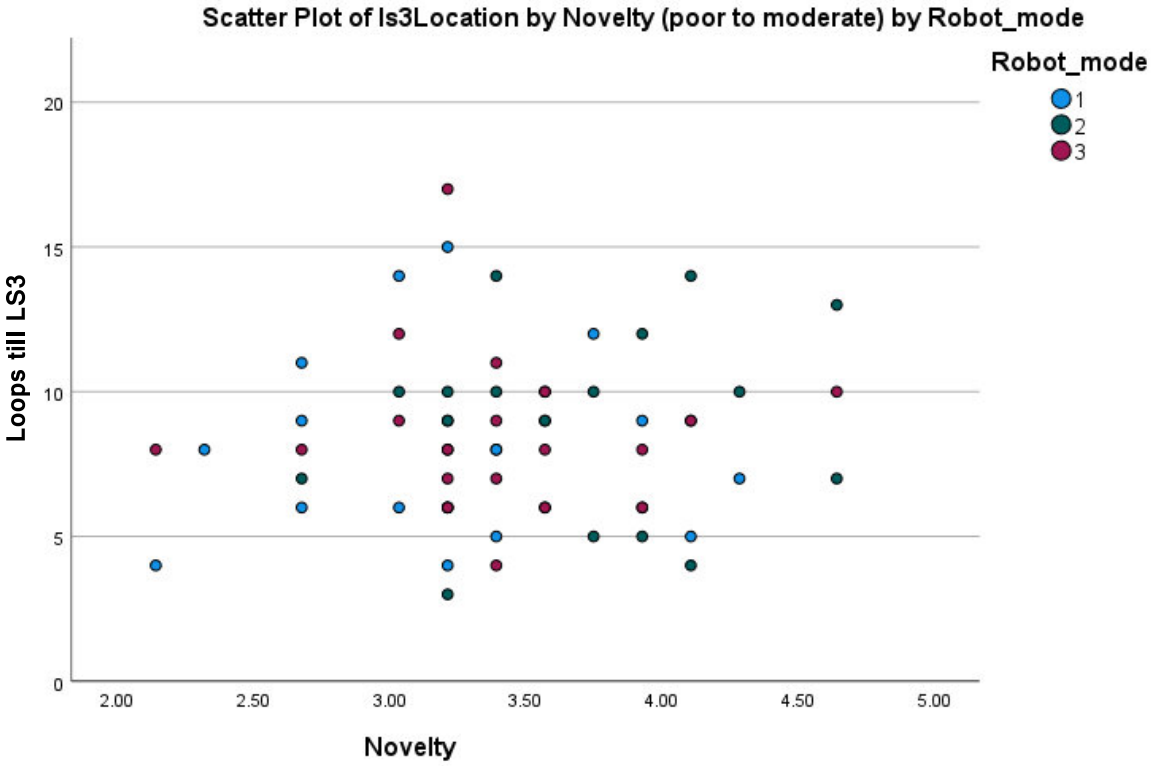


Figure C.42

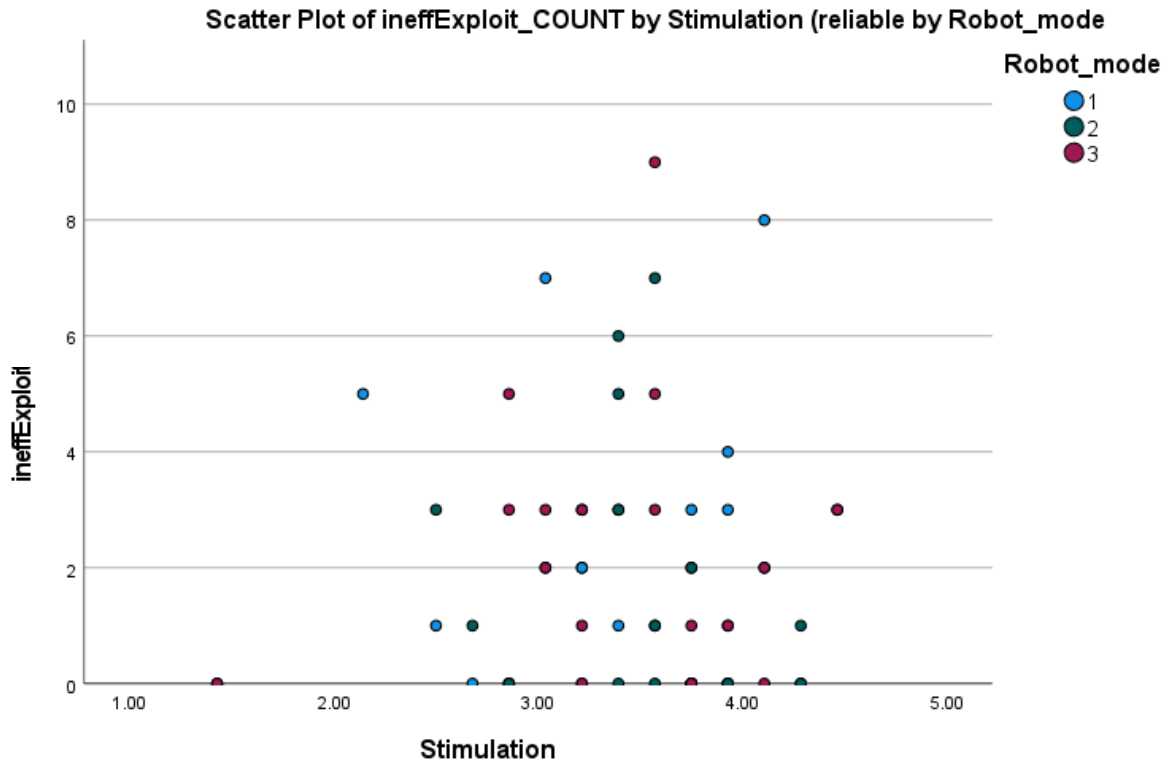


Figure C.43

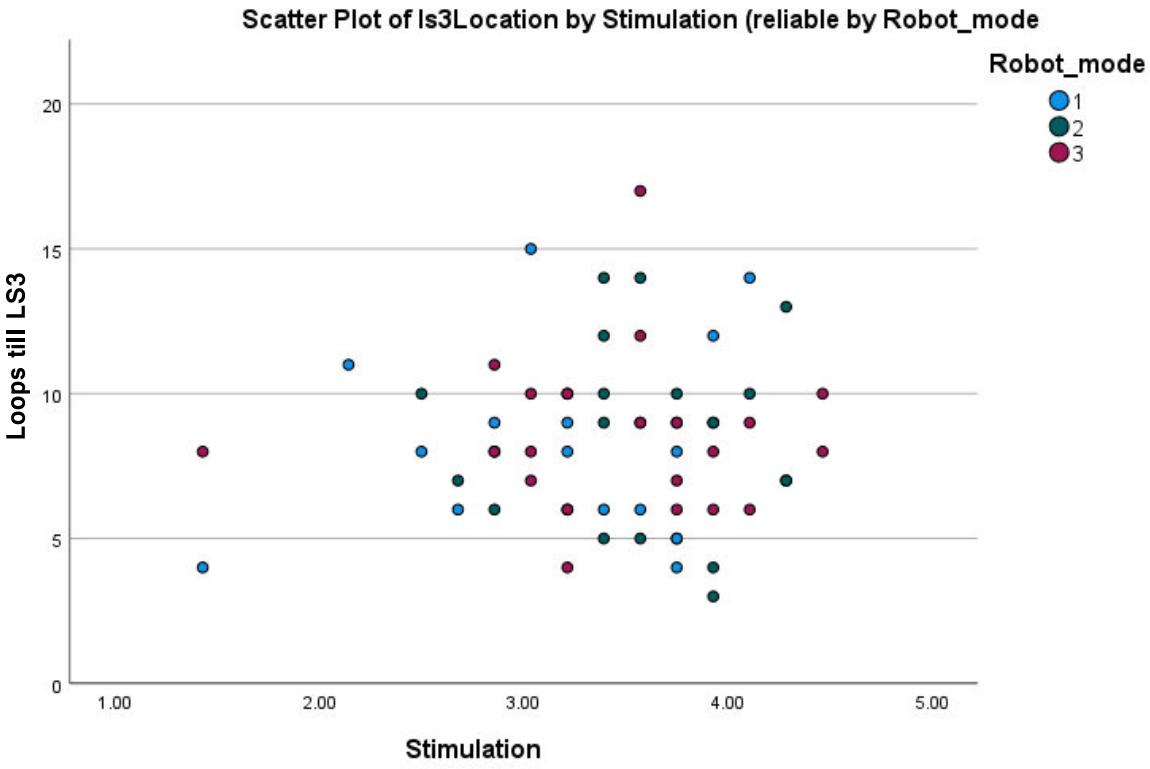


Figure C.44

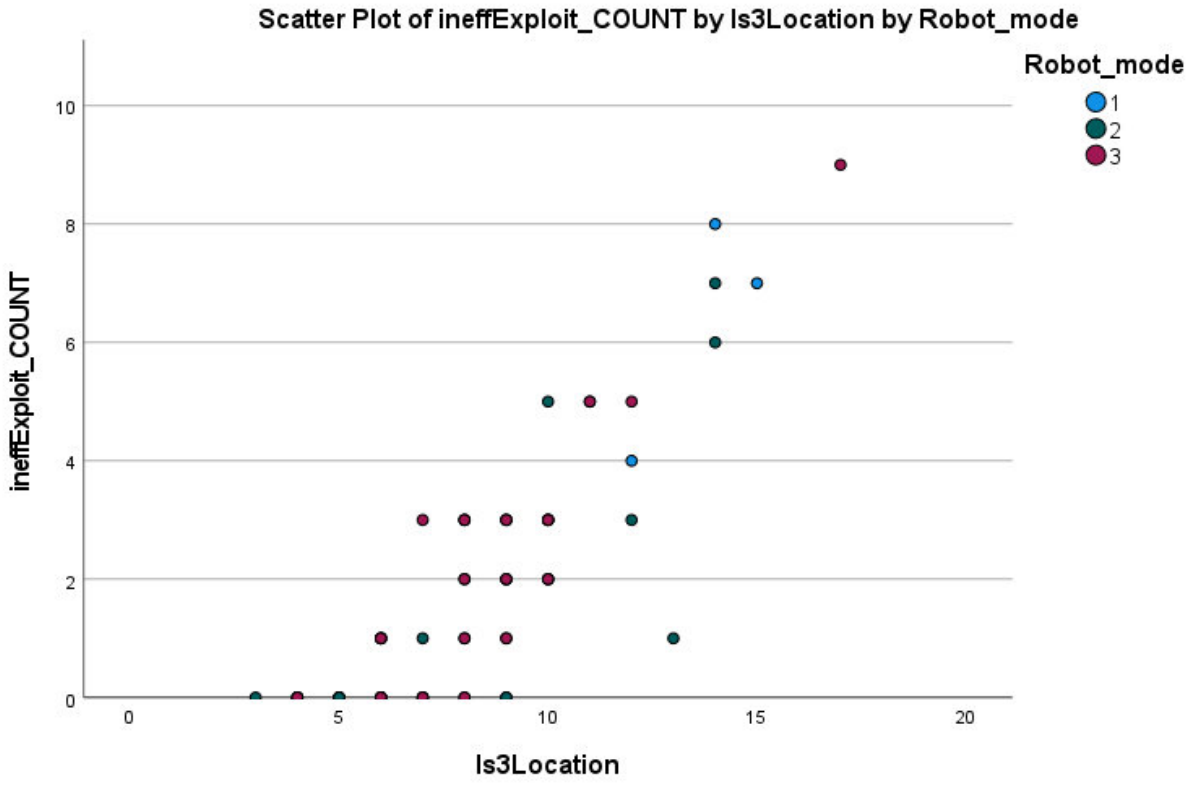

Scatter Plot of ineffExploit_COUNT by Is3Location by Robot_mode
Robot_mode
1
2
3
ineffExploit_COUNT
0
2
4
6
8
10
0
5
10
15
20
Is3Location

Figure C.45